Quantum Information and Computation, Vol. 22, No. 15&16 (2022) 1320–1350
© Rinton Press

RECONSTRUCTING BAYESIAN NETWORKS ON A QUANTUM ANNEALER

ENRICO ZARDINI
Department of Information Engineering and Computer Science
University of Trento
via Sommarive 9, 38123 Povo, Trento, Italy
enrico.zardini@unitn.it

MASSIMO RIZZOLI
Department of Information Engineering and Computer Science
University of Trento
via Sommarive 9, 38123 Povo, Trento, Italy

SEBASTIANO DISSEGNA
Department of Information Engineering and Computer Science
University of Trento
via Sommarive 9, 38123 Povo, Trento, Italy

ENRICO BLANZIERI
Department of Information Engineering and Computer Science
University of Trento
via Sommarive 9, 38123 Povo, Trento, Italy
Trento Institute for Fundamental Physics and Applications
via Sommarive 14, 38123 Povo, Trento, Italy

DAVIDE PASTORELLO
Department of Information Engineering and Computer Science
University of Trento
via Sommarive 9, 38123 Povo, Trento, Italy
Trento Institute for Fundamental Physics and Applications
via Sommarive 14, 38123 Povo, Trento, Italy

Received April 14, 2022
Revised October 26, 2022

Bayesian networks are widely used probabilistic graphical models, whose structure is hard to learn starting from the generated data. O'Gorman et al. have proposed an algorithm to encode this task, i.e., the Bayesian network structure learning (BNSL), into a form that can be solved through quantum annealing, but they have not provided an experimental evaluation of it. In this paper, we present (i) an implementation in Python of O'Gorman's algorithm, (ii) a divide et impera approach that allows addressing BNSL problems of larger sizes in order to overcome the limitations imposed by the current architectures, and (iii) their empirical evaluation. Specifically, several problems with an increasing number of variables have been used in the experiments. The results have shown the effectiveness of O'Gorman's formulation for BNSL instances of small sizes, and the superiority of the divide et impera approach on the direct execution of O'Gorman's algorithm.

Keywords: Bayesian Network Structure Learning, Quantum Annealing, Quantum Software, Empirical Evaluation

1 Introduction

Bayesian networks (BNs) are graphical probabilistic models in which the joint density distribution of multiple random variables is represented over a directed acyclic graph [1]. In detail, each variable corresponds to a node of the graph, and the overall joint density distribution is obtained by multiplying the conditional density distribution of each variable given its parents on the graph. As a consequence, the topology of the graph defines the independence conditions, i.e., a variable is independent of its non-descendants given its parents. BNs are widely used for representing uncertain domains and their structure allows for probabilistic reasoning. Obtaining a BN representation from data is a learning task with a long history; in particular, the subtask of learning the topology, also known as BN reconstruction, has received much attention [2], especially when BNs are used to represent causal relationships [3]. Moreover, some recent papers have dealt with the possible application of quantum computing to Bayesian networks [4, 5, 6].

Quantum computing (QC) is a kind of computation that exploits quantum mechanical phenomena for information processing, and, nowadays, working quantum computers are available on the market [7]. QC will have an impact on artificial intelligence and machine learning. Indeed, it has the potentiality to allow efficient solutions to many of the search and optimization problems encountered in these fields. In the last years, some applications of quantum computing to Bayesian networks have been proposed, such as the following: a method for learning the structure of a BN using a quantum annealer [4]; an algorithm for Bayesian inference based on amplitude amplification, which is a quantum version of the classical rejection sampling algorithm used for inference in Bayesian networks [5]; a systematic method for designing a quantum circuit to represent a generic discrete BN [6]. In particular, the proposal of O'Gorman et al. [4] considers a quantum annealing architecture instead of a gate-based quantum computer. Quantum annealing is a type of heuristic search for solving optimization problems by finding the low-energy states of a quantum system [8], and quantum annealers are non-universal specific-purpose quantum computers implementing quantum annealing. The advantage of the existing quantum annealers lies in the high number of qubits w.r.t. the available prototypes of general-purpose quantum computers. The paper by O'Gorman et al. describes an effective encoding of the BN reconstruction problem into a quantum annealer architecture. However, no implementation and empirical evaluation on a real quantum machine are provided.

In this paper, we present an empirical evaluation of the proposal of O'Gorman et al. in order to assess its practical applicability using the available architectures. Since the problem encoding and the subsequent embedding in the quantum architecture limit the direct application to around 18 Bayesian variables (at time of writing), we also propose a *divide et impera* approach to overcome this limitation. Both the original algorithm and the new scheme have been tested on different problems with a growing number of variables. The code is available under the GPLv2 licence [9, 10].

The paper is organized as follows: Section 2 provides some background information; Section 3 describes the implementation of O'Gorman's algorithm [4]; Section 4 presents the divide et impera approach; Section 5 is devoted to the empirical evaluation; Section 6 contains the concluding remarks.

2 Background

This section provides information about QUBO problems, quantum annealing and D-Wave, the embedding into quantum processing units (QPUs), the Bayesian network structure learning problem, and O'Gorman's QUBO algorithm [4] to address it.

2.1 QUBO problems

Quadratic Unconstrained Binary Optimization (QUBO) problems are optimization problems of the form

$$\arg\min_x x^T Q x \tag{1}$$

where x is a binary vector, and Q is an upper triangular (or symmetric) matrix of real values. In particular, let x be an $n \times 1$ vector and Q a $n \times n$ upper triangular matrix; then, it is possible to rewrite the QUBO problem as follows:

$$\begin{aligned} x^T Q x &= \sum_{i=1}^{n} q_{ii} x_i^2 + \sum_{i=1}^{n} \sum_{j=i+1}^{n} q_{ij} x_i x_j \\ &= \sum_{i=1}^{n} q_{ii} x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} q_{ij} x_i x_j \end{aligned} \tag{2}$$

where $x_i^2 = x_i$ since $x_i \in \mathbb{B} = \{0, 1\}$. In practice, the main diagonal of Q contains the linear coefficients (q_{ii}), whereas the rest of the matrix contains the quadratic ones (q_{ij}). Although QUBO problems are unconstrained by definition, it is actually possible to introduce constraints by representing them as penalties. Several examples are provided by Glover et al. [11].

The significance of the QUBO formulation mainly lies in its computational equivalence with the Ising model, which is the physical model upon which annealers are built. The only difference is the domain of variables: $\{0, 1\}$ for the QUBO formulation, $\{-1, +1\}$ for the Ising one. Hence, by applying a trivial conversion, it is possible to exploit quantum annealers to solve problems expressed as QUBO.

2.2 Quantum annealing and D-Wave machine

Quantum annealing (QA) is a heuristic search used to solve optimization problems [8]. The solution of a given problem corresponds to the *ground state* (the less energetic physical state) of a n-qubit system with energy described by a *problem Hamiltonian* H_P, which is a Hermitian $2^n \times 2^n$ matrix. The annealing procedure is implemented by a time evolution of the quantum system towards the ground state of the problem Hamiltonian. More precisely, let us consider the time-dependent Hamiltonian

$$H(t) = \Gamma(t) H_D + H_P, \tag{3}$$

where H_P is the problem Hamiltonian, and H_D is the *transverse field Hamiltonian*, which gives the kinetic term inducing the exploration of the solution landscape by means of quantum fluctuations. Γ is a decreasing function that attenuates the kinetic term, driving the system towards the global minimum of the problem landscape represented by H_P.

QA can be physically realized by considering a quantum spin glass, which is a network of qubits arranged on the vertices of a graph $\langle V, E\rangle$, with $|V| = n$ and whose edges E represent the couplings among the qubits. The problem Hamiltonian is defined as

$$H_P = H(\mathbf{\Theta}) = \sum_{i \in V} \theta_i \sigma_z^{(i)} + \sum_{(i,j)\in E} \theta_{ij} \sigma_z^{(i)} \sigma_z^{(j)}, \tag{4}$$

where the real coefficients θ_i, θ_{ij} are arranged into the matrix $\mathbf{\Theta}$. $H(\mathbf{\Theta})$ is an operator on the n-qubit Hilbert space $\mathsf{H} = (\mathbb{C}^2)^{\otimes n}$, whereas $\sigma_z^{(i)}$ acts as the Pauli matrix

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{5}$$

on the ith tensor factor and as the 2×2 identity matrix on the other tensor factors. Regarding the coefficient matrix $\mathbf{\Theta}$, it is the $n \times n$ symmetric square matrix of real coefficients of E (called *weights*) defined as

$$\mathbf{\Theta}_{ij} := \begin{cases} \theta_i, & i = j, \\ \theta_{ij}, & (i,j) \in E, \\ 0, & (i,j) \notin E, \end{cases} \tag{6}$$

with θ_i physically corresponding to the local field on the ith qubit, and θ_{ij} to the coupling between the qubits i and j. In particular, the Pauli matrix σ_z has two eigenvalues $\{-1, 1\}$, which correspond to the binary states, *spin down* and *spin up*, of each qubit. Thus, the spectrum of eigenvalues of the problem Hamiltonian (Eq. 4) is the set of all possible values of the cost function given by the energy of the well-known *Ising model*:

$$\mathsf{E}(\mathbf{\Theta}, \boldsymbol{z}) = \sum_{i \in V} \theta_i z_i + \sum_{(i,j)\in E} \theta_{ij} z_i z_j, \quad \boldsymbol{z} = (z_1, ..., z_n) \in \{-1, 1\}^{|V|}. \tag{7}$$

In practice, the annealing procedure, also called *cooling*, drives the system into the ground state of $H(\Theta)$, which corresponds to the spin configuration encoding the solution:

$$\boldsymbol{z}^* = \underset{\boldsymbol{z} \in \{-1,1\}^{|V|}}{\arg\min}\ \mathsf{E}(\mathbf{\Theta}, \boldsymbol{z}). \tag{8}$$

Given a problem, the annealer is initialized using a suitable choice of the weights $\mathbf{\Theta}$, and the binary variables $z_i \in \{-1, 1\}$ are physically realized by the outcomes of the measurements performed on the qubits located in the vertices V. In order to solve a general optimization problem through QA, it is first necessary to find an *encoding* of the objective function in terms of the cost function (7), which is not easy in general.

D-Wave Systems is a Canadian company producing quantum annealers, i.e., physical machines implementing the quantum annealing process. Currently, the available models are the D-Wave 2000Q, exploiting the *Chimera* topology, and the D-Wave Advantage, featuring the *Pegasus* topology. The former has 2048 qubits, each connected to 6 other qubits, whereas the latter has 5640 qubits, each connected to 15 other qubits. A higher amount of qubits allows for larger problems to be submitted, but the most relevant feature is the connectivity, which determines the complexity of the representable problems. For these reasons, the D-Wave Advantage has been chosen for the experiments.

2.3 Quantum processing unit (QPU) embedding

To practically use quantum annealing for solving QUBO problems, the problem variables must be mapped to the QPU qubits. However, due to the sparseness of the available annealer topologies, a direct representation of the problem is typically not possible. The solution consists in chaining together multiple physical qubits that will act as a single logical qubit. In this way, the connectivity of the annealer graph is increased at the price of reducing the number of logical qubits available and, consequently, the size of representable problems. The entire process is known as embedding or *minor embedding* (in the glossary of D-Wave) [12, 13]. In particular, D-Wave's Ocean library provides the *EmbeddingComposite* class [14] to automatically perform the minor embedding of the supplied QUBO matrices, and a new embedding is computed for every annealer read.

2.4 Bayesian network structure learning (BNSL)

A Bayesian network (BN) is a directed acyclic graph (DAG) representing the conditional dependencies of a set of random variables. In particular, the nodes of the graph represent the variables, whereas the edges represent the conditional dependencies between them. Moreover, each node is associated with the conditional probability distribution of the node itself given its parents.

The method proposed by O'Gorman et al. [4] focuses on the network structure learning (BNSL) problem, which consists in finding the Bayesian network that most likely has generated a given set of data. The problem is NP-Complete [15], and the authors expect a polynomial speedup using quantum annealing. In detail, to take advantage of the new technology, a hardware compatible QUBO formulation of the BNSL problem is provided in the paper together with sufficient lower bounds for the penalties.

More formally, a Bayesian network can be defined as a pair (B_s, B_p), where B_s is a DAG and B_p is the set of associated conditional probabilities. Then, given a database $D = \{\mathbf{x}_i | 1 \leq i \leq N\}$ with $\mathbf{x}_i$ representing the state of all variables, the objective consists in finding the structure that maximises the posterior probability distribution $p(B_s|D)$. However, due to the proportionality of $p(B_s|D)$ and $p(D|B_s)$ by Bayes' theorem, it is possible to reformulate the problem as maximizing $p(D|B_s)$, which is given by

$$p(D|B_s) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})}, \tag{9}$$

where Γ is the gamma function, q_i is the number of joint states of the parent set of the i-th random variable, r_i is the number of states of the random variable itself, N_{ijk} is the number of occurrences in D with the i-th random variable in its k-th state and the variable's parent set in its j-th state, α_{ijk} is the hyperparameter of the assumed Dirichlet prior for the node's conditional probability distribution, N_{ij} and α_{ij} are the sums of the corresponding parameter values over k.

2.4.1 QUBO formulation of BNSL

In their work [4], O'Gorman et al. provide a Hamiltonian function for the BNSL problem. Given the BNSL Hamiltonian, the construction of the QUBO matrix is straightforward: it is sufficient to map the coefficients of the variables into the matrix entries. In particular, the BNSL Hamiltonian consists of three components: the score Hamiltonian (H_{score}), which is responsible for evaluating the quality of the solution graph; the max Hamiltonian (H_{max}), which is in charge of penalising the solutions including nodes with a number of parents greater than m, a constraint dictated by resource limits; the cycle Hamiltonian (H_{cycle}), further divided in consistency Hamiltonian ($H_{consist}$) and transitivity Hamiltonian (H_{trans}), which penalises the solutions containing cycles. Hence, the full Hamiltonian (H) is given by

$$H(\mathbf{d}, \mathbf{y}, \mathbf{r}) = H_{score}(\mathbf{d}) + H_{max}(\mathbf{d}, \mathbf{y}) + H_{cycle}(\mathbf{d}, \mathbf{r}), \tag{10}$$

where $\mathbf{d}$ corresponds to the $n(n-1)$ bits used to represent the presence/absence of edges between nodes, whereas $\mathbf{y}$ and $\mathbf{r}$ are additional variables exploited to encode the constraints.

Score Hamiltonian

The score Hamiltonian (H_{score}) is calculated separately for each variable, and the components are then summed together. In detail, the score Hamiltonian for the i-th variable is given by

$$H^{(i)}_{score}(\mathbf{d}_i) = \sum_{\substack{J \subset \{1..n\} \setminus \{i\} \\ |J| \le m}} \left(w_i(J) \prod_{j \in J} d_{ji} \right), \tag{11}$$

where $\mathbf{d}_i$ includes all the bits (d_{ji}) encoding edges towards the considered node, m is the largest allowed size for the parent set, and w_i is computed as follows:

$$w_i(J) = \sum_{l=0}^{|J|} (-1)^{|J|-l} \sum_{\substack{K \subset J \\ |K| = l}} s_i(K), \tag{12}$$

with s_i being a score value obtained from Eq. (9), introducing a logarithm for numerical efficiency. Specifically, s_i is given by

$$s_i(\Pi_i(B_s)) = -\log \left(\prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})} \right), \tag{13}$$

where $\Pi_i(B_s)$ denotes the parent set of the i-th node. In practice, the sum of the s_i values is equal to $-\log p(D|B_s)$.

Max Hamiltonian

Analogously to the score Hamiltonian, the max Hamiltonian is computed separately for each variable as

$$H_{max}^{(i)}(\mathbf{d}_i, \mathbf{y}_i) = \delta_{max}^{(i)}(m - d_i - y_i)^2, \tag{14}$$

where $\delta_{max}^{(i)} > 0$ is the penalty weight, d_i is the in-degree of the considered node (given by $\sum_{1\le j\le n \,\cap\, j\ne i} d_{ji}$), and $y_i \in \mathbb{Z}$ is a slack variable (encoded via binary expansion in $\mathbf{y}_i$ using μ bits [a]) that allows $H_{max}^{(i)}$ being zero when the constraint is satisfied. Indeed, $H_{max}^{(i)}$ is zero if the considered node has at most m parents, otherwise it carries a positive penalty.

Cycle Hamiltonian

As mentioned previously, the cycle Hamiltonian is defined as the sum of two components:

$$H_{cycle}(\mathbf{d}, \mathbf{r}) = H_{trans}(\mathbf{r}) + H_{consist}(\mathbf{d}, \mathbf{r}), \tag{15}$$

where $\mathbf{r}$ represents $n(n-1)/2$ additional boolean variables encoding a topological order (r_{ij} is 1 if the i-th node precedes the j-th one, 0 otherwise). In detail, the transitivity Hamiltonian penalises the cycles of length three in the r_{ij} values, and is computed separately for each possible 3-set of variables as

$$H_{trans}^{(ijk)}(r_{ij}, r_{jk}, r_{ik}) = \delta_{trans}^{(ijk)}\left(r_{ik} + r_{ij}r_{jk} - r_{ij}r_{ik} - r_{jk}r_{ik}\right), \tag{16}$$

where $\delta_{trans}^{(ijk)}$ is the positive penalty added if the i-th, j-th and k-th variables form a 3-cycle. As in the previous cases, the $H_{trans}^{(ijk)}$ components are summed up to obtain the full H_{trans}. Instead, the consistency Hamiltonian penalises the solutions for which the topological order contained in $\mathbf{r}$ is inconsistent with the graph structure encoded in $\mathbf{d}$. In practice, it makes disadvantageous the solutions in which $r_{ij} = 1$ and $d_{ji} = 1$, or $r_{ij} = 0$ and $d_{ij} = 1$. The Hamiltonian is computed separately for each pair of variables as

$$H_{consist}^{(ij)}(d_{ij}, d_{ji}, r_{ij}) = \delta_{consist}^{(ij)}(d_{ji}r_{ij} + d_{ij} - d_{ij}r_{ij}), \tag{17}$$

where $\delta_{consist}^{(ij)}$ is the positive penalty associated with the inconsistency. It is also worth highlighting that the penalties $\delta_{trans}^{(ijk)}$ and $\delta_{consist}^{(ij)}$ are invariant to the permutation of the superscript indices (the set of variables remains the same).

2.4.2 QUBO size and penalty values

The QUBO formulation of the BNSL problem consist of $n(n-1)$ binary variables (d_{ij}) encoding the graph structure, $n\mu = n\lceil\log_2(m+1)\rceil$ binary slack variables (y_{il}) related to the maximum parent constraint, and $n(n-1)/2$ binary variables (r_{ij}) encoding a topological order (related to the absence of cycles constraint). Hence, the QUBO encoding of n Bayesian variables requires $\mathcal{O}(n^2)$ binary variables. Nevertheless, since H_{score} contains multiplications with m factors, if $m \ge 3$, additional steps and slack variables are needed to convert the

[a] $y_i = \sum_{l=1}^{\mu} 2^{l-1} y_{il}$, with $\mu = \lceil\log_2(m+1)\rceil$

problem into a quadratic equation. For instance, according to O'Gorman et al., $n\lfloor\frac{(n-2)^2}{4}\rfloor$ binary slack variables are needed to reduce a BNSL problem with $m = 3$ to a quadratic form, increasing the total number of binary variables to $\mathcal{O}(n^3)$. In this work, only the formulation for $m = 2$ has been used in the experiments, although some problems taken into account have more than two parents per node.

Concerning the penalty values, O'Gorman et al. provide the following lower bounds (they have also demonstrated their sufficiency):

$$\delta_{max}^{(i)} > \max_{j \neq i} \Delta_{ji},\ 1 \leq i \leq n, \tag{18}$$

$$\delta_{consist}^{(ij)} > (n-2) \max_{k \notin \{i,j\}} \delta_{trans}^{(ijk)},\ 1 \leq i < j \leq n, \tag{19}$$

$$\delta_{trans}^{(ijk)} = \delta_{trans} > \max_{\substack{1 \leq i',j' \leq n \\ i' \neq j'}} \Delta_{i'j'},\ 1 \leq i < j < k \leq n, \tag{20}$$

where Δ_{ji} is an estimate of the largest increase in score due to the insertion of an arc from the j-th to the i-th node. In the case of $m = 2$, it is given by

$$\Delta_{ji} = \max\{0, \Delta'_{ji}\}, \tag{21}$$

$$\Delta'_{ji} = -w_i(\{j\}) - \sum_{\substack{1 \leq k \leq n \\ k \neq i,j}} \min\{0, w_i(\{j,k\})\}. \tag{22}$$

Instead, for $m \geq 3$, computing Δ_{ji} becomes an intractable optimization problem.

3 O'Gorman's Algorithm Implementation

A Python implementation of O'Gorman's algorithm, which provides a way to build the QUBO matrix for a BNSL problem, has been developed in this work. Since the D-Wave's Ocean suite, which is necessary for interacting with the quantum annealer, is implemented in Python, the programming language in question has turned out to be the most reasonable choice. This section provides the implementation details, some considerations about the complexity of the implementation, and the description of a method to speed up the execution.

3.1 QUBO matrix construction

The pseudocode of the implementation of O'Gorman's algorithm is shown in Algorithm 1, which includes calls to Algorithms 2-3-4. In particular, the main algorithm takes as input the number of Bayesian variables n, the number of states r_i for each variable, and the dataset of examples, and produces as output the QUBO matrix Q that represents the considered BNSL problem.

Before effectively building the matrix, it is necessary to compute several intermediate values. First, all possible parent sets ($\mathit{\Pi}_i(B_s)$ in O'Gorman's formulation) are calculated for each Bayesian variable. The maximum number of parents m has been set to two (for the reasons explained in Section 2.4.2) and, as a consequence, the complexity of this step turns out to be $\mathcal{O}(n^3)$. It is also worth noticing that the empty set is a valid parent set.

After that, the α_{ijk} hyperparameters of the Dirichlet priors are set to the uninformative value $1/(r_i \cdot q_i)$, with r_i being the number of states of the i-th variable and q_i (denoted as

```
Input: number of Bayesian variables $n$, list $r = (r_i)_{i=1}^n$ with $r_i$ being the number of states of the $i^{th}$
       variable, dataset $examples$
Result: QUBO matrix $Q$
// calculation of the values needed to construct Q
1  $parentSets \leftarrow calcParentSets(n)$;
2  $\alpha \leftarrow calcAlpha(n, r, parentSets)$;
3  $s \leftarrow calcS(n, r, parentSets, \alpha, examples)$;                              // Algorithm 2
4  $w \leftarrow calcW(n, parentSets, s)$;                                                // Algorithm 3
5  $\Delta \leftarrow calcDelta(n, parentSets, w)$;                                        // Eq. (21) and (22)
6  $\delta_{max} \leftarrow calcDeltaMax(n, \Delta)$;                                      // Eq. (18)
7  $\delta_{trans} \leftarrow calcDeltaTrans(n, \Delta)$;                                  // Eq. (20)
8  $\delta_{consist} \leftarrow calcDeltaConsist(n, \delta_{trans})$;                      // Eq. (24)
// construction of Q
9  $Q \leftarrow zeroMatrix()$;
10 $Q \leftarrow fillQ(Q, n, parentSets, w, \delta_{max}, \delta_{trans}, \delta_{consist})$;   // Algorithm 4
11 return $Q$;
```

Algorithm 1: *calcQUBOMatrix(n, r, examples)*

$q_{i\pi}$ in the pseudocode) being the number of states of the considered parent set. In practice, all α_{ijk} related to a specific variable i and parent set π (denoted as $\alpha_{i\pi jk}$ in the pseudocode) have the same value; further details about this choice are provided in Section 5.4. In this step, the α_{ijk} value for all possible "variable" - "parent set" - "parent set state" - "variable state" combinations must be generated; hence, the complexity is $\mathcal{O}(n^3 r_{max}^3)$, where r_{max} is the maximum number of states of the Bayesian variables.

The next step consists in computing the local scores s for all possible "Bayesian variable" - "parent set" combinations according to Eq. (13). Nevertheless, due to the factorial nature of the Γ function and the presence of multiplications of Γ values, the calculations turn out to be feasible only for very small datasets; indeed, the values quickly go out of range. The solution lies in moving the logarithm inside through algebraic steps until its argument becomes the gamma function alone. In the implementation presented here, the natural logarithm (ln) has been used, and the resulting formula, the one employed in Algorithm 2, is the following:

$$s_i(\Pi_i(B_s)) = -\sum_{j=1}^{q_i}\Big[\ln(\Gamma(\alpha_{ij})) - \ln(\Gamma(N_{ij} + \alpha_{ij})) + \sum_{k=1}^{r_i}[\ln(\Gamma(N_{ijk} + \alpha_{ijk})) - \ln(\Gamma(\alpha_{ijk}))]\Big]. \tag{23}$$

This form allows exploiting the (natural) log-gamma function (denoted as $\ln\Gamma$ in the pseudocode) instead of the gamma one, a function characterised by a far slower growth. Moreover, by doing this, the products in Eq. (13) are replaced by additions, further decreasing the risk of out of range values. Concerning the pseudocode, the *calcNiπjk* procedure just computes the number of times the variable i is in its k-th state while its parent set π is in its j-th state (in the case of an empty parent set, the state of the i-th variable alone is considered). The complexity of the algorithm is $\mathcal{O}(n^3 N r_{max}^3)$.

Once the score values s have been calculated, it is possible to compute the parent set weights w for the score Hamiltonian according to (12). The pseudocode for this step is available in Algorithm 3. As previously mentioned, the maximum number of allowed parents has been set to two, hence the pseudocode does not take into account cases with larger parent sets. The complexity of the algorithm for the computation of w is $\mathcal{O}(n^3)$.

```
Input: number of Bayesian variables $n$, list of number of states $r$, list of parent sets $parentSets$,
       prior distributions hyperparameters $\alpha = (\alpha_{i\pi jk})$, dataset $examples$
Result: $s = (\{s_i(\pi)\ s.t.\ \pi \in parentSets[i]\})_{i=1}^{n}$ with $s_i(\pi)$ being the score for the Bayesian variable
        $i$ given the parent set $\pi$
function calcSi(i, $\pi$, r, $\alpha$, examples):                                        // Eq. (23)
    $q_{i\pi} \leftarrow \prod_{p\in\pi} r_p$;                                          // $q_{i\pi} = 1$ if $\pi = \emptyset$
    $sum \leftarrow 0$;
    for $j \leftarrow 1$ to $q_{i\pi}$ do
        $\alpha_{i\pi j} \leftarrow \sum_{k=1}^{r_i} \alpha_{i\pi jk}$;
        $N_{i\pi j} \leftarrow \sum_{k=1}^{r_i} calcNi\pi jk(examples, \pi, i, j, k, r)$;
        $sum \leftarrow sum + \ln\Gamma(\alpha_{i\pi j}) - \ln\Gamma(\alpha_{i\pi j} + N_{i\pi j})$;
        for $k \leftarrow 1$ to $r_i$ do
            $N_{i\pi jk} \leftarrow calcNi\pi jk(examples, \pi, i, j, k, r)$;
            $sum \leftarrow sum + \ln\Gamma(\alpha_{i\pi jk} + N_{i\pi jk}) - \ln\Gamma(\alpha_{i\pi jk})$;
        end
    end
    return -$sum$;
for $i \leftarrow 1$ to $n$ do
    for $\pi \in parentSets[i]$ do
        $s_i(\pi) \leftarrow calcSi(i, \pi, r, \alpha, examples))$;
    end
end
return $s$;
```

Algorithm 2: *calcS(n, r, parentSets, α, examples)*

```
Input: number of Bayesian variables $n$, list of parent sets $parentSets$, score values $s$
Result: $w = (\{w_i(\pi)\ s.t.\ \pi \in parentSets[i]\})_{i=1}^{n}$ with $w_i(\pi)$ being the weight calculated for the
        Bayesian variable $i$ given the parent set $\pi$
function calcWi(i, $\pi$, s):                                                            // Eq. (12)
    if $\pi = \emptyset$ then
        return $s_i(\emptyset)$;
    else if $size(\pi) = 1$ then
        return $s_i(\pi) - s_i(\emptyset)$;
    else if $size(\pi) = 2$ then
        $p_1,\ p_2 \leftarrow \pi[1],\ \pi[2]$;
        return $s_i(\pi) - s_i(\{p_1\}) - s_i(\{p_2\}) + s_i(\emptyset)$;
    end
for $i \leftarrow 1$ to $n$ do
    for $\pi \in parentSets[i]$ do
        $w_i(\pi) \leftarrow calcWi(i, \pi, s)$;
    end
end
return $w$;
```

Algorithm 3: *calcW(n, parentSets, s)*

Eventually, the penalty values must be calculated, starting from the auxiliary quantities Δ, which are computed according to Eq. (21) and (22) with a complexity of $\mathcal{O}(n^4)$; actually, this complexity derives from the data structure used in the code to store the parent sets (according to the formulas, the complexity would be $\mathcal{O}(n^3)$). Given Δ, all penalties can be determined. In detail, $\delta_{max}^{(i)}$ is computed for each Bayesian variable according to (18) with a resulting complexity (for all $\delta_{max}^{(i)}$) of $\mathcal{O}(n^2)$. Instead, the penalty bound related to the consistency Hamiltonian (Eq. (19)) can be simplified due to independence of δ_{trans} from its superscript indices. The outcome is the following:

$$\delta_{consist} > (n-2)\delta_{trans}. \tag{24}$$

In practice, δ_{trans} is computed according to Eq. (20) with complexity $\mathcal{O}(n^2)$ (notice that δ_{trans} is a single value). Then, $\delta_{consist}$ is calculated according to the simplified bound (Eq. (24)) with complexity $\mathcal{O}(1)$. In order to satisfy the lower bounds, the penalty values have been set to the boundary values plus one.

At this point, it is possible to fill the QUBO matrix Q as shown in Algorithm 4; the matrix, whose size has been already described in Section 2.4.2, initially contains only zeros (see line 9 of Algorithm 1, whose complexity is $O(n^4)$). In detail, the first section of Algorithm 4 (lines 2-12) is related to the score Hamiltonian, namely, to Eq. (11). For each "Bayesian variable" - "parent set" combination, the parent set weight $w_i(\pi)$ ($w_i(J)$ in O'Gorman's formulation) is added to the appropriate cell; the outermost loop, which includes almost all the pseudocode, is the one iterating on the Bayesian variables. In practice, the coefficients of the linear terms of Eq. (11), i.e., the terms involving only one QUBO variable (d_{ji}), are summed to cells of Q located on the main diagonal. Instead, the coefficients of the quadratic terms, which involve two QUBO variables ($d_{xi}d_{yi}$), contribute to cells outside the diagonal; indeed, the first variable determines the row, while the other the column. The subsequent part of the algorithm is related to the max Hamiltonian (lines 13-25), i.e., to Eq. (14). The approach used for the coefficient insertion is similar to that employed for the score Hamiltonian, however the presence of a square must be taken into account. Hence, for each Bayesian variable (outermost loop), the binary variables involved in Eq. (14) and their coefficients inside the square are determined and stored in two lists (lines 14-15). Then, based on the square expansion, the resulting linear and quadratic coefficients (which include the multiplication by $\delta^{(i)}_{max}$) are summed to the respective cells. Finally, there is the section related to the transitivity and consistency Hamiltonians (lines 26-43), namely, to Eq. (16) and (17). In this case, due to the small amount of terms in the formulas and the absence of squares, the procedure is simpler. In detail, lines 28-35 sum the coefficients given by Eq. (16) to the corresponding locations for each set of three Bayesian variables (H_{trans} penalties). Analogously, lines 37-42 add the coefficients given by Eq. (17) to the appropriate cells for any pair of Bayesian variables ($H_{consist}$ penalties). The resulting complexity for the matrix filling procedure (Algorithm 4) is $\mathcal{O}(n^3)$.

It is also worth mentioning that, in the QUBO matrix Q, the variables are ordered in the following way: first, the binary variables encoding the edges (d_{ij}); then, the binary slack variables (y_{il}) related to the maximum parent constraint; finally, the binary variables (r_{ij}) encoding the topological order. By sorting appropriately the variables in the quadratic terms (they define the row and column indices), the outcome is an upper triangular matrix; otherwise, the content of the non-zero cells below the main diagonal should be transferred to the corresponding cells above the diagonal (summing up the values).

3.2 Complexity

The overall complexity of the QUBO matrix construction (Algorithm 1) is $\mathcal{O}(n^4 + n^3 N r^3_{max})$. Hence, it is determined by several factors: the number of Bayesian variables (n) of the considered BNSL problem, the number of examples (N) in the dataset, and the maximum number

```
Input: zero matrix $Q$, number of Bayesian variables $n$, list of parent sets $parentSets$, parent set
       weights $w$, list of penalty values $\delta_{max}$, penalty value $\delta_{trans}$, penalty value $\delta_{consist}$
Result: QUBO matrix $Q$ filled according to $H_{score}$, $H_{max}$, $H_{trans}$, and $H_{consist}$
for $i \leftarrow 1$ to $n$ do
    /* $H_{score}$-related terms (Eq. (11)) */
    for $\pi \in parentSets[i]$ do
        if $size(\pi) = 1$ then                                         // diagonal elements
            $j \leftarrow \pi[1]$;
            $row \leftarrow col \leftarrow indexOf(d_{ji})$;
            $Q[row][col] \leftarrow Q[row][col] + w_i(\pi)$;
        else if $size(\pi) = 2$ then                                    // out-of-diagonal elements
            $x,\ y \leftarrow \pi[1],\ \pi[2]$;
            $row,\ col \leftarrow indexOf(d_{xi}),\ indexOf(d_{yi})$;
            $Q[row][col] \leftarrow Q[row][col] + w_i(\pi)$;
        end
    end
    /* $H_{max}$-related terms (Eq. (14)) */
    $m \leftarrow 2$;                                                   // max. num. of parents
    $sqBinVars \leftarrow binaryVarsInSquare()$;        // $d_i$ and $y_i$ in (14) are sums of binary vars
    $c \leftarrow binaryVarsCoefficientsInSquare()$;        // the coefficients are either -1 or -2
    for $j \leftarrow 1$ to $size(sqBinVars)$ do
        $row \leftarrow col \leftarrow indexOf(sqBinVars[j])$;                 // diagonal elements indices
        $Q[row][col] \leftarrow Q[row][col] + \delta^{(i)}_{max} \cdot c[j]^2$;                // squared term
        $Q[row][col] \leftarrow Q[row][col] + \delta^{(i)}_{max} \cdot (2 \cdot m \cdot c[j])$;          // double product with $m$
        for $k \leftarrow j+1$ to $size(sqBinVars)$ do                   // out-of-diagonal elements
            $row \leftarrow indexOf(sqBinVars[j])$;
            $col \leftarrow indexOf(sqBinVars[k])$;
            $Q[row][col] \leftarrow Q[row][col] + \delta^{(i)}_{max} \cdot (2 \cdot c[j] \cdot c[k])$;     // double product between vars
        end
    end
    /* $H_{cycle}$-related terms */
    for $j \leftarrow i+1$ to $n$ do
        /* $H_{trans}$-related terms (Eq. (16)) */
        for $k \leftarrow j+1$ to $n$ do
            $row \leftarrow col \leftarrow indexOf(r_{ik})$;
            $Q[row][col] \leftarrow Q[row][col] + \delta_{trans}$;          // $r_{ik}$ coefficient (diagonal element)
            $row,\ col \leftarrow indexOf(r_{ij}),\ indexOf(r_{jk})$;
            $Q[row][col] \leftarrow Q[row][col] + \delta_{trans}$;                        // $r_{ij} \cdot r_{jk}$ coefficient
            $row,\ col \leftarrow indexOf(r_{ij}),\ indexOf(r_{ik})$;
            $Q[row][col] \leftarrow Q[row][col] - \delta_{trans}$;                        // $r_{ij} \cdot r_{ik}$ coefficient
            $row,\ col \leftarrow indexOf(r_{ik}),\ indexOf(r_{jk})$;
            $Q[row][col] \leftarrow Q[row][col] - \delta_{trans}$;                        // $r_{ik} \cdot r_{jk}$ coefficient
        end
        /* $H_{consist}$-related terms (Eq. (17) */
        $row,\ col \leftarrow indexOf(d_{ji}),\ indexOf(r_{ij})$;
        $Q[row][col] \leftarrow Q[row][col] + \delta_{consist}$;                        // $d_{ji} \cdot r_{ij}$ coefficient
        $row \leftarrow col \leftarrow indexOf(d_{ij})$;
        $Q[row][col] \leftarrow Q[row][col] + \delta_{consist}$;          // $d_{ij}$ coefficient (diagonal element)
        $row,\ col \leftarrow indexOf(d_{ij}),\ indexOf(r_{ij})$;
        $Q[row][col] \leftarrow Q[row][col] - \delta_{consist}$;                        // $d_{ij} \cdot r_{ij}$ coefficient
    end
end
return $Q$;
```

Algorithm 4: *fillQ(Q, n, parentSets, w, δ_{max}, δ_{trans}, $\delta_{consist}$)*

of states (r_{max}) among the Bayesian variables. In particular, if the number of Bayesian variables is smaller than the dataset size, the dominant complexity term becomes $n^3 N r_{max}^3$. The situation just depicted is typical. Indeed, n cannot be too big due to the limitations (in the number of qubits and connectivity) of the current quantum annealers, whereas N must be considerably large to provide enough information to learn from. Therefore, typically, the calculation of the local score values s (Algorithm 2) turns out to be the most expensive operation in the QUBO matrix construction; otherwise, it would be the initialization of Q to zero (on a par with the Δ calculation, given the implementation of the operation in question). Concerning the maximum number of states of the Bayesian variables, its contribution is particularly relevant for Bayesian variables with continuous states. In fact, the variables in question must be discretized, and the greater the representation accuracy, the higher the execution time.

3.3 Execution speedup

The construction of the QUBO matrix (including the intermediate values calculation) is what takes most of the execution time. In general, a speedup could be obtained by using a different programming language such as C++; nevertheless, this is not feasible here due to D-Wave's Ocean library, which is necessary to interface with the quantum annealer and is written in Python. Instead, a valid solution consists in performing a dynamic compilation of the code through Numba [16], a just-in-time compiler for Python. In detail, Numba requires to apply a decorator to the functions that must be compiled. Then, during the execution, the first time a function with a decorator is called, it is compiled into machine code, and all the subsequent calls will run the machine code instead of the original Python code. It is important to notice that Numba works better on code including loops, NumPy arrays and library functions. Moreover, the speedup is effective only if a function is called several times; otherwise, in the case of one call in a run, the execution will be slower.

Actually, Numba has been exploited only in the experiments related to the divide et impera approach (Section 5.5), since it has been introduced after the completion of the experiments related to O'Gorman's algorithm (Section 5.4). It is also worth mentioning that the dynamic compilation has been applied only to the two functions called most often in the execution (i.e., $calcNi\pi jk$ and another internal procedure).

4 Divide et Impera Approach

Embedding problems in the QPU topology requires a huge number of qubits due to the limited connectivity of the current quantum annealers. Moreover, the QUBO formulation of the BNSL problem proposed by O'Gorman et al. is densely connected by definition, making infeasible its application even to instances with a not-so-high number of variables. For these reasons, a divide et impera approach has been developed and tested; the pseudocode is shown in Algorithm 5.

The first step is the subproblems formulation (lines 1-3). Let n be the number of variables of the original BNSL problem, r be an array containing the number of states for each variable, and *examples* be a $N \times n$ matrix representing the dataset. The BNSL subproblems are generated as combinations of the n variables taken k at a time, where k represents the desired number of variables for each subproblem. In detail, all possible combinations of variables are generated, and each subproblem is identified by the (*examples* matrix column) indices

```
Input: number of variables of the original BNSL problem $n$, number of variables for each
       subproblem $k$, list of number of states $r$, dataset $examples$
Output: adjacency matrix of the solution to the original problem $sol$
/* Subproblems formulation */
$subproblems \leftarrow combinations(n, k)$;
$subproblemsR \leftarrow filter(r, subproblems)$;
$subproblemsEx \leftarrow filter(examples, subproblems)$;
/* Subproblems solution */
$S \leftarrow Set()$;
for $i \leftarrow 0$ to $size(subproblems)$ do
    $subprob$, $subprobR$, $subprobEx \leftarrow subproblems[i]$, $subproblemsR[i]$, $subproblemsEx[i]$;
    $subprobQ \leftarrow calcQUBOMatrix(size(subprob), subprobR, subprobEx)$;          // Algorithm 1
    $subprobAdjMatrix \leftarrow solveQUBO(subprobQ)$;
    $S.add(subprob, subprobAdjMatrix)$;
end
/* Original solution reconstruction */
$C,\ P \leftarrow countEdgesAndPenalties(S, k)$;
$sol \leftarrow zeroMatrix()$;
for $i \leftarrow 0$ to $n$ do
    for $j \leftarrow 0$ to $n$ do
        if $i \neq j$ then
            if $(C_{ij} - P_{ij}) > 0$ and $C_{ij} > C_{ji}$ then
                $sol[i][j] = 1$;
            end
        end
    end
end
return $sol$;
```

Algorithm 5: *divideEtImpera(n, k, r, examples)*

of the variables included. In practice, the *combinations* function from the Python *itertools* module is used. The complexity of this procedure is $O(c\binom{n}{k})$, where c is the (constant) cost for creating a list of indices. It is also worth mentioning that k should be larger or equal than 3 since that is the minimum reasonable number of variables for the QUBO encoding (H_{trans} assumes $n \geq 3$). Then, for each k-variables combination, it is necessary to filter the r vector and the *examples* matrix, obtaining a vector of k elements and a $N \times k$ matrix, respectively. The total complexity of the filtering operations is $O(\binom{n}{k}(k + Nk))$.

After the subproblems generation, the implementation of O'Gorman's algorithm presented in Section 3 can be applied to each subproblem (line 7), obtaining the respective QUBO matrix, which can be submitted to the annealer or solved with alternative methods (line 8). The outcome is an adjacency matrix for each subproblem. Regarding the complexity of this step, it is linear with respect to the number of subproblems ($\binom{n}{k}$).

Eventually, the solution to the original BNSL problem must be reconstructed starting from the subproblems solutions (lines 11-21). Let S be the set of all subproblems solutions, where each solution consists of the list of indices of the variables included and an adjacency matrix for the corresponding graph. The relevant information here is the presence of edges, thus the first step consists in counting how many times each edge appears in the subproblems solutions. Indeed, each pair of variables is present in more than one subproblem. Let C be the set of counts, with C_{ij} representing the number of appearances of the (i, j) edge (the edges are directed, hence (i, j) and (j, i) are different). After the counting phase, whose complexity is $O(\binom{n}{k}k^2)$, the reconstruction of the solution can start. Actually, two strategies have been developed for this. The first one (the simplest one) consists in inserting in the adjacency matrix of the original problem every edge (i, j) that appears at least one time in one subproblem, i.e., for which $C_{ij} > 0$. If both C_{ij} and C_{ji} are larger than 0, then the edge with the highest number of counts is picked; in this way, cycles of two nodes are avoided (the resulting graph must be a DAG). Instead, the second strategy (the one shown in the pseudocode) requires additional information to perform the reconstruction, namely, the penalty values P_{ij}. Basically, P_{ij} represents the number of subproblems including the variables i and j in which the edge (i, j) is not present; hence, the computation complexity is the same as for C. In practice, an edge (i, j) is added to the final solution if (i) the difference between C_{ij} and P_{ij} is larger than 0 and (ii) C_{ij} is larger than C_{ji}, otherwise it is discarded. Note that, if these conditions are satisfied, so are those of the first method ($C_{ij} > 0$ and $C_{ij} > C_{ji}$). In the experiments presented in the next section, only the second strategy has been exploited, since some preliminary experiments have confirmed its superiority. Concerning the resulting complexity of the reconstruction phase, it is $O(\binom{n}{k}k^2 + n^2)$.

5 Empirical Evaluation

This section deals with the Bayesian problems selected, the datasets generation procedure employed, the methods tested, the experimental setup, and the results obtained for both O'Gorman's algorithm and the divide et impera approach.

5.1 Bayesian problems

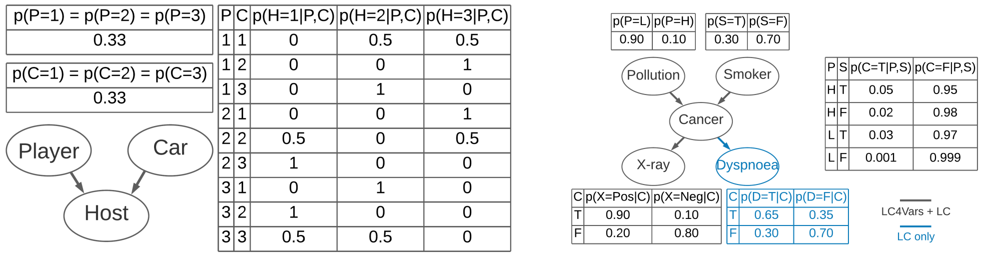
Three out of the four Bayesian problems used have been selected from the examples provided by the Bayes Server site [17], whereas the last one (the Lung Cancer) has been taken from a different source [18]. In detail, the implementation of O'Gorman's algorithm has been tested on the Monty Hall, the Lung Cancer and the Waste problem. Instead, the divide et impera approach has been tested on the Lung Cancer, the Waste, and the Alarm problem. It is also worth highlighting that most of these problems have been subjected to some modifications (explained in the following paragraphs) before applying the presented methods.

Monty Hall Problem

The Monty Hall Problem (MHP) has been chosen because of its simplicity. In detail, the Bayesian network of the problem (see Figure 1, on the left) is composed of three variables ($n = 3$), and each variable has three possible states ($\{1, 2, 3\}$). Actually, three is also the minimum reasonable number of variables for O'Gorman's QUBO formulation. Indeed, the transitivity Hamiltonian is based on the assumption that at least three Bayesian variables are present.

Lung Cancer

The Lung Cancer problem has been selected due to its (not excessively) higher number of variables with respect to the MHP. In particular, the original network (LC) consists of $n = 5$ variables and each Bayesian variable admits two possible states. Actually, also a variant (LC4Vars) with $n = 4$ variables obtained by removing the "Dyspnoea" node has been tested



p(P=1) = p(P=2) = p(P=3)
0.33

p(C=1) = p(C=2) = p(C=3)
0.33

P	C	p(H=1|P,C)	p(H=2|P,C)	p(H=3|P,C)
1	1	0	0.5	0.5
1	2	0	0	1
1	3	0	1	0
2	1	0	0	1
2	2	0.5	0	0.5
2	3	1	0	0
3	1	0	1	0
3	2	1	0	0
3	3	0.5	0.5	0

p(P=L)	p(P=H)
0.90	0.10

p(S=T)	p(S=F)
0.30	0.70

P	S	p(C=T|P,S)	p(C=F|P,S)
H	T	0.05	0.95
H	F	0.02	0.98
L	T	0.03	0.97
L	F	0.001	0.999

C	p(X=Pos|C)	p(X=Neg|C)
T	0.90	0.10
F	0.20	0.80

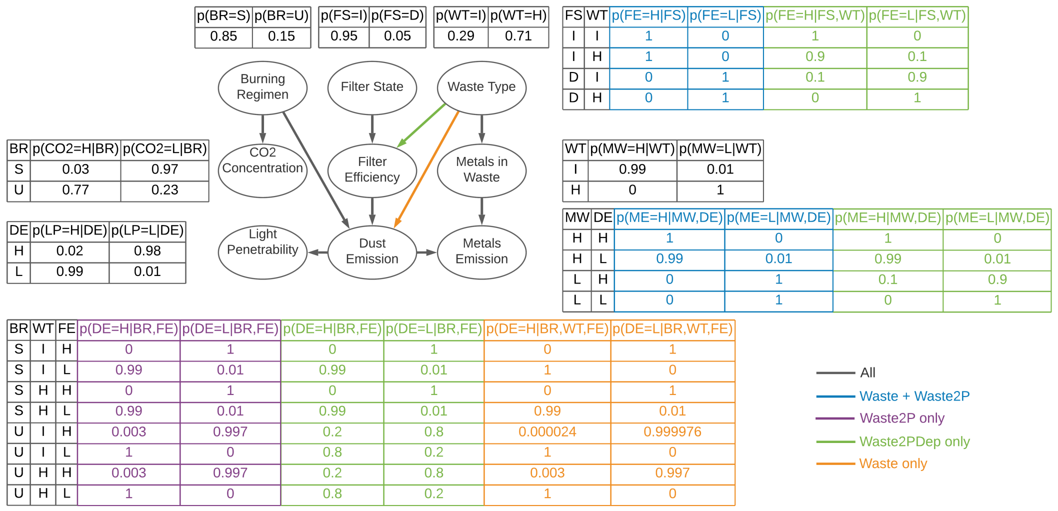
C	p(D=T|C)	p(D=F|C)
T	0.65	0.35
F	0.30	0.70

Fig. 1. Monty Hall Problem (left) and Lung Cancer (right).

here. In this way, a more accurate analysis on the problem size scaling could be performed. Both networks are shown in Figure 1 (right).

Waste

The Waste problem has been chosen for different reasons: it has a considerably larger size than the previous two; it includes continuous Bayesian variables, and also a variable with more than two parents. In detail, the Bayesian network of the original problem is composed of nine variables ($n = 9$), of which three are discrete with two states, and six are continuous. Since the QUBO encoding admits only discrete variables, a discretization has been required for the continuous ones. However, it is not possible to use a single discretization threshold because the variables have different mean values. Hence, each continuous variable has been transformed into a discrete one with two states by applying the following procedure: first, the lowest and the highest values are identified by evaluating all the settings of the parent variables (this is possible since the number of variables and states is small), and the average of the two values is kept as a discretization threshold; then, for each combination of parent states, the mean and



p(BR=S)	p(BR=U)
0.85	0.15

p(FS=I)	p(FS=D)
0.95	0.05

p(WT=I)	p(WT=H)
0.29	0.71

FS	WT	p(FE=H|FS)	p(FE=L|FS)	p(FE=H|FS,WT)	p(FE=L|FS,WT)
I	I	1	0	1	0
I	H	1	0	0.9	0.1
D	I	0	1	0.1	0.9
D	H	0	1	0	1

BR	p(CO2=H|BR)	p(CO2=L|BR)
S	0.03	0.97
U	0.77	0.23

WT	p(MW=H|WT)	p(MW=L|WT)
I	0.99	0.01
H	0	1

DE	p(LP=H|DE)	p(LP=L|DE)
H	0.02	0.98
L	0.99	0.01

MW	DE	p(ME=H|MW,DE)	p(ME=L|MW,DE)	p(ME=H|MW,DE)	p(ME=L|MW,DE)
H	H	1	0	1	0
H	L	0.99	0.01	0.99	0.01
L	H	0	1	0.1	0.9
L	L	0	1	0	1

BR	WT	FE	p(DE=H|BR,FE)	p(DE=L|BR,FE)	p(DE=H|BR,FE)	p(DE=L|BR,FE)	p(DE=H|BR,WT,FE)	p(DE=L|BR,WT,FE)
S	I	H	0	1	0	1	0	1
S	I	L	0.99	0.01	0.99	0.01	1	0
S	H	H	0	1	0	1	0	1
S	H	L	0.99	0.01	0.99	0.01	0.99	0.01
U	I	H	0.003	0.997	0.2	0.8	0.000024	0.999976
U	I	L	1	0	0.8	0.2	1	0
U	H	H	0.003	0.997	0.2	0.8	0.003	0.997
U	H	L	1	0	0.8	0.2	1	0

Fig. 2. Waste.

the variance of the continuous variable are taken down, and the probability of the Gaussian with these parameters having a value higher than the threshold is set as the probability of the high state (H, opposed to the low state L). Specifically, to determine the lowest (highest) value, the respective variance is subtracted from (summed to) the minimum (maximum) mean value observed; moreover, in the case of a continuous variable with a continuous parent, the parent is discretized first, and the lowest and highest values are used as evidence for the parent states L and H. The resulting Bayesian problem is denoted here as Waste.

Figure 2 summarizes all the considered variants of the problem. In particular, Waste differs from the original problem only in the lack of the edge between *Waste Type* and *Filter Efficiency*, which has been lost in the discretization procedure. Instead, in Waste2P (variant of Waste), the edge between *Waste Type* and *Dust Emission* has been removed so that the maximum number of parents is equal to two (in practice, the most balanced probability values have been kept for the *Dust Emission* node). Finally, Waste2PDep is a variant of Waste2P in which the edge between *Waste Type* and *Filter Efficiency* has been reintroduced by manually altering some probability values (in the *Filter Efficiency* probability table). In addition, in Waste2PDep, some other probabilities have been slightly changed (*Dust Emission* and *Metals Emission* tables) in order to have more balanced probability distributions.

Alarm

Eventually, Alarm has been picked mainly for its size. Indeed, it is quite close to the maximum BNSL problem size that can be embedded in the Pegasus topology using O'Gorman's formulation (≈ 18). Moreover, the presence of a variable with four parents allows evaluating the ability of the divide et impera approach to reconstruct Bayesian networks with more than two parents for a single variable. Actually, the original Alarm problem consists of 38 Bayesian variables, whereas the version used here includes only 15 of them (with their structure preserved). The Bayesian network employed in the experiments is shown in Figure 3.

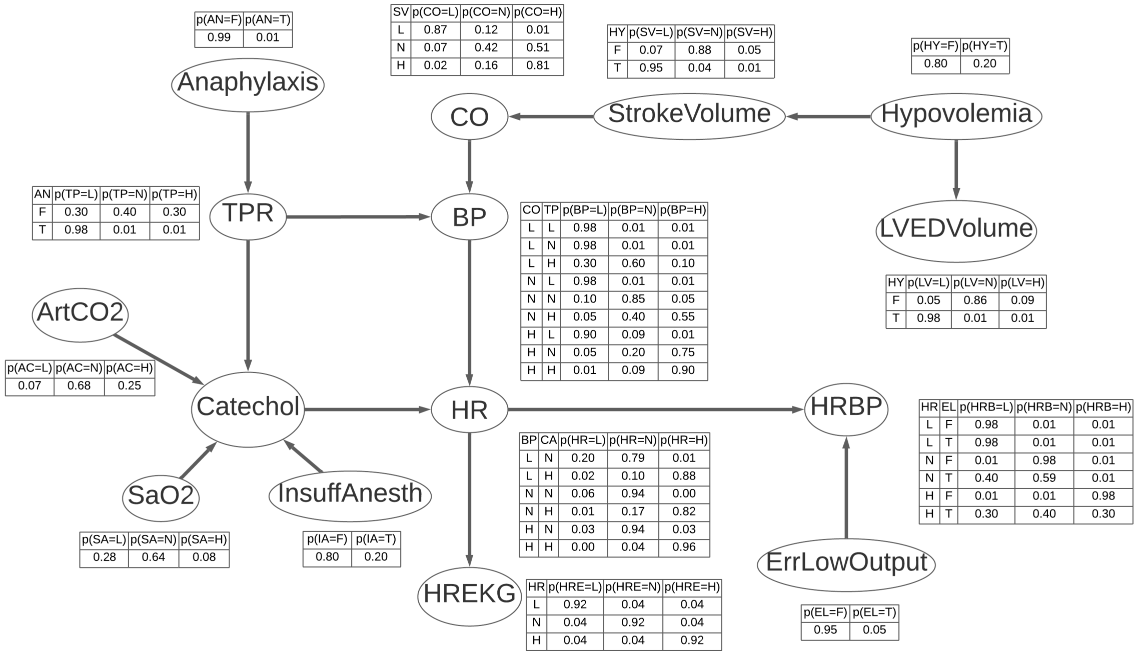
5.2 Datasets generation

For each problem, several datasets have been generated varying both the size and the creation method. Specifically, three dataset sizes (N) have been used, namely, 10^4 (10K), 10^5 (100K), and 10^6 (1M). Regarding the generation methods, two have been employed. The first one consists in generating N examples through the *uniform* function from the Python *random* module by sampling from the network probability distribution. Instead, the second method aims at generating datasets with zero variance, i.e., with combinations of states appearing exactly the expected amount of times. In particular, in order to generate the expected datasets, the probability p of every combination of states of the network variables is calculated; then, for each combination, $\lfloor N * p \rfloor$ examples are inserted in the dataset. Actually, for some probability values, it may not be possible to generate an integer number of examples, and, consequently, the variance of the dataset is not exactly zero. In addition, the resulting dataset may have a number of samples lower than the desired one. For instance, for the Alarm problem, the dataset of size $N = 10^4$ generated using this method has a considerably lower number of samples (≈ 9000) due to the presence of many state combinations that are not represented at all because of their very low probability values (the other problems datasets are not significantly affected by this issue). The datasets generated using the second method are denoted as *Exp*.

Catechol

IA	TP	AC	SA	p(CA=N)	p(CA=H)
F	L	L	L	0.05	0.95
F	L	L	N	0.10	0.90
F	L	L	H	0.95	0.05
F	L	N	L	0.05	0.95
F	L	N	N	0.10	0.90
F	L	N	H	0.95	0.05
F	L	H	L	0.01	0.99
F	L	H	N	0.10	0.90
F	L	H	H	0.30	0.70
F	N	L	L	0.05	0.95
F	N	L	N	0.95	0.05
F	N	L	H	0.99	0.01
F	N	N	L	0.05	0.95
F	N	N	N	0.95	0.05
F	N	N	H	0.99	0.01
F	N	H	L	0.01	0.99
F	N	H	N	0.30	0.70
F	N	H	H	0.99	0.01
F	H	L	L	0.05	0.95
F	H	L	N	0.95	0.05
F	H	L	H	0.95	0.05
F	H	N	L	0.05	0.95
F	H	N	N	0.95	0.05
F	H	N	H	0.99	0.01
F	H	H	L	0.01	0.99
F	H	H	N	0.30	0.70
F	H	H	H	0.30	0.70

IA	TP	AC	SA	p(CA=N)	p(CA=H)
T	L	L	L	0.01	0.99
T	L	L	N	0.01	0.99
T	L	L	H	0.01	0.99
T	L	N	L	0.01	0.99
T	L	N	N	0.01	0.99
T	L	N	H	0.01	0.99
T	L	H	L	0.01	0.99
T	L	H	N	0.01	0.99
T	L	H	H	0.01	0.99
T	N	L	L	0.01	0.99
T	N	L	N	0.01	0.99
T	N	L	H	0.05	0.95
T	N	N	L	0.01	0.99
T	N	N	N	0.01	0.99
T	N	N	H	0.05	0.95
T	N	H	L	0.01	0.99
T	N	H	N	0.01	0.99
T	N	H	H	0.01	0.99
T	H	L	L	0.01	0.99
T	H	L	N	0.05	0.95
T	H	L	H	0.05	0.95
T	H	N	L	0.01	0.99
T	H	N	N	0.05	0.95
T	H	N	H	0.05	0.95
T	H	H	L	0.01	0.99
T	H	H	N	0.01	0.99
T	H	H	H	0.01	0.99



p(AN=F)	p(AN=T)
0.99	0.01

SV	p(CO=L)	p(CO=N)	p(CO=H)
L	0.87	0.12	0.01
N	0.07	0.42	0.51
H	0.02	0.16	0.81

HY	p(SV=L)	p(SV=N)	p(SV=H)
F	0.07	0.88	0.05
T	0.95	0.04	0.01

p(HY=F)	p(HY=T)
0.80	0.20

AN	p(TP=L)	p(TP=N)	p(TP=H)
F	0.30	0.40	0.30
T	0.98	0.01	0.01

CO	TP	p(BP=L)	p(BP=N)	p(BP=H)
L	L	0.98	0.01	0.01
L	N	0.98	0.01	0.01
L	H	0.30	0.60	0.10
N	L	0.98	0.01	0.01
N	N	0.10	0.85	0.05
N	H	0.05	0.40	0.55
H	L	0.90	0.09	0.01
H	N	0.05	0.20	0.75
H	H	0.01	0.09	0.90

HY	p(LV=L)	p(LV=N)	p(LV=H)
F	0.05	0.86	0.09
T	0.98	0.01	0.01

p(AC=L)	p(AC=N)	p(AC=H)
0.07	0.68	0.25

HR	EL	p(HRB=L)	p(HRB=N)	p(HRB=H)
L	F	0.98	0.01	0.01
L	T	0.98	0.01	0.01
N	F	0.01	0.98	0.01
N	T	0.40	0.59	0.01
H	F	0.01	0.01	0.98
H	T	0.30	0.40	0.30

BP	CA	p(HR=L)	p(HR=N)	p(HR=H)
L	N	0.20	0.79	0.01
L	H	0.02	0.10	0.88
N	N	0.06	0.94	0.00
N	H	0.01	0.17	0.82
H	N	0.03	0.94	0.03
H	H	0.00	0.04	0.96

p(SA=L)	p(SA=N)	p(SA=H)
0.28	0.64	0.08

p(IA=F)	p(IA=T)
0.80	0.20

HR	p(HRE=L)	p(HRE=N)	p(HRE=H)
L	0.92	0.04	0.04
N	0.04	0.92	0.04
H	0.04	0.04	0.92

p(EL=F)	p(EL=T)
0.95	0.05

Fig. 3. Alarm.

5.3 Methods and experimental setup

After the application of O'Gorman's algorithm, the BNSL problem encoded in the QUBO matrix must be solved. For this purpose, three methods have been exploited in the experiments: quantum annealing (QA), simulated annealing (SA) and exhaustive search (ES). The functioning of quantum annealing has been already explained in Section 2.2. Instead, simulated annealing is a well-known classical metaheuristic technique used for solving optimization problems [19]; further details about the algorithm can be found here [b]. Eventually, the exhaustive search represents an optimized brute force approach.

In detail, a simple brute force on the QUBO representation would be unfeasible even for small BNSL instances due to the high number of binary variables. Hence, ES limits the brute force to the edge binary variables d_{ij}, while considering only the best setup of y_{il} and r_{ij}. In particular, for each Bayesian variable i, the y_{il} binary variables must be set so that y_i (in Eq. (14)) is equal to the difference between m and d_i. In this way, there is no penalty from the max Hamiltonian. Obviously, this is possible only for nodes that do not have more than m parents; otherwise, the minimum penalty is given by y_i equal to zero. Instead, setting the r_{ij} binary variables is more complex. Indeed, computing the topological order of the graph encoded in d_{ij} is not enough since, if two Bayesian variables i and j are not connected (i.e., it does not exist a path from one variable to the other), there is no straightforward setup for r_{ij}. In addition, setting all the uncertain binary variables to either 0 or 1 does not solve the problem since a cycle might be produced, with consequent penalty from H_{trans}. The solution consists in completing the graph encoded in d_{ij} by adding one edge at a time (while verifying that no cycle is formed), and then computing the topological order of the resulting graph. In this way, it is possible to properly set all r_{ij} binary variables and avoid any penalty from the cycle Hamiltonian. Obviously, if the graph encoded in d_{ij} contains a cycle, it is not possible to avoid the penalty. The resulting complexity for setting y_{il} and r_{ij} turns out to be $\mathcal{O}(n^3)$ for sparse graphs and $\mathcal{O}(n^4)$ for dense graphs. In the end, these optimizations do not change the complexity class of ES with respect to the simple brute force (it remains exponential in the number of Bayesian variables n) but significantly reduce the number of operations to be performed. Another improvement that has been introduced in ES is the parallelization of the solutions evaluation.

Regarding the setup for the experiments on the implementation of O'Gorman's algorithm, different combinations of annealing parameters (number of annealer reads and annealing time) [20] have been tested for QA; the specific values are reported in the various results sections. In addition, the default annealing schedule has been employed (the system used is Advantage 1.1), and the QPU embedding has been performed through the *EmbeddingComposite* class (see Section 2.3). As concerns SA, the implementation provided by D-Wave [21] has been exploited. Except for the number of reads (different values have been employed), the default configuration has been used, and the execution has been carried out on a local machine. Eventually, ES does not require to set parameters and has also been executed locally. In detail, a machine with a quad-core CPU (Intel i5-6400) and 16 GB of RAM has been used for the datasets generation, the QUBO matrices construction, and the execution of the classical methods.

Concerning the divide et impera approach, only one configuration of annealing parameters

[b] `https://en.wikipedia.org/wiki/Simulated_annealing`

has been evaluated for QA; the reason and the values are specified in the related results section. Furthermore, the default annealing schedule has been employed, but, in this case, the system used is Advantage 4.1 since the previous model had already been dismissed. Instead, the QPU embedding method and the SA implementation are the same as those exploited in the experiments on O'Gorman's algorithm; the only difference lies in the number of reads used for SA, which has been set to a single value too. Finally, ES has not been evaluated on this approach due to its high time requirements. All the classical operations for the divide et impera approach have been executed on a machine with a quad-core CPU (Intel i7-7700HQ) and 16 GB of RAM.

Eventually, it is worth mentioning that, for both the implementation of O'Gorman's algorithm and the divide et impera approach, the performance-related analyses presented here involve only *Exp* datasets because some preliminary tests have confirmed that the difference in terms of performance when working on *Exp* or non-*Exp* datasets is negligible.

5.4 O'Gorman's algorithm results

Several experiments have been performed on the implementation of O'Gorman's algorithm. The results obtained are presented in the subsequent paragraphs.

QUBO formulation correctness and α_{ijk} hyperparameters

The QUBO encoding must accurately represent the original BNSL problem; to this end, the α_{ijk} hyperparameters must be set appropriately. In detail, if α_{ijk} have suitable values, the image (x^TQx) of the expected solution will be the global minimum. To verify this, it is necessary to find the global minimum solution through ES and compare it with the expected one. In particular, the QUBO version of the expected solution (x) is obtained as follows: the Bayesian network that has been used to generate the dataset is exploited to set the d_{ij} binary variables encoding the edges, whereas the best setup of y_{il} and r_{ij} is found using the same approach employed by ES (see Section 5.3).

Since the objective is to learn the structure of a Bayesian network from a set of data, the values of α_{ijk} must be uninformative, i.e., they must not encode information about the target Bayesian network. For this reason, the first values of α_{ijk} that have been tested are $N/(r_i \cdot q_i)$ and 1, as proposed by Heckerman et al. [22]. The results obtained are reported in Table 1. In detail, the first alternative ($N/(r_i \cdot q_i)$) has performed decently on the MHP problem (the smallest one), and extremely bad on all the others (characterised by a higher number of variables); in practice, N prior counts uniformly distributed among all "variable" - "parent set" state combinations are added in Eq. (13). Instead, the second possibility (i.e., 1) has not worked at all. Therefore, other values have been evaluated, namely, $1/(r_i \cdot q_i)$ and $1/r_i$, which have been selected with the idea of influencing the counts as little as possible. As reported in the same table, both of them have shown the desired behaviour, and the first one has been chosen as the default setup.

In particular, the ratios in Table 1 have been obtained using 8 datasets for each "problem" - "α_{ijk} value" combination; the datasets in question have been created with different sizes and exploiting both generation methods (see Section 5.2). Specifically, four datasets of size $N = 10^4$ (of which one *Exp*), two datasets of size $N = 10^5$ (of which one *Exp*), and two datasets of size $N = 10^6$ (of which one *Exp*) have been used.

Table 1. Ratio of the number of times in which the best and the expected solutions coincide to the number of tests (8 for each cell), for different problems and α_{ijk} values.

Problem	$\alpha_{ijk} = N/(r_i \cdot q_i)$	$\alpha_{ijk} = 1$	$\alpha_{ijk} = 1/(r_i \cdot q_i)$	$\alpha_{ijk} = 1/r_i$
MHP	0.75	0.00	**1.00**	1.00
LC4Vars	0.00	0.00	**1.00**	1.00
LC	0.00	0.00	**1.00**	1.00

Dataset size and QUBO matrix construction time

After the choice of the α_{ijk} value, the impact of the dataset size on the QUBO matrix construction time has been analysed. As shown in Table 2, the required time increases linearly with the dataset size in accordance with the complexity $\mathcal{O}(n^4 + n^3 N r_{max}^3)$ discussed in Section 3.2. In practice, large dataset sizes turn out to be prohibitive, especially when constructing the QUBO matrix for problems with a high number of variables. Because of this and the fact that a dataset size larger than $N = 10^4$ leads to no improvement in terms of performance (see Table 3), it is advantageous to keep the dataset size limited.

Regarding the tables data, the time values in Table 2 have been obtained using one *Exp* and four non-*Exp* datasets for each "problem" - "dataset size" combination (hence, five runs for each entry). Instead, the values in Table 3 have been acquired through QA, using *Exp* datasets only, 10^4 reads, 20μs of annealing time, and 10 runs for each dataset size. The metric reported in the second table is described in the performance section later on; for the time being, it is sufficient to know that larger values in the table correspond to better performance.

Table 2. Average QUBO matrix (Q) construction time in seconds, for different problems and dataset sizes. For each entry, 5 different datasets (of which one *Exp*) have been used.

Problem	$N = 10^4$	$N = 10^5$	$N = 10^6$
MHP	0.55	4.03	40.03
LC4Vars	0.64	5.86	58.78
LC	1.40	13.55	136.21
Waste	9.79	98.85	1010.30
Waste2P	9.74	98.78	1001.96
Waste2PDep	9.77	100.26	1007.61

Table 3. Average solution value found by QA in Waste2PDep on *Exp* datasets of different sizes. 10^4 reads, 20μs of annealing time, and 10 runs (for each dataset size) have been used.

Problem	$N = 10^4$	$N = 10^5$	$N = 10^6$
Waste2PDep	0.963	0.960	0.965

Number of reads and annealing time (QA)

The number of reads, i.e., measurements, and the annealing time per read are two extremely relevant parameters for the performance of QA. Hence, an extensive experimental evaluation has been performed, with 10 runs for each configuration; the results are reported in Table 4. In detail, the maximum allowed number of reads (on D-Wave systems) is 10^4, whereas the maximum annealing time per read is 2000μs. However, there is also an internal constraint that prevents an annealing time larger than 999μs with 10^3 reads, and an annealing time larger than 99μs with 10^4 reads. Looking at the results, it is clear that a higher number of reads provides better performance, and the same holds for the annealing time. Nevertheless, the number of reads has a more significant impact. Indeed, the results achieved with the maximum allowed number of reads and an annealing time far lower than the joint limit are clearly better than those achieved with the annealing time maximized. Hence, the best setup corresponds to the maximum allowed number of reads (10^4) and the joint limit annealing time (99μs).

Table 4. Ratio of the number of times the global minimum is found by QA to the number of experiments (10 for each configuration), for different numbers of reads and annealing times on the LC *Exp* dataset with size $N = 10^4$.

	Annealing time (μs)					
reads #	1	10	20	50	99	999
10^3	—	0.00	—	—	—	0.10
10^4	0.00	0.20	0.20	0.40	0.50	—

Performance

Finally, all the methods described in Section 5.3 have been applied to all the problems presented in Section 5.1 with the purpose of comparing their performance; the results are reported in Table 5. In particular, for these experiments, *Exp* datasets of size $N = 10^4$ have been employed. Moreover, 10^4 reads have been used for SA, whereas, for QA, the best setup has been exploited (i.e., 10^4 reads and an annealing time per read equal to 99μs). The number of runs, for both SA and QA, is 10, and the success rate is given by the ratio between the number of runs in which the expected solution has been discovered and the total number of runs. Instead, the result value represents the ratio between the QUBO image (x^TQx) of the solution found and the QUBO image of the expected solution, averaged over the runs (larger values correspond to better performance, since the image value of the expected solution is always negative in these experiments). Regarding the time values, they include only the resolution of the QUBO matrix; specifically, in the case of QA, they correspond to the QPU access time (see [23] for additional details). Eventually, for QA, the last column (*Average # exp. sol.*) reports the number of times that the expected solution has been found in a single run, averaged over the runs.

In practice, ES has outperformed both SA and QA on the smallest problems, i.e., MHP and LC4Vars, always finding the minimum in less time. However, due to its exponential complexity, it has lost the comparison on the LC problem ($n = 5$), and has turned out to be too time-consuming on the largest ones.

Concerning the other methods, QA has always managed to find the optimum solution to the problems with three and four Bayesian variables, also outperforming SA in terms of execution time. Moreover, it has detected the global minimum several times in each run

Table 5. Comparison of ES, SA, and QA performances on different problems, using *Exp* datasets of size $N = 10^4$, 10^4 reads for SA, and the best setup for QA (10^4 reads, 99 μs annealing time). The number of runs, for both SA and QA, is 10.

		Exhaustive search		Simulated annealing			Quantum annealing			
n	Problem	Success rate	Avg. resol. time (s)	Success rate	Average result	Avg. sol. time (s)	Success rate	Average result	Avg. resol. time (s)	Average # exp. sol.
3	MHP	1.00	0.04	1.00	1.0000	3.40	1.00	1.0000	1.76	215.90
4	LC4Vars	1.00	0.42	1.00	1.0000	5.52	1.00	1.0000	1.77	11.40
5	LC	1.00	137.53	1.00	1.0000	8.93	0.50	0.9987	1.80	0.60
9	Waste	—	—	0.00	1.0145	12.85	0.00	0.9898	2.09	0.00
9	Waste2P	—	—	0.00	0.9999	13.15	0.00	0.9754	2.17	0.00
9	Waste2PDep	—	—	0.00	0.9998	12.39	0.00	0.9780	2.10	0.00

(10^4 measurements per run are performed), which suggests that a fewer number of reads could be enough to achieve the same results on these problems. Instead, for the five-variable problem, QA has managed to discover the minimum only half of the times (with the minimum occasionally appearing more than once), whereas SA has always found it at the cost of a slightly higher runtime. Eventually, neither QA nor SA have ever detected the optimum solution to the largest problems ($n = 9$). Nevertheless, the quality of the solutions found is good on average, especially of those found by SA, whose execution time has turned out to be slightly higher also in this case. It is also worth highlighting that, for the Waste problem, solutions with a better score than that of the expected solution have been found; this has always happened with SA (the average result value is larger than 1.0) and sometimes with QA. The reason lies in the presence of a node with three parents in the expected solution. Basically, this penalises the expected solution and makes possible to have other solutions respecting the maximum parent constraint ($m \leq 2$) with a better score.

In addition, the impact of the annealing time on the performance of QA in the same experiments has been analysed; the results for an annealing time of 1μs and 99μs are reported in Table 6. In practice, a higher annealing time has led to no improvement on the smallest problem (MHP) but has provided better results on average, with only a little additional time required, on all the others. In particular, for the problems with four (LC4Vars) and five (LC) Bayesian variables, it has also provided a higher success rate and a higher number of occurrences of the minimum solution.

Table 6. Comparison of quantum annealing performances on several problems for different values of annealing time, using *Exp* datasets of size $N = 10^4$ and 10^4 reads. The number of runs is 10.

		Annealing time per sample $1\mu s$				Annealing time per sample $99\mu s$			
n	Problem	Success rate	Average result	Avg. resol. time (s)	Average # exp. sol.	Success rate	Average result	Avg. resol. time (s)	Average # exp. sol.
3	MHP	1.00	1.0000	0.78	304.70	1.00	1.0000	1.76	215.90
4	LC4Vars	0.90	0.9997	0.81	5.20	1.00	1.0000	1.77	11.40
5	LC	0.40	0.9980	0.87	0.50	0.50	0.9987	1.80	0.60
9	Waste	0.00	0.9619	1.15	0.00	0.00	0.9898	2.09	0.00
9	Waste2P	0.00	0.9473	1.15	0.00	0.00	0.9754	2.17	0.00
9	Waste2PDep	0.00	0.9633	1.33	0.00	0.00	0.9780	2.10	0.00

Eventually, since SA does not have limits on the number of reads, further experiments have been executed on the Waste2PDep problem with the *Exp* dataset of size $N = 10^4$, using a number of reads equal to 10^5 and 10^6, respectively. The execution time has increased linearly, but no substantial improvement in the performance has been observed.

5.5 Divide et impera results

As mentioned in Section 5.1, the divide et impera approach has been tested on two problems used for the evaluation of O'Gorman's algorithm, namely, LC and Waste (only in their main variant), and a new additional problem, i.e., Alarm. The results are presented in the following paragraphs.

Execution speedup and timing

First of all, the speedup achieved exploiting the technique illustrated in Section 3.3 has been analysed. In particular, for each problem taken into account, one *Exp* dataset of size $N = 10^4$ and one run have been used. Moreover, regarding the number of variables for each subproblem

Table 7. Speedup achieved for different k values on LC and Waste, using an *Exp* datasets of size $N = 10^4$ and a single run. The time values, expressed in seconds, include the subproblems formulation and the QUBO matrices construction. In particular, D.e.I. = divide et impera, O'G. = O'Gorman.

Problem	k	Subproblems #	Time (no speedup)	Time (with speedup)	Speedup
LC ($n = 5$)	3 (D.e.I.)	10	27.66	4.54	6.09x
	4 (D.e.I.)	5	54.4	4.55	11.96x
	5 (O'G.)	1	21.41	3.84	5.58x
Waste ($n = 9$)	3 (D.e.I.)	84	237.5	10.27	23.13x
	4 (D.e.I.)	126	1754.35	32.62	53.78x
	5 (D.e.I.)	126	2704.54	66.02	40.97x
	6 (D.e.I.)	84	2907.48	78.8	36.90x
	7 (D.e.I.)	36	2186.56	54.91	39.82x
	8 (D.e.I.)	9	858.57	23.97	35.82x
	9 (O'G.)	1	149.58	8.8	17x

(k), all values between three (the minimum reasonable value, see Section 4) and n have been tested; it is also worth highlighting that $k = n$ corresponds to the direct application of the implementation of O'Gorman's algorithm. The results are reported in Table 7, with the time values including the subproblems formulation and the QUBO matrices construction (thus, neither the subproblems resolution nor the final solution reconstruction). In detail, only LC and Waste have been considered here, since the times without speedup for Alarm would have been unfeasible to collect using the machine available. Concerning LC, the time required has been reduced by ≈ 9 times on average for the divide et impera approach and ≈ 5.6 times for O'Gorman's algorithm. Instead, for the Waste problem, the speedup has been more significant due to the higher number of variables and/or subproblems, with an average of ≈ 38.4 times for the divide et impera approach and ≈ 17 times for O'Gorman's algorithm.

Instead, Table 8 reports some statistics computed on analogous time values (including subproblems formulation and QUBO matrices construction), which have been obtained using four different non-*Exp* datasets of size $N = 10^4$. The *Exp* datasets have not been included

Table 8. Statistics on subproblems formulation and QUBO matrices construction time for different k values. Specifically, 4 non-*Exp* datasets of size $N = 10^4$ have been used for each problem (one run for each dataset). In addition, the time values, expressed in seconds, refer to the optimized code (i.e., with speedup).

Problem	k	Subproblems #	Average time	STD time	CV time	Average time per subproblem
LC ($n = 5$)	3	10	1.32	0.02	0.014	0.132
	4	5	1.28	0.03	0.022	0.256
Waste ($n = 9$)	3	84	3.72	0.15	0.040	0.044
	4	126	8.89	0.10	0.011	0.071
	5	126	16.54	0.28	0.017	0.131
	6	84	18.80	0.22	0.012	0.224
	7	36	13.55	0.16	0.012	0.376
	8	9	5.92	0.13	0.022	0.658
Alarm ($n = 15$)	3	455	26.68	0.20	0.007	0.059
	4	1365	203.15	0.62	0.003	0.149
	5	3003	1063.36	4.00	0.004	0.354
	6	5005	2952.22	7.08	0.002	0.590
	7	6435	6548.12	235.37	0.036	1.018
	8	6435	10361.01	489.65	0.047	1.610
	9	5005	11370.70	112.19	0.010	2.272

since they may have a lower number of samples, as explained in Section 5.2. In this case, also the Alarm problem has been considered; indeed, all the time values refer to the optimized code (i.e., with speedup). However, due to the still high times, the maximum k value used for it is 9. Eventually, it is worth highlighting that the limit case corresponding to the direct execution of O'Gorman's algorithm ($k = n$) has not been taken into account here. In detail, the highest average time across k values is determined by both the subproblems size and the number of subproblems. Indeed, the average time per subproblem grows with the subproblems size (see the last column). Moreover, looking at the standard deviation (STD) and the coefficient of variation (CV), it turns out that the variance in the input data does not significantly affect the time values. Specifically, the CV value is always lower than 0.05.

Performance

To evaluate the performance of the divide et impera approach, the following setup has been used for each problem: one *Exp* dataset of size $N = 10^4$, five runs, and a number of variables for each subproblem (k) ranging from three (the minimum reasonable value) to n (the total number of Bayesian variables), with the upper limit representing the direct application of O'Gorman's algorithm. Regarding the methods for solving the QUBO encoding, only SA and QA have been exploited in these experiments. Indeed, ES would have required an unreasonable amount of time to solve the subproblems generated with a high k value. Furthermore, 100 reads and an annealing time equal to $1\mu s$ have been used for QA due to the limited quantum resources available and the high number of subproblems to resolve (considering all experiments). To make a fair comparison, 100 reads have been used also for SA. Eventually, it is worth highlighting that only the second reconstruction strategy developed for the divide et impera approach has been applied in these experiments, as explained in Section 4.

Starting from LC, the results achieved for it are reported in Table 9. In addition, a "ROC curve"-like plot is provided in Figure 4. In practice, SA has turned out to perform better than QA on LC, and the divide et impera approach has outperformed the direct application of O'Gorman's algorithm for both resolution methods. Indeed, the higher the sensitivity and

Table 9. Results achieved by the divide et impera approach on the LC problem, for different numbers of variables per subproblem (k) and methods (SA/QA), using an *Exp* dataset of size $N = 10^4$, five runs, 100 reads for SA, and 100 reads and $1\mu s$ of annealing time for QA. The last k value (5) corresponds to the direct application of the implementation of O'Gorman's algorithm. In particular, D.e.I. = divide et impera, O'G. = O'Gorman.

LC ($n = 5$, edges = 4)											
k	Method	Metric	# for each run					# unique	Average #	Sensitivity	Specificity
3 (D.e.I.)	SA	Correct edges	2	2	4	4	3	4	3	0.75	0.94
		Wrong edges	2	2	0	0	1	2	1		
	QA	Correct edges	1	4	4	1	2	4	2.4	0.60	0.90
		Wrong edges	3	0	0	3	2	4	1.6		
4 (D.e.I.)	SA	Correct edges	4	4	4	4	4	4	4	1.00	1.00
		Wrong edges	0	0	0	0	0	0	0		
	QA	Correct edges	3	3	3	3	2	4	2.8	0.70	0.90
		Wrong edges	2	2	1	1	2	5	1.6		
5 (O'G.)	SA	Correct edges	3	4	3	3	2	4	3	0.75	0.89
		Wrong edges	2	0	2	2	3	4	1.8		
	QA	Correct edges	1	2	0	2	2	3	1.4	0.35	0.74
		Wrong edges	5	5	6	2	3	11	4.2		

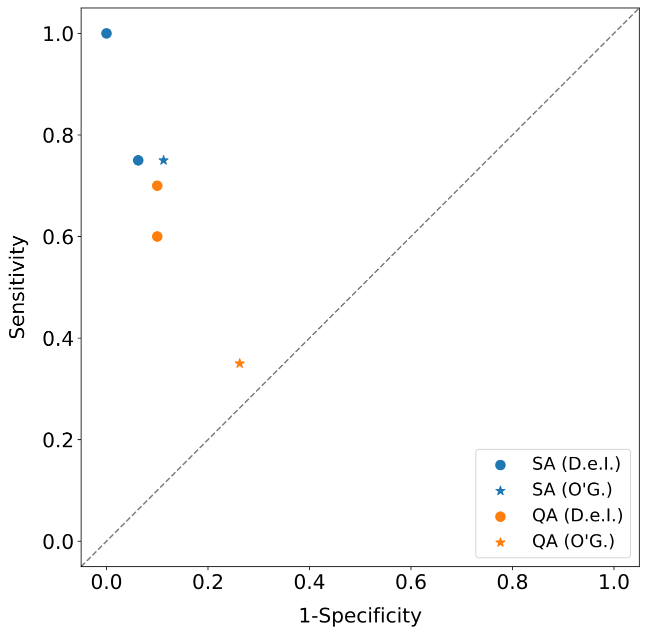


Fig. 4. Sensitivity versus (1 - Specificity) for the LC problem. This plot results from the data reported in Table 9.

the specificity, the better the result. It is also worth mentioning that SA with $k = 4$ has been able to find the perfect solution (four correct and zero wrong edges) in all five runs. In addition, by looking at the number of unique edges found across all runs (fifth column), it turns out that, for the divide et impera approach, SA tends to find always the same correct and wrong edges, whereas QA shows more variability, as well as O'Gorman's algorithm.

Concerning the Waste problem, whose results are reported in Table 10 and displayed in Figure 5, the overall performance is worse for both SA and QA. Specifically, also in this case, SA has performed better than QA overall. Only for $k = 3$, QA has been able to achieve better results on average. Instead, the superiority of the divide et impera approach w.r.t. the direct application of O'Gorman's algorithm has turned out to be less marked. In detail, for SA, the divide et impera approach has won the comparison for almost all (but not all) k values. Regarding QA, O'Gorman's algorithm has achieved a relatively high sensitivity (w.r.t. all QA results), but the corresponding specificity is quite low. In the end, for both methods (SA and QA), it is possible to find a k value for which the divide et impera approach has performed better than O'Gorman's algorithm. Actually, for this problem, the perfect solution has never been found. The best results have been achieved by SA with $k = 6$, which has discovered four correct edges out of nine in almost all runs and only two wrong edges on average. In addition, the maximum number of correct edges that have been found in a single run is equal to 6 (SA with $k = 4$). However, it is worth remarking that the problem in question has been subjected to a discretization procedure and includes a node with three parents. Eventually, in general, the observations made for LC on the number of unique edges found turn out to be valid also for the Waste problem. The only difference lies in the correct edges found by the divide et impera approach with QA; indeed, they tend to be the same across runs.

Table 10. Results achieved by the divide et impera approach on the Waste problem, for different numbers of variables per subproblem (k) and methods (SA/QA), using an *Exp* dataset of size $N = 10^4$, five runs, 100 reads for SA, and 100 reads and $1\mu s$ of annealing time for QA. The last k value (9) corresponds to the direct application of the implementation of O'Gorman's algorithm. In particular, D.e.I. = divide et impera, O'G. = O'Gorman, Sens. = sensitivity, Spec. = specificity.

Waste ($n = 9$, edges$\,=9$)											
k	Method	Metric	# for each run					# unique	Average #	Sens.	Spec.
3 (D.e.I.)	SA	Correct edges	2	2	2	2	1	2	1.8	0.20	0.88
		Wrong edges	8	7	7	8	9	10	7.8		
	QA	Correct edges	1	4	3	4	2	6	2.8	0.31	0.90
		Wrong edges	8	6	6	5	7	13	6.4		
4 (D.e.I.)	SA	Correct edges	2	6	2	3	2	6	3	0.33	0.94
		Wrong edges	5	1	5	4	5	5	4		
	QA	Correct edges	4	1	3	3	4	6	3	0.33	0.92
		Wrong edges	4	7	4	5	5	10	5		
5 (D.e.I.)	SA	Correct edges	2	3	4	3	3	5	3	0.33	0.94
		Wrong edges	5	4	3	2	4	7	3.6		
	QA	Correct edges	3	5	2	0	1	5	2.2	0.24	0.94
		Wrong edges	4	2	5	4	5	10	4.0		
6 (D.e.I.)	SA	Correct edges	4	4	4	4	3	5	3.8	0.42	0.97
		Wrong edges	2	1	3	1	3	4	2		
	QA	Correct edges	1	1	1	2	1	3	1.2	0.13	0.98
		Wrong edges	1	2	1	0	2	5	1.2		
7 (D.e.I.)	SA	Correct edges	5	4	1	3	3	5	3.2	0.36	0.96
		Wrong edges	1	1	5	3	3	6	2.6		
	QA	Correct edges	0	0	0	1	0	1	0.2	0.02	0.97
		Wrong edges	2	3	1	2	1	8	1.8		
8 (D.e.I.)	SA	Correct edges	2	1	4	5	2	5	2.8	0.31	0.96
		Wrong edges	3	4	2	0	3	6	2.4		
	QA	Correct edges	1	0	0	0	0	1	0.2	0.02	0.90
		Wrong edges	4	8	6	6	7	25	6.2		
9 (O'G.)	SA	Correct edges	3	3	3	1	2	5	2.4	0.27	0.86
		Wrong edges	9	8	8	9	9	25	8.6		
	QA	Correct edges	2	2	3	3	4	7	2.8	0.31	0.77
		Wrong edges	15	16	15	12	16	49	14.8		

Finally, the results related to the Alarm problem are reported in Table 11 and shown in Figure 6. In particular, the divide et impera approach with QA has not been evaluated in this case because the number of subproblems is really high and the sequential submission of numerous QUBO problems to the D-Wave's annealer tends to fail due to connectivity issues (from D-Wave's side), invalidating the run. As for the other problems, the divide et impera approach has outperformed the direct application of O'Gorman's algorithm. Indeed, the latter has won the comparison (with a worse specificity) only for $k = 12$. In addition, SA has achieved far better results than O'Gorman's algorithm with QA. Concerning the quality of the solution found, the best results have been achieved by SA with $k = 4$, which has been capable of detecting 12.6 correct edges out of 15 on average (note that the problem includes a variable with four parents). However, the number of wrong edges is quite high (16.4 on average). The same configuration has also discovered the highest number of correct edges (13). Eventually, it is worth making two last observations: the number of edges (both correct and wrong) detected by the divide et impera approach tends to decrease by increasing the value of k; the divide et impera approach tends to discover always the same correct and

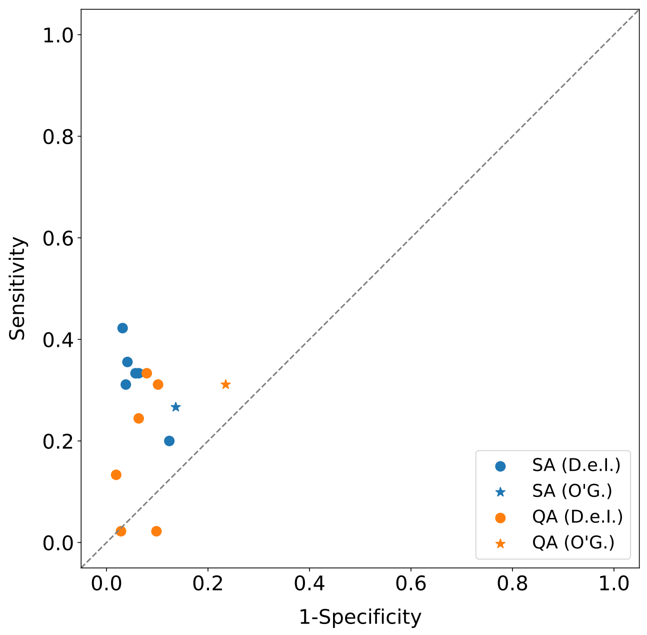


Fig. 5. Sensitivity versus (1 - Specificity) for the Waste problem. This plot results from the data reported in Table 10.

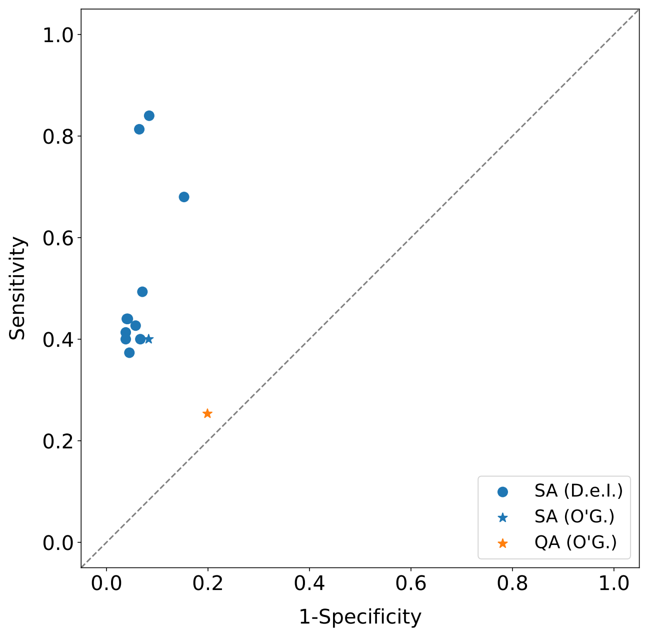


Fig. 6. Sensitivity versus (1 - Specificity) for the Alarm problem. This plot results from the data reported in Table 11.

Table 11. Results achieved by the divide et impera approach on the Alarm problem, for different numbers of variables per subproblem (k), using an *Exp* dataset of size $N = 10^4$, five runs, 100 reads for SA, and 100 reads and 1μs of annealing time for QA. The last k value (15) corresponds to the direct application of the implementation of O'Gorman's algorithm. In particular, D.e.I. = divide et impera, O'G. = O'Gorman, Sens. = sensitivity, Spec. = specificity.

Alarm ($n = 15$, edges $= 15$)											
k	Method	Metric	# for each run					# unique	Average #	Sens.	Spec.
3 (D.e.I.)	SA	Correct edges	10	10	11	10	10	12	10.2	0.68	0.85
		Wrong edges	31	31	30	29	28	38	29.8		
4 (D.e.I.)	SA	Correct edges	12	13	13	12	13	13	12.6	0.84	0.92
		Wrong edges	17	16	17	16	16	20	16.4		
5 (D.e.I.)	SA	Correct edges	12	12	12	12	13	13	12.2	0.81	0.94
		Wrong edges	13	13	13	12	12	15	12.6		
6 (D.e.I.)	SA	Correct edges	8	7	7	7	8	8	7.4	0.49	0.93
		Wrong edges	14	14	14	14	13	14	13.8		
7 (D.e.I.)	SA	Correct edges	6	6	6	6	6	6	6	0.40	0.93
		Wrong edges	13	13	13	13	13	13	13		
8 (D.e.I.)	SA	Correct edges	7	6	6	7	6	7	6.4	0.43	0.94
		Wrong edges	11	11	11	11	12	12	11.2		
9 (D.e.I.)	SA	Correct edges	7	6	7	7	6	7	6.6	0.44	0.96
		Wrong edges	8	9	7	7	9	10	8.0		
10 (D.e.I.)	SA	Correct edges	6	6	7	7	7	7	6.6	0.44	0.96
		Wrong edges	9	8	8	7	7	9	7.8		
11 (D.e.I.)	SA	Correct edges	6	7	7	7	6	7	6.6	0.44	0.96
		Wrong edges	8	8	8	8	9	9	8.2		
12 (D.e.I.)	SA	Correct edges	5	6	6	5	6	6	5.6	0.37	0.95
		Wrong edges	9	9	8	9	9	10	8.8		
13 (D.e.I.)	SA	Correct edges	5	7	6	7	6	8	6.2	0.41	0.96
		Wrong edges	9	6	8	7	7	9	7.4		
14 (D.e.I.)	SA	Correct edges	5	7	6	7	5	9	6	0.40	0.96
		Wrong edges	10	8	6	6	7	11	7.4		
15 (O'G.)	SA	Correct edges	4	5	6	6	9	9	6	0.40	0.92
		Wrong edges	18	16	16	16	15	55	16.2		
	QA	Correct edges	2	4	5	7	1	12	3.8	0.25	0.80
		Wrong edges	39	41	36	40	38	128	38.8		

wrong edges across runs (look at the fifth column), whereas O'Gorman's algorithm exhibits more variability. Actually, the wrong edges for $k = 3$ and the correct edges for O'Gorman's algorithm with SA represent two exceptions.

6 Conclusion

In this work, we have presented an implementation in Python of the algorithm proposed by O'Gorman et al. for solving the BNSL problem on a quantum annealer, a divide et impera approach that allows addressing BNSL instances with a higher number of variables, a complexity analysis of them, and their experimental evaluation. In detail, to make O'Gorman's formulation effectively usable, algebraic manipulations have been applied to the computation of the local scores $s_i(\Pi_i(B_s))$. Moreover, a simplified lower bound has been introduced for the penalty value $\delta_{consist}$, and the best setup of the α_{ijk} hyperparameters has been empirically determined. The results achieved in the experiments have demonstrated that O'Gorman's algorithm can be effectively used to reconstruct Bayesian networks of small sizes ($n <= 5$)

with less than three parents per node ($m < 3$). Instead, in presence of more Bayesian variables ($n = 9$), the algorithm performance have turned out to be worse. Indeed, good quality solutions (in terms of QUBO image value) have been found, but not the correct one. It is also worth remarking that one of these larger problems includes a node with three parents. In addition, the linear dependency between the dataset size and the QUBO matrix construction time has been confirmed. Eventually, QA (using the best annealing parameters) has been able to achieve comparable or slightly worse results with respect to SA, proving the competitiveness of the current annealing architectures on this task.

Concerning the divide et impera approach, which has been developed to overcome the limitation on the problem size dictated by the available annealing devices, the results have demonstrated that it performs better than the direct application of O'Gorman's algorithm. Indeed, in all problems considered, for all resolution methods tested, there is more than one k value for which the divide et impera approach has achieved better results; actually, in almost all cases, these k values represent the majority. Instead, in general, the quality (in terms of resulting Bayesian network) of the solutions found has turned out to be not optimal. However, non-ideal annealing parameters have been used for QA due to the limited quantum resources at our disposal, and the number of reads for SA has also been reduced (w.r.t. the value used for the experiments on O'Gorman's algorithm) for a fair comparison. Moreover, in this second set of experiments, unlike in the first one, SA has performed definitely better than QA; nevertheless, this is probably related to the less-performing parameters used here. Finally, the experiments on the subproblems formulation and QUBO matrices construction time have confirmed the effectiveness of the technique used to speed up the execution (and also the independence of the times from the variance in the input data).

Future work includes the following possibilities: testing the divide et impera approach on Bayesian problems whose size is larger than the maximum size embeddable in the Pegasus architecture using O'Gorman's algorithm; evaluating the aforementioned approach with QA using more-performing annealing parameters; analysing the impact of considering only part of the subproblems of size k. We conclude by reminding that the code of both the implementation of O'Gorman's algorithm and the divide et impera approach is available under the GPLv2 licence [9, 10].

Acknowledgements

This work was supported by Q@TN, the joint lab between University of Trento, FBK-Fondazione Bruno Kessler, INFN-National Institute for Nuclear Physics and CNR-National Research Council. In addition, the authors gratefully acknowledge the Jülich Supercomputing Center (`https://www.fz-juelich.de/ias/jsc`) for funding this project by providing computing time through the Jülich UNified Infrastructure of Quantum computing (JUNIQ) on the D-Wave quantum annealer.

References

1. J. Pearl (1985), *Bayesian networks: A model of self-activated memory for evidential reasoning*, Proceedings of the 7th conference of the Cognitive Science Society, pp. 15-17.
2. P. Spirtes and C. Meek (1995), *Learning Bayesian networks with discrete variables from data*, KDD, Vol.1, pp. 294-299.

3. J. Pearl (1995), *From Bayesian networks to causal networks*, Mathematical models for handling partial knowledge in artificial intelligence, pp. 157-182.
4. B. O'Gorman and R. Babbush and A. Perdomo-Ortiz and A. Aspuru-Guzik and V. Smelyanskiy (2015), *Bayesian Network Structure Learning Using Quantum Annealing*, The European Physical Journal Special Topics, Vol.224, Num.1, pp. 163–188.
5. M. Ozols and M. Roetteler and J. Roland (2013), *Quantum rejection sampling*, ACM Transactions on Computation Theory (TOCT), Vol.5, Num.11.
6. S. E. Borujeni and S. Nannapaneni and N. H. Nguyen and E. C. Behrman and J. E. Steck (2021), *Quantum circuit representation of Bayesian networks*, Expert Systems With Applications, Vol.176.
7. D-Wave Systems Inc., *D-Wave Systems*, `https://www.dwavesys.com` [online, last access on 23 February 2022].
8. T. Kadowaki and H. Nishimori (Nov 1998), *Quantum annealing in the transverse Ising model*, Phys. Rev. E, Vol.58, pp. 5355-5363.
9. M. Rizzoli (2021), *Implementation of O'Gorman's algorithm*, `https://github.com/massimo-rizzoli/BNSL-QA-python`.
10. S. Dissegna (2021), *Implementation of the divide et impera approach*, `https://github.com/sebdisdv/BNSL`.
11. F. Glover and G. Kochenberger and Y. Du (2019), *Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models*, 4OR - A Quarterly Journal of Operations Research, Vol. 17.
12. D-Wave Systems Inc., *Minor embedding*, `https://docs.dwavesys.com/docs/latest/c_gs_3.html#minor-embedding` [online, last access on 20 October 2021].
13. D-Wave Systems Inc., *Minor embedding example*, `https://docs.dwavesys.com/docs/latest/c_gs_7.html#getting-started-embedding` [online, last access on 20 October 2021].
14. D-Wave Systems Inc., *Embedding Composite*, `https://docs.ocean.dwavesys.com/en/stable/docs_system/reference/composites.html#embeddingcomposite` [online, last access on 20 October 2021].
15. D. M. Chickering (1996), *Learning Bayesian Networks is NP-Complete*, Learning from Data: Artificial Intelligence and Statistics V, pp. 121-130.
16. Anaconda Inc. and others, *Numba, compiling Python code with @jit*, `https://numba.readthedocs.io/en/stable/user/jit.html` [online, last access on 12 August 2021].
17. Bayes Server, *Bayesian network examples*, `https://www.bayesserver.com/examples` [online, last access on 22 June 2021].
18. E. Correa and A. Freitas and C. Johnson (2007), *Particle Swarm and Bayesian Networks Applied to Attribute Selection for Protein Functional Classification*, Proceedings of GECCO 2007: Genetic and Evolutionary Computation Conference, pp. 2651-2658.
19. S. Kirkpatrick and C. D. Gelatt and M. P. Vecchi (1983), *Optimization by Simulated Annealing*, Science, Vol.220, Num.4598, pp. 671-680.
20. D-Wave Systems Inc., *D-Wave solver parameters*, `https://docs.dwavesys.com/docs/latest/c_solver_parameters.html` [online, last access on 20 October 2021].
21. D-Wave Systems Inc., *D-Wave simulated annealing sampler*, `https://docs.ocean.dwavesys.com/projects/neal/en/latest/reference/sampler.html` [online, last access on 29 November 2021].
22. D. Heckerman and D. Geiger and D. M. Chickering (1995), *Learning Bayesian Networks: The Combination of Knowledge and Statistical Data*, Machine Learning, Vol.20, Num.3, pp. 197-243.
23. D-Wave Systems Inc., *Operation and Timing*, `https://docs.dwavesys.com/docs/latest/c_qpu_timing.html` [online, last access on 13 December 2021].