Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Final Dissertation

Implementation and testing of Bayesian network structure learning using quantum annealing

Supervisor
Enrico Blanzieri

Student
Massimo Rizzoli

Cosupervisors
Davide Pastorello
Enrico Zardini

Academic year 2020/2021

Contents

Abstract

Bayesian Network Structure Learning (BNSL) is a machine learning method which aims at extracting conditional dependency information for a given set of variables from data in the form of examples, each containing a setting for the states of all variables. O'Gorman et al. [1] published an article proposing a formal method for representing BNSL problems into an equivalent Quadratic Unconstrained Binary Optimisation (QUBO) formulation. Such a formulation is computationally equivalent to the Ising model, which makes QUBO the go-to problem for quantum annealing, a metaheuristic process for finding the global minimum, taking advantage of various quantum mechanical phenomena, such as tunnelling, superposition and entanglement.

The objective is to implement O'Gorman et al. formulation and test its performance on the state of the art quantum computer D-Wave Advantage. The proposed mapping is mathematically expressed into a Hamiltonian function that represents the QUBO formulation. This function uses several intermediate values that are the first to be computed. The first difficulty manifested with the need to compute several factorials and their product while trying to implement the calculation of one of these values. The solution is to change the function that calculates it through algebraic manipulations, so that the resulting function is equivalent to the original and so that its result is computable. Once all the intermediate values are computed, the Hamiltonian has to be converted into a QUBO matrix for the variables contained in it. To do this, the Hamiltonian is executed step by step, memorising every coefficient in the position corresponding to the variables it is multiplied with, inside the QUBO matrix. Before testing the approach on the target quantum computer, some further steps are needed. Firstly the algorithm has to be tested for correctness and suitable α hyperparameters values need to be found. To test the algorithm, several problems with different amounts of variables are used for generating datasets of various sizes, both with variability and with no variance. As a first approach to test the algorithm, an exhaustive search using brute force to find the minimum value is used. It is then improved with several optimisations to make its use possible on larger problems. The first tests are performed with the exhaustive search and are essential to find suitable values for the α hyperparameters to make the algorithm behave as expected. The complexity of the algorithm is calculated to better understand what factors have the highest impact on the computation time. With the algorithm working, the QUBO problems are then submitted to the quantum computer. This is done by using the D-Wave Ocean Python library, which allows the submission of problems in both QUBO and Ising form.

Several tests are performed using different datasets representing problems of different sizes, with different settings for the quantum annealer's parameters regarding the number of reads and the annealing time per read. From this data, both the algorithm's performance for constructing the QUBO matrix and the quantum annealer's performance are evaluated. The QUBO computation time increases with larger BNSL problem size, larger datasets and larger number of states for the Bayesian variables. The quantum annealer's performance is maximised by increasing first the number of reads to the maximum value and then increasing the annealing time per read, until a joint limit is reached. Quantum annealing performed worse than simulated annealing even in its best settings. This is due to the highly connected nature of the BNSL problem in QUBO form and the sparsely connected nature of the quantum computer's physical graph.

1 Introduction

In this introductory section, the theory necessary for understanding the problem and for the implementaion is reported, discussing QUBO problems, the quantum technology to be used and O'Gorman et al. [1] QUBO formulation for BNSL problems.

1.1 QUBO problems

A Quadratic Unconstrained Binary Optimisation problem is an optimisation problem of the form

$$\arg\min_x x^T Q x \tag{1.1}$$

in the minimisation case, where

- x is a vector of binary decisions;
- Q is a square matrix of constants.

Q is an upper triangular (or equivalently a symmetric) matrix that contains all problem data. This data can be divided into *linear* and *quadratic* coefficients. For x being an $n \times 1$ vector and Q being an $n \times n$ upper triangular square matrix

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

where $x_i = x_i^2$ holds because $x_i \in \mathbb{B}$. The linear coefficients q_{ii} correspond to the main diagonal of Q and are the weights applied to each x_i individually. The quadratic coefficients q_{ij} correspond to the rest of the upper triangular matrix and are the weights for the quadratic terms $x_i x_j$. As per its name and definition (1.1), QUBO is an unconstrained problem.

1.1.1 Constrained problems

Some problems can be naturally mapped into QUBO problems, but the largest number of problems of interest by far require additional constraints that must be satisfied as the optimiser searches for good solutions[3]. Glover et al. [3] provide a general approach to transform binary quadratic problems subject to linear constraints to QUBO problems. Let

$$\arg\min_x x^T C x$$

$$\text{subject to: } Ax = b$$

be a quadratic problem with x being an $n \times 1$ binary decision vector, C being an $n \times n$ upper diagonal square matrix of coefficients, subject to m constraints given by A, an $m \times n$ matrix of coefficients and b, an $m \times 1$ vector of constants. $Ax = b$ can natively represent equality constraints of the form $a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n = b_i$. To instead encode inequality constraints of the form

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \leq b_i$$

assuming that A and b have integer components, the introduction of a slack variable is sufficient for the conversion, obtaining

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n + s = b_i$$

where $s \in \mathbb{Z}$ and $s \leq b_i$. With a process of *binary expansion*, s can be represented by $\mu = \lceil log_2(b_i+1) \rceil$ bits as

$$s = \sum_{l=1}^{\mu} 2^{l-1} s_l$$

By doing so, the value for s is limited between 0 and $2^\mu - 1$ and s is expressed with binary slack variables that become part of the optimisation problem.
If all the constraints are respected, $Ax - b = 0$ holds. It can be shown that $(Ax-b)^T(Ax-b) = x^T D'x + c'$, obtaining a mapping to a QUBO problem encoding the constraints, with an additive constant c'. This allows to merge the unconstrained problem with its constraints

$$\begin{aligned} & x^T Cx + P(Ax-b)^T(Ax-b) = \\ & = x^T Cx + P(x^T D'x + c') \\ & = x^T Cx + x^T PD'x + Pc' \\ & = x^T Cx + x^T Dx + c = x^T(C+D)x + c \\ & = x^T Qx \end{aligned}$$

where P is a positive scalar penalty that will be added for every violated constraint and c is an additive constant that can be ignored for the optimisation problem. The obtained Q matrix encodes both the problem and its constraints.

1.1.2 Why QUBO?

The QUBO problem is computationally equivalent to the Ising model, with the difference that QUBO variables take Boolean values $\{0, 1\}$ while Ising variables take spin values $\{-1, +1\}$. Quantum annealing problems are described using the Ising model, making QUBO, because of its similarity, the most natural way of encoding a problem for it to be solved by a quantum annealer.

1.2 D-Wave Systems

D-Wave Systems is the sole producer of commercial quantum computers that perform quantum annealing. These quantum computers are also made available for problem submission for a limited time each month, with one minute of access time for the free account. The available models are the D-Wave 2000Q featuring the *Chimera* topology and the latest D-Wave Advantage featuring the *Pegasus* topology.

1.2.1 QPU topologies

The Chimera topology contains 2048 qubits, each of which is connected to 6 other qubits. The Pegasus topology contains 5640 qubits, each of which is connected to 15 other qubits. While a higher amount of qubits allows for larger problems to be submitted, the most interesting feature is the higher connectivity, that allows for both a native representation of denser problems and minor-embedded representations with less and overall shorter qubit chains. For these reasons and the densely connected nature of the QUBO formulation of the BNSL problem, only D-Wave Advantage has been used for the tests.

1.2.2 Quantum annealing

D-Wave systems implement a process called *quantum annealing*, a metaheuristic for finding the global minimum of a given objective function, exploiting the quantum mechanical phenomena of tunneling to escape local minima, the one of superposition to simultaneously evaluate different regions of the energy landscape and the one of entanglement to find correlations between low energy states [6].

1.3 Bayesian network structure learning using quantum annealing

A Bayesian network (BN) is a directed acyclic graph (DAG) representation of conditional dependencies for a set of random variables. The variables are represented by the nodes of the graph while the conditional dependencies are the directed edges between them. The probability function for the conditional distribution of the node given its parents is associated to each node.

The method proposed by O'Gorman et al. [1] focuses on the structure learning problem (BNSL), for which the goal is to find the Bayesian network most likely to have generated a given set of data. The problem is known to be NP-Complete and though is believed that quantum computers cannot solve this class of problems efficiently [4], the authors expect a polynomial speedup with quantum annealing. To take advantage of the new technology, a hardware compatible QUBO formulation of the BNSL problem is provided. Sufficient lower bounds for the penalties have been proved, guaranteeing the lowest scoring solutions to be valid solutions.

1.3.1 Bayesian network

A Bayesian network B for n random variables $\mathbf{X} = (X_i)_{i=1}^n$ is a pair (B_s, B_p) where:

- B_s is a DAG representing the structure;
- B_p is the set of n conditional probalilities $\{p(X_i|\Pi_i(B_s))|1 \leq i \leq n\}$ that give the probability distribution for the state of each variable X_i on the joint state of the parent set $\Pi_i(B_s)$ (Π_i for short, when it is clear from the context).

Let:

- r_i be the number of states of X_i;
- $q_i = \prod_{j \in \Pi_i} r_j$ be the number of joint states of the parent set Π_i of X_i;
- x_{ik} indicate the k-th state of variable X_i;
- π_{ij} indicate the j-th joint state of the parent set Π_i;
- $\theta_{ijk} = p(x_{ik}|\pi_{ij})$ be the probability of X_i being in its k-th state conditioned on its parent set Π_i being in its j-th joint state;
- B_p consist of the n probability distributions $(((\theta_{ijk})_{k=1}^{r_i})_{j=1}^{q_i})_{i=1}^n$.

1.3.2 Structure learning

Given a database $D = \{\mathbf{x}_i|1 \leq i \leq N\}$ consisting of N cases, where each $\mathbf{x}_i$ denotes the state of all variables $\mathbf{X}$, the goal is to find the structure that maximises the posterior distribution $p(B_s|D)$ out of all possible structures.

In the work, proportionality between $p(B_s|D)$ and $p(D|B_s)$ is shown and Dirichlet priors are assumed, resulting in the following objective:

$$p(D|B_s) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})} \tag{1.2}$$

where

- Γ is the gamma function extending the factorial to real numbers;
- N_{ijk} is the number of cases in D such that variable X_i is in its k-th state and its parent set Π_i is in the j-th state;
- $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$;
- α_{ijk} is the hyperparameter for θ_{ijk} in the Dirichlet distribution from which θ_{ijk} is assumed to be drawn;

- $\alpha_{ij} = \sum_{k=1}^{r_i} \alpha_{ijk}$.

1.3.3 BNSL to QUBO

The QUBO encoding of the BNSL problem is given by a pseudo-Boolean function $H : \mathbb{B}^S \to \mathbb{R}$ called *Hamiltonian*. This function is divided into subfunctions: the *Score Hamiltonian* H_{score} responsible for the evaluation of the solution graph, the *Max Hamiltonian* H_{max} responsible for the penalisation of solutions that include a number of parents for a node greater than m, the *Cycle Hamiltonian* H_{cycle} further divided in *Consistency Hamiltonian* $H_{consist}$ and *Transitivity Hamiltonian* H_{trans} that together are responsible for the penalisation of solutions containing cycles.
The full Hamiltonian

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

is expanded in detail in the following sections.

To encode the original problem $n(n-1)$ bits $\mathbf{d} = (d_{ij})_{\substack{1 \le i < j \le n \\ i \ne j}}$ are used. Each d_{ij} indicates whether an edge from X_i to X_j is present ($d_{ij} = 1$) or not ($d_{ij} = 0$). This way the matrix whose entries are $\{d_{ij}\}$ (and $d_{ii} = 0$) is the adjacency matrix of the directed graph representing the Bayesian network structure of the solution.

1.3.4 Score Hamiltonian

The Score Hamiltonian is calculated separateley for each variable:

$$H_{score}(\mathbf{d}) = \sum_{i=1}^{n} H^{(i)}_{score}(\mathbf{d}_i) \tag{1.3}$$

where $\mathbf{d}_i \equiv (d_{ji})_{\substack{1 \le j \le n \\ j \ne i}}$, i.e. ite includes all the bits that encode edges towards X_i. The Score Hamiltonian for variable X_i is computed as

$$H^{(i)}_{score}(\mathbf{d}_i) = \sum_{\substack{J \subset \{1..n\} \setminus \{i\} \\ |J| \le m}} \left(w_i(J) \prod_{j \in J} d_{ji} \right) \tag{1.4}$$

where m is the largest size for a parent set to be considered in the calculation and w_i is computed as follows

$$w_i(J) = \sum_{l=0}^{|J|} (-1)^{|J|-l} \sum_{\substack{K \subset J \\ |K| = l}} s_i(K) \tag{1.5}$$

The score values s_i are obtained from (1.2), introducing a logarithm for numerical efficiency. Let

$$s_i(\Pi_i(B_s)) = -\ln \left(\prod_{j=1}^{q_i} \frac{\Gamma(\alpha_{ij})}{\Gamma(N_{ij} + \alpha_{ij})} \prod_{k=1}^{r_i} \frac{\Gamma(N_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})} \right) \tag{1.6}$$

so that

$$\log p(D|B_s) = -\sum_{i=1}^{n} s_i(\Pi_i(B_s)).$$

1.3.5 Max Hamiltonian

The Max Hamiltonian is calculated separateley for each variable:

$$H_{max}(\mathbf{d}, \mathbf{y}) = \sum_{i=1}^{n} H^{(i)}_{max}(\mathbf{d}_i, \mathbf{y}_i) \tag{1.7}$$

where $H^{(i)}_{max}$ is zero if X_i has at most m parents and carries a positive penalty otherwise. More precisely, for variable X_i

$$H^{(i)}_{max}(\mathbf{d}_i, \mathbf{y}_i) = \delta^{(i)}_{max}(m - d_i - y_i)^2 \tag{1.8}$$

where

- m is the maximum allowed number of parents;
- $\delta^{(i)}_{max} > 0$ is the penalty weight;
- $d_i \equiv |\mathbf{d}_i| = \sum_{1 \le j \le n} d_{ji}$ is the in-degree for variable X_i calculated from the edge bits;
- $y_i \in \mathbb{Z}$, $y_i = \sum_{l=1}^{\mu} 2^{l-1} y_{il}$ is a slack value to allow $H^{(i)}_{max}$ to be zero when $d_i < m$, encoded via binary expansion with $\mu = \lceil \log_2(m+1) \rceil$ bits in $\mathbf{y}_i = (y_{il})_{l=1}^{\mu}$.

1.3.6 Cycle Hamiltonian

Since a directed graph must be acyclic to be a valid solution, it is necessary to penalise the structures that encode directed cycles in $\{d_{ij}\}$. To do so, additional Boolean variables $\mathbf{r} = (r_{ij})_{1 \le i < j \le n}$ encoding a topological order are added such that

- $r_{ij} = 1$ represents the case in which X_i preceeds X_j;
- $r_{ij} = 0$ represents the case in which X_i succeeds X_j.

The Cycle Hamiltonian is thus defined as

$$H_{cycle}(\mathbf{d}, \mathbf{r}) = H_{trans}(\mathbf{r}) + H_{consist}(\mathbf{d}, \mathbf{r}) \tag{1.9}$$

where

- H_{trans} is zero if the relation encoded in $\{r_{ij}\}$ is transitive, a positive penalty otherwise;
- $H_{consist}$ is zero if the order encoded in $\{r_{ij}\}$ is consistent with the directed edge structure encoded in $\{d_{ij}\}$ or a positive penalty otherwise.

A tournament is a directed graph in which every pair of vertices is connected by one and only one of the two possible directed edges. The Transitivity Hamiltonian only needs to penalise 3-cycles since if a tournament has any cycle it has a cycle of length three, and is calculated as follows on all possible 3-sets of variables:

$$H_{trans} = \sum_{1 \le i < j < k \le n} H^{(ijk)}_{trans}(r_{ij}, r_{ik}, r_{jk}) \tag{1.10}$$

For a single 3-set of variables $\{X_i, X_j, X_k\}$, $H^{(ijk)}_{trans}$ is computed as follows:

$$H^{(ijk)}_{trans}(\mathbf{r}) = \delta^{(ijk)}_{trans}\left[r_{ik} + r_{ij}r_{jk} - r_{ij}r_{ik} - r_{jk}r_{ik}\right] \tag{1.11}$$

where $\delta^{(ijk)}_{trans}$ is the positive penalty added if the variables $\{X_i, X_j, X_k\}$ form a 3-cycle.
The Consistency Hamiltonian needs to penalise any state where the topological order is inconsistent with the directed graph, i.e. penalise solutions when $r_{ij} = 1$ and $d_{ji} = 1$, and when $r_{ij} = 0$ and $d_{ij} = 1$:

$$H_{consist}(\mathbf{d}, \mathbf{r}) = \sum_{1 \le i < j \le n} H^{(ij)}_{consist}(d_{ij}, d_{ji}, r_{ij}) \tag{1.12}$$

For a pair of variables $\{X_i, X_j\}$, $H^{(ij)}_{consist}$ is computed as follows:

$$H^{(ij)}_{consist}(d_{ij}, d_{ji}, r_{ij}) = \delta^{(ij)}_{consist}(d_{ji}r_{ij} + d_{ij} - d_{ij}r_{ij}) \tag{1.13}$$

where $\delta^{(ij)}_{consist}$ is the positive penalty added if there is inconsistency between the topological order and the directed graph with respect to the variables $\{X_i, X_j\}$.
Note that the permutation of the superscript indices for the penalties $\delta^{(ijk)}_{trans}$ and $\delta^{(ij)}_{consist}$ always refers to the same set of variables, making penalties with any permutation of these indices equivalent.

1.3.7 QUBO formulation size

The sets of variables that encode the BNSL problem into a QUBO problem have the following sizes:

$$\begin{aligned} |\{d_{ij}\}| &= n(n-1) \\ |\{r_{ij}\}| &= \frac{n(n-1)}{2} \\ |\{y_{il}\}| &= n\mu \end{aligned} \tag{1.14}$$

This means that, due to the encoding, $\mathcal{O}(n^2)$ qubits are required for n original Bayesian variables. Since H_{score} contains m-degree terms (multiplications with m factors), if $m \geq 3$ the formulation becomes polynomial and an additional step and additional slack variables are required to convert it back to a quadratic equation. In O'Gorman et al. [1] is stated that for $m = 3$, $n\lfloor\frac{(n-2)^2}{4}\rfloor$ slack variables are needed for the reduction. This raises the number of qubits needed per original Bayesian variable to $\mathcal{O}(n^3)$, furhter reducing the maximum considerable problem size.

1.3.8 Penalties

O'Gorman et al. provide lower bounds for the penalties, also proving their sufficiency:

$$\delta^{(i)}_{max} > \max_{j \neq i} \Delta_{ji}, 1 \leq i \leq n \tag{1.15}$$

$$\delta^{(ijk)}_{trans} = \delta_{trans} > \max_{\substack{1 \leq i', j' \leq n \\ i' \neq j'}} \Delta_{i'j'}, 1 \leq i < j < k \leq n \tag{1.16}$$

$$\delta^{(ij)}_{consist} > (n-2) \max_{k \notin \{i,j\}} \delta^{(ijk)}_{trans}, 1 \leq i < j \leq n \tag{1.17}$$

where Δ_{ji} is a worst case estimate of the greatest increase to the score that adding the arc from X_j to X_i can produce, and (for $m = 2$) is calculated as

$$\Delta_{ji} = \max\{0, \Delta'_{ji}\} \tag{1.18}$$

$$\Delta'_{ji} = -w_i(\{j\}) - \sum_{1 \leq k \leq n} \min(0, w_i(\{j,k\})) \tag{1.19}$$

For $m \geq 3$, finding the value for Δ_{ji} is an intractable optimisation problem, so only a "reasonable bound" is provided.

2 Implementation

The implemetation is written in Python. The choice of the language is binded with D-Wave's Ocean suite, which is necessary for the interaction with the quantum computers and is implemented in Python. In this section, the pseudocode of the implementation of the BNSL encoding into QUBO is reported.

2.1 High level view

Algorithm 1 is responsible for the full construction of Q and provides a high level view of the process. Starting only from dataset data, namely the number of variables n, the number of states r_i for each variable and all of the examples in the dataset, it produces the QUBO matrix Q by utilising the algorithms described later in this section. Before calling the functions to map the Hamiltonian into the matrix, it initialises Q to all zeros. This way the Hamiltonian components can just be added to Q.

Data: number of bayesian variables n, $r = (r_i)_{i=1}^{n}$ where r_i is the number of states of the i^{th} bayesian variable, list *examples* of states for all variables generated form the structure to be learned
Result: QUBO matrix Q of size $size(indexQUBO) \times size(indexQUBO)$ representing the QUBO problem

```
// calculation of the values necessary to construct Q
parentSets ← calcParentSets(n);                                   // Algorithm 2
α ← calcAlpha(n, r, parentSets);                                  // Algorithm 3
s ← calcS(n, r, parentSets, α, examples);
;                                                                 // Algorithm 4
w ← calcW(n, parentSets, s);                                      // Algorithm 5
Δ ← calcDelta(n, parentSets, w);                                  // Algorithm 6
δ_max ← calcDeltaMax(n, Δ);                                       // Algorithm 7
δ_trans ← calcDeltaTrans(n, Δ);                                   // Algorithm 8
δ_consist ← calcDeltaConsist(n, δ_trans);                         // Algorithm 9
// construction of Q
indexQUBO ← calcIndexQUBO(n);                                     // Algorithm 10
posOfIndex ← calcPosOfIndex(indexQUBO);                           // Algorithm 11
Q ← Vector(size(indexQUBO)) for row ← 1 to size(indexQUBO) do
    Q[row] ← Vector(size(indexQUBO));
    for col ← 1 to size(indexQUBO) do
        Q[row][col] ← 0;
    end
end
Q ← addHscore(Q, posOfIndex, n, parentSets, w);                   // Algorithm 12
Q ← addHmax(Q, posOfIndex, n, δ_max);                             // Algorithm 13
Q ← addHtrans(Q, posOfIndex, n, δ_trans);                         // Algorithm 14
Q ← addHconsist(Q, posOfIndex, n, δ_consist);                     // Algorithm 15
return Q;
```

Algorithm 1: *calcQUBOMatrix()*

2.2 Values calculation

Before computing the QUBO matrix it is necessary to compute the several intermediate values, namely the parent sets, hyperparameters α, local scores s, parent set weights w and the penalties δ.

First, all the possible parent sets for each Bayesian variable are calculated. For the reasons stated in sections 1.3.7 and 1.3.8, the maximum number of parents m will be set to 2. Because of this, Algorithm 2 performs this computation with a complexity of $\mathcal{O}(n^3)$. Note that the empty parent set is a valid parent set.

Input: number of bayesian variables n
Result: list *parentSets* of all possible parent sets for each of the n bayesian variables

```
function calcSubSets(s):
    sSet ← Vector();
    sSet.append(∅);
    // all singletons
    for i ∈ s do
        sSet.append({i})
    end
    // all sets of 2
    for i ∈ s do
        for j ∈ s \ {i} do
            sSet.append({i, j})
        end
    end
    return sSet;
parentSets ← Vector(n);
for i ← 1 to n do
    maxSet ← {1...n} \ {i};
    parentSets[i] ← calcSubSets(maxSet);
end
return parentSets;
```

Algorithm 2: *calcParentSets(n)*

Next, Algorithm 3 sets all the hyperparameters α to the uninformative $1/(r_i \cdot q_i)$, which for all α related to a variable i and a parent set π for that variable, has the same value. For more details on this choice see Section 3. The algorithm has to generate α values for all possible variable - parent set - parent set state - variable state combinations, with a complexity of $\mathcal{O}(n^3 r_{max}^3)$ where r_{max} is the highest number of states among the Bayesian variables.

Input: number of bayesian variables n, $r = (r_i)_{i=1}^n$ where r_i is the number of states of the i^{th} bayesian variable, list *parentSets* of all possible parent sets for each of the n bayesian variables
Result: hyperparameter α for the prior distribution

```
for i ← 1 to n do
    for π ∈ parentSets[i] do
        q_i ← 1;
        for p ∈ π do q_i ← q_i · r_p;
        for j ← 1 to q_i do
            for k ← 1 to r_i do
                α_iπjk ← 1/(r_i · q_i);
            end
        end
    end
end
return α;
```

Algorithm 3: *calcAlpha(n, r, parentSets)*

With these values calculated and given a dataset, Algorithm 4 is now able to compute the local scores s for all possible Bayesian variable - parent set combinations as per equation (1.6). Due to the factorial increasing nature of the Γ function and multiplications between its values, the calculation of s is possible only for very small datasets because of the extremely large values that get out of range. The solution is to move the logarithm inside, through algebraic steps, so that it always includes only the gamma function:

$$s_i(\Pi_i(B_s)) = -\left(\sum_{j=1}^{q_i}\left[\ln(\Gamma(\alpha_{ij})) - \ln(\Gamma(N_{ij}+\alpha_{ij})) + \sum_{k=1}^{r_i}[\ln(\Gamma(N_{ijk}+\alpha_{ijk})) - \ln(\Gamma(\alpha_{ijk}))]\right]\right) \quad (2.1)$$

This form allows for the use of the log-gamma (written as $\ln\Gamma$ in the pseudocode) function instead of the gamma one, while still obtaining the same results. This function grows much more slowly and, as a collateral effect, the products are substituted by additions, further decreasing the risk of having out of range values. In fact, it works without any problem even with datasets containing one million examples. The algorithm has a complexity of $\mathcal{O}(n^3 N r_{max}^3)$.

Finally, with s calculated, the parent set weights w for the Score Hamiltonian can be computed by Algorithm 5 according to (1.5), with a complexity of $\mathcal{O}(n^3)$.

Next, the penalties need to be calculated, starting form the intermediate values Δ computed according to (1.18) and (1.19) by Algorithm 6 with a complexity of $\mathcal{O}(n^4)$.

With Δ, all the penalties can be calculated. Algorithm 7 computes the penalties $\delta_{max}^{(i)}$ corresponding to each Bayesian variable according to (1.15) with a complexity of $\mathcal{O}(n^2)$. The proposed penalty bounds (1.16) and (1.17) can be further simplified due to the fact that the suitable δ_{trans} is independent of its indices:

$$\delta_{trans} > \max_{\substack{1 \le i,j \le n \\ i \neq j}} \Delta_{ij} \tag{2.2}$$

$$\delta_{consist} > (n-2)\delta_{trans} \tag{2.3}$$

Algorithm 8 computes the δ_{trans} penalty in accordance with the simplified bound (2.2) with $\mathcal{O}(n^2)$ complexity, while Algorithm 9 computes the $\delta_{consist}$ penalty with the simplified bound (2.3) with $\mathcal{O}(1)$ complexity. For all the penalties, to enforce the ">" constraint, a value of 1 has been added appropriately.

2.3 QUBO matrix calculation

With all the values calculated, the next step is to construct the QUBO matrix Q. To do this, first it is necessary to index the rows and columns of Q and equally the positions of the solution vector x. Algorithm 10 does just that: it creates the $indexQUBO$ index containing tuples representing the QUBO variables at each position. These positions will be used by both Q and x. The complexity for this computation is $\mathcal{O}(n^2)$. Inside the index, the groups of variables are ordered starting from the edge bits d, then the slack bits for the Max Hamiltonian y and finally the topological order bits r for the Cycle Hamiltonian. It is also useful to know the position inside $indexQUBO$ of any given variable. For this purpose, Algorithm 11 creates the $posOfIndex$ dictionary with the tuples representing the variables as keys and the corresponding position of the variables as values, having a complexity of $\mathcal{O}(n^2)$.

With the index system sorted, it is now possible to translate the Hamiltonian to the QUBO matrix Q. First, the coefficients calculated from the Score Hamiltonian according to (1.3) and (1.4) are added to Q by Algorithm 12, with a complexity of $\mathcal{O}(n^3)$. For each Bayesian variable X_i and related parent sets combinations, all $w_i(J)$ are calculated. These values are the coefficients for linear and quadratic terms. The linear terms d_{ji} are composed of only one QUBO variable and their coefficient corresponds to the element of the diagonal of Q with row and column equal to d_{ji}. The quadratic terms $d_{j_1 i} d_{j_2 i}$ are composed of two QUBO variables and their coefficient corresponds to the element outside the diagonal of Q with row equal to $d_{j_1 i}$ and column $d_{j_2 i}$, assuming that $d_{j_1 i}$ comes first in $indexQUBO$. With this ordering, Q becomes an upper triangular matrix.

A similar approach is used for the Max Hamiltonian according to (1.7) and (1.8) with Algorithm 13, but taking into account the square. For each variable X_i, the coefficients in $H_{max}^{(i)}$ of all the variables are stored. Then the square is calculated and the resulting new linear and quadratic coefficients, after being multiplied by $\delta_{max}^{(i)}$, are added to Q at the coordinates indicated by the terms. The resulting algorithm has a complexity of $\mathcal{O}(n^3)$.

For the Transitivity Hamiltonian and the Consistency Hamiltonian the operations are simpler, thanks to the absence of squares and the reduced amount of terms in the formulas. Algorithm 14 adds the H_{trans} penalties to Q in accordance with (1.10) and (1.11): for each 3-set of variables $\{X_i, X_j, X_k\}$ it adds the coefficients of the linear and quadratic terms present in $H_{trans}^{(i)}$ multiplied by the δ_{trans} penalty to the corresponding positions. The complexity is $\mathcal{O}(n^3)$. Similarly, Algorithm 15 adds the $H_{consist}$ penalties to Q in accordance with (1.12) and (1.13): for the edges between all possible pairs

Input: number of bayesian variables n, $r = (r_i)_{i=1}^n$ where r_i is the number of states of the i^{th} bayesian variable, list *parentSets* of all possible parent sets for each of the n bayesian variables, hyperparameter α for the prior distribution, list *examples* of states for all variables generated form the structure to be learned

Result: $s = (s_i(\pi), \pi \in parentSets[i])_{i=1}^n$ where $s_i(\pi)$ is the score for the bayesian variable i given the parent set π

function *calcJthState(j, π, r)*:
    // ASSUMPTION: all variables have the same number of states r_b
    // Idea: j as a number in base r_b, where the states of the parents are digits
    // $j = r_b^1 \cdot sp_1 + r_b^0 \cdot sp_2 = r_b \cdot sp_1 + sp_2$
    $p_1 \leftarrow \pi[1]$;
    $r_b \leftarrow r_{p_1}$;
    $sp_1 \leftarrow \lfloor j/r \rfloor$;
    $sp_2 \leftarrow j\%r_b$;
    return sp_1, sp_2
function *calcNijk(examples, π, i, j, k, r)*:
    $count \leftarrow 0$;
    for $example \in examples$ **do**
        if $example[i] = k$ **then** // check if variable i is in the k^{th} state
            if $\pi = \emptyset$ **then**
                // empty parent set: only has one state $j = 0$
                if $j = 0$ **then**
                    $count \leftarrow count + 1$;
                end
            else if $size(\pi) = 1$ **then**
                // singleton: just check if the only parent is in the j^{th} state
                $p_1 \leftarrow \pi[1]$;
                if $example[p_1] = j$ **then**
                    $count \leftarrow count + 1$;
                end
            else // parent set has 2 variables
                $sp_1, sp_2 \leftarrow calcJthState(j, \pi, r)$;
                $p_1 \leftarrow \pi[1]$;$p_2 \leftarrow \pi[2]$;
                if $example[p_1] = sp_1$ **and** $example[p_2] = sp_2$ **then**
                    $count \leftarrow count + 1$;
                end
            end
        end
    end
    return *count*;
function *calcSi(i, π, r, α, examples)*:
    $q_i \leftarrow 1$;
    for $p \in \pi$ **do** $q_i \leftarrow q_i \cdot r_p$;
    $sum \leftarrow 0$;
    for $j \leftarrow 1$ **to** q_i **do**
        $\alpha_{i\pi j} \leftarrow \sum_{k=1}^{r_i} \alpha_{i\pi jk}$;
        $N_{ij} \leftarrow \sum_{k=1}^{r_i} calcNijk(examples, \pi, i, j, k, r)$;
        $sum \leftarrow sum + \ln\Gamma(\alpha_{i\pi j}) - \ln\Gamma(\alpha_{i\pi j} + N_{ij})$;
        for $k \leftarrow 1$ **to** r_i **do**
            $N_{ijk} \leftarrow calcNijk(examples, \pi, i, j, k, r)$;
            $sum \leftarrow sum + \ln\Gamma(\alpha_{i\pi jk} + N_{ijk}) - \ln\Gamma(\alpha_{i\pi jk})$;
        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 4: *calcS(n, r, parentSets, α, examples)*

Input: number of bayesian variables n, list *parentSets* of all possible parent sets for each of the n bayesian variables, $s = (s_i(\pi), \pi \in parentSets[i])_{i=1}^{n}$ where $s_i(\pi)$ is the score for the bayesian variable i given the parent set π

Result: $w = (w_i(\pi), \pi \in parentSets[i])_{i=1}^{n}$ where $w_i(\pi)$ is the weight calculated for the bayesian variable i given the parent set π

function *calcWi(i, π, s)*:
  if $\pi = \emptyset$ **then**
    return $s_i(\emptyset)$
  else if $size(\pi) = 1$ **then**
    return $s_i(\pi) - s_i(\emptyset)$;
  else
    $p_1 \leftarrow \pi[1]$; $p_2 \leftarrow \pi[2]$;
    return $s_i(\pi) - s_i(\{p_1\}) - s_i(\{p_2\}) + s_i(\emptyset)$;
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 5: *calcW(n, parentSets, s)*

Input: number of bayesian variables n, list *parentSets* of all possible parent sets for each of the n bayesian variables, $w = (w_i(\pi), \pi \in parentSets[i])_{i=1}^{n}$ where $w_i(\pi)$ is the weight calculated for the bayesian variable i given the parent set π

Result: $\Delta = ((\Delta_{ji})_{i=1}^{n})_{j=1}^{n}$ with $j \neq i$ where Δ_{ji} are intermediate values necessary to calculate the penalties δ

function *calcDeltaji(j, i, w, parentSets, n)*:
  $\Delta'_{ji} \leftarrow -w_i(\{j\})$;
  for $\pi \in parentSets[i]$ **do**
    if $size(\pi) = 2$ **and** $j \in \pi$ **then** // $\pi = \{j, k\}$ `with` $k \neq i$
      $\Delta'_{ji} \leftarrow -min(0, w_i(\pi))$
    end
  end
  return $max(0, \Delta'_{ji})$;
for $j \leftarrow 1$ **to** n **do**
  for $i \leftarrow 1$ **to** n **do**
    if $i \neq j$ **then**
      $\Delta_{ji} \leftarrow calcDeltaji(j, i, w, parentSets, n)$;
    end
  end
end
return Δ;

Algorithm 6: *calcDelta(n, parentSets, w)*

of variables $\{X_i, X_j\}$ it adds the coefficients of the linear and quadratic terms present in $H^{(i)}_{consist}$ multiplied by the $\delta_{consist}$ penalty to the corresponding positions. The complexity is $\mathcal{O}(n^2)$.

2.4 Expected solution calculation for comparison

To test the algorithm and have an appropriate way of evaluating solutions, the need to find the expected minimum solution arose. The problems used for testing are known, so the expected Bayesian network for each problem is known. This can easily be encoded in the edge bits d, but the y and r bits, that are part of the QUBO solution, also need to be set. As a first approach a simple brute force was applied, but due to the need of computing all the $2^{\frac{n(n-1)}{2}+n\mu}$ instances and performing $x^T Q x$ to get the value, with a complexity of $\mathcal{O}(n^2 2^{n^2})$ it quickly became unfeasible even for very small problems. Algorithm 16 more intelligently sets the values for the y and r bits, with a complexity of $\mathcal{O}(n^3)$ for sparse graphs and $\mathcal{O}(n^4)$ for dense graphs. Setting the y bits is quite simple, since it is only necessary to bring $H^{(i)}_{max}$ (1.8) to zero. This is possible when the number of parents for X_i is at most m. For any number of parents smaller than m, the y bits just have to take the value of the difference, so

Input: number of bayesian variables n, $\Delta = ((\Delta_{ji})_{i=1}^{n})_{j=1}^{n}$ with $j \neq i$ where Δ_{ji} are intermediate values necessary to calculate the penalties δ
Result: list δ_{max} of penalties $\delta_{max}^{(i)}$, s.t. $\delta_{max}^{(i)} > \underset{j \neq i}{max} \Delta_{ji}, 1 \leq i \leq n$

```
for i ← 1 to n do
    // look for the max Δ_ji for variable i
    tmpMax ← 0;
    for j ← 1 to n do
        if i ≠ j then
            tmpMax ← max(tmpMax, Δ_ji);
        end
    end
    // +1 to satisfy the constraint on δ_max^(i)
    δ_max^(i) ← tmpMax + 1;
end
return δ_max;
```

Algorithm 7: *calcDeltaMax(n, Δ)*

Input: number of bayesian variables n, $\Delta = ((\Delta_{ji})_{i=1}^{n})_{j=1}^{n}$ with $j \neq i$ where Δ_{ji} are intermediate values necessary to calculate the penalties δ
Result: penalty δ_{trans} that satisfies $\delta_{trans} > \underset{\substack{1 \leq i,j \leq n \\ i \neq j}}{max} \Delta_{ji}$

```
tmpMax ← 0;
for j ← 1 to n do
    for i ← 1 to n do
        if i ≠ j then
            tmpMax ← max(tmpMax, Δ_ji);
        end
    end
end
// +1 to satisfy the constraint on δ_trans
δ_trans ← tmpMax + 1;
return δ_trans;
```

Algorithm 8: *calcDeltaTrans(n, Δ)*

Input: number of bayesian variables n, penalty δ_{trans}
Result: penalty $\delta_{consist}$ that satisfies $\delta_{trans} > (n-2) \cdot \delta_{consist}$

```
δ_consist ← (n − 2) · δ_trans + 1;
return δ_consist;
```

Algorithm 9: *calcDeltaConsist(n, δ_{trans})*

that the sum of d and y will be equal to m, thus preventing any penalty. Setting the r bits is more complex. These bits encode a topological ordering of the variables X_i, but they assume the graph to be a tournament. The topological order for the real graph can be set by simply finding the successor for every node and setting the r bits accordingly. The issue is that the real graph is usually not a tournament, meaning that some or many of these bits cannot be set properly. Defaulting to 0 or 1 for the non-described bits is not a solution, since depending on the graph it could produce a cycle. The solution to this is to start from the real graph indicated by the edge bits d, complete it by adding one edge at a time, while making sure no cycle is added, obtaining a tournament and finally computing the successors for the derived tournament. From these successors all the r bits can be set adding no cycles and no additional penalties. Note that if the original graph indicated by the edge bits d contained a cycle, it will remain with its relative penalty.

2.5 Exhaustive search

For the purpose of testing the correctness of the implementation and also for finding suitable values for the α hyperparameters, the first approach that has been used for finding the minimum is an exhaustive

Input: number of bayesian variables n
Result: list *indexQUBO* containing the tuple representation of the QUBO variables used as indices for the QUBO matrix

```
// calculate the d variables representing arcs between bayesian variables
d ← Vector();
for j ← 1 to n do
    for i ← 1 to n do
        if j ≠ i then
            d.append(('d', j, i));
        end
    end
end
// calculate the y variables necessary for the constraint on the nlumber of parents
y ← Vector();
for i ← 1 to n do
    for l ← 1 to 2 do // fixed m = 2
        y.append(('y', i, l));
    end
end
// calculate the r variables necessary for the constraint on cycle avoidance
r ← Vector();
for j ← 1 to n do
    for i ← j + 1 to n do
        y.append(('r', j, i));
    end
end
indexQUBO ← Vector();
indexQUBO.append(d);
indexQUBO.append(y);
indexQUBO.append(r);
return indexQUBO;
```

Algorithm 10: *calcIndexQUBO(n)*

Input: list *indexQUBO* containing the tuple representation of the QUBO variables used as indices for the QUBO matrix
Result: dicionary *posOfIndex* with tuple representation of the QUBO variables as keys and the corresponding row/column of the QUBO matrix as values

```
posOfIndex ← Dictionary();
for i ← 1 to size(indexQUBO) do
    index ← indexQUBO[i];
    posOfIndex[index] ← i;
end
return posOfIndex;
```

Algorithm 11: *calcPosOfIndex(indexQUBO)*

search on all possible values for the solution vector x (1.1). Initially it consisted of a full brute force procedure with a complexity of $\mathcal{O}(n^2 2^{n^2})$.
To improve the performance, Algorithm 16 was used, so that it is only necessary to calculate all the combinations of the edge bits d while taking only the best setting for the y and r bits. This does not improve the complexity from an asintotic point of view, but it reduces the number of operations from $2^{n(n-1)+\frac{n(n-1)}{2}+n\mu}$ to $\mathcal{O}(n^4) \cdot 2^{n(n-1)}$. Another improvement introduced, is the parallelisation, to exploit all the 4 cores available. The parallelisation further reduces the necessary time to about a fourth, by equally distributing the combinations to evaluate. With these improvements the search passed from being able to solve problems with $n = 3$ in seconds, solve problems with $n = 4$ in 18 minutes and not beying able to solve problems with $n \geq 5$ to solving problems with $n = 3$ and $n = 4$ in seconds, problems with $n = 5$ in 2 minutes and not being able to solve problems with $n \geq 6$. Although the improvement seems marginal, it allowed for testing the code on a lager variety of problems with increasing complexity. This aspect has been essential for finding a robust setting for

Input: QUBO matrix Q of size $size(indexQUBO) \times size(indexQUBO)$, dicionary $posOfIndex$ with tuple representation of the QUBO variables as keys and the corresponding row/column of the QUBO matrix as values, number of bayesian variables n, list $parentSets$ of all possible parent sets for each of the n bayesian variables, $w = (w_i(\pi), \pi \in parentSets[i])_{i=1}^{n}$ where $w_i(\pi)$ is the weight calculated for the bayesian variable i given the parent set π

Result: QUBO matrix Q with values added according to the score hamiltonian H_{score}

```
for i ← 1 to n do
    for π ∈ parentSets[i] do
        if π ≠ ∅ then
            if size(π) = 1 then
                // elements of the diagonal
                j ← π[1];
                index ← ('d', j, i);
                col ← row ← posOfIndex[index];
                Q[row][col] ← Q[row][col] + w_i(π);
            else
                // elements outside the diagonal
                x ← π[1]; y ← π[2];
                indexR ← ('d', x, i);
                indexC ← ('d', y, i);
                row ← posOfIndex[indexR];
                row ← posOfIndex[indexC];
                Q[row][col] ← Q[row][col] + w_i(π);
            end
        end
    end
end
return Q;
```

Algorithm 12: *addHscore(Q, posOfIndex, n, parentSets, w)*

the α hyperparameters.

2.6 Complexity

The construction of the QUBO matrix Q in Algorithm 1 has a complexity of $\mathcal{O}(n^4 + n^3 N r_{max}^3)$. It depends on several factors, namely the number of Bayesian variables n in the original BNSL problem, the size (number of examples) N of the dataset and the maximum number of states r_{max} across all bayesian variables. For a number of Bayesian variables n smaller than the dataset size N, the first term can be ignored. This can be safely done since n has to be small because of the current physical limitations, while N has to be considerably larger than n to provide enough information to learn from. The complexity then becomes $\mathcal{O}(n^3 N r_{max}^3)$. The cause is to be found in the calculation of the local scores s (Algorithm 4) that are of central importance for the problem encoding.
With the maximum number of parents m set to 2, the construction requires $\mathcal{O}(n^2)$ QUBO variables, which is a limiting factor on the BNSL problem size. For this reason, the main factors that can have a negative impact on the performance are the dataset size and the number of states of the variables, since the size of the QUBO representation is independent of these values. The latter in particular has to be taken in consideration in the case of the discretisation of Bayesian variables with continuous states. The accuracy of the representation for this kind of problems comes at the cost of the time necessary to construct the QUBO matrix.

2.7 D-Wave Ocean suite

2.7.1 Problem format

Ocean is the D-Wave Systems suite that allows interacting with the quantum annealers. Ocean provides a class called `BinaryQuadraticModel` (BQM) that can represent problems expressed both as Ising model and as QUBO. This class takes as input a *linear* and a *quadratic* dictionary with the same semantics as in section 1.1. The linear component has a single variable name as key and the

Input: QUBO matrix Q of size $size(indexQUBO) \times size(indexQUBO)$, dicionary $posOfIndex$ with tuple representation of the QUBO variables as keys and the corresponding row/column of the QUBO matrix as values, number of bayesian variables n, list δ_{max} of penalties $\delta_{max}^{(i)}$

Result: QUBO matrix Q with values added according to the max hamiltonian H_{max}

for $i \leftarrow 1$ **to** n **do**
  // $H_{max}^{(i)} = \delta_{max}^{(i)} \cdot (m - d_i - y_i)^2$
  $m \leftarrow 2$; // max number of parents m is fixed
  $toSquare \leftarrow Vector()$;
  $c \leftarrow Vector()$; // coefficients for the variables in $toSquare$
  // add all d_{ji}
  for $j \leftarrow 1$ **to** n **do**
    if $j \neq i$ **then**
      $toSquare.append(('d', j, i))$;
      $c.append(-1)$;
    end
  end
  // add all y_{il} for $m = 2$
  $toSquare.append(('y', i, 1))$;
  $c.append(-1)$;
  $toSquare.append(('y', i, 2))$;
  $c.append(-2)$;
  /* elements of the diagonal: */
  /* * square of elements (m excluded since it is a constant) */
  /* * double products with */
  for $index \in toSquare$ **do**
    $col \leftarrow row \leftarrow posOfIndex[index]$;
    // add penalty multiplied by the square of the coefficient
    $Q[row][col] \leftarrow Q[row][col] + \delta_{max}^{(i)} \cdot c[i]^2$;
    // add penalty multiplied by m and the coefficient of the index
    $Q[row][col] \leftarrow Q[row][col] + \delta_{max}^{(i)} \cdot (2 \cdot m \cdot c[i])$;
  end
  // outside the diagonal
  for $j \leftarrow 1$ **to** $size(toSquare)$ **do**
    $indexR \leftarrow toSquare[j]$;
    $row \leftarrow posOfIndex[indexR]$;
    for $k \leftarrow j + 1$ **to** $size(toSquare)$ **do**
      $indexC \leftarrow toSquare[k]$;
      $col \leftarrow posOfIndex[indexC]$;
      // add penalty multiplied by the double product of the coefficients
      $Q[row][col] \leftarrow Q[row][col] + \delta_{max}^{(i)} \cdot (2 \cdot c[j] \cdot c[k])$;
    end
  end
end
return Q;

Algorithm 13: *addHmax(Q, posOfIndex, n, δ_{max})*

corresponding linear bias as value. The quadratic component has a tuple containing two variables as key and the corresponding quadratic bias as value. BQM accepts linear and quadratic components in both the symmetric and the upper triangular form: considering two variables x_i, x_j, the bias for (i, j) is set to the sum of the biases assigned to (i, j) and (j, i), always resulting in an upper triangular form. Since a QUBO formulation of the problem is already stored in Q, no particular computation is required: the linear dictionary contains the values from the main diagonal of Q, while the quadratic dictionary contains all the values of Q above the main diagonal. The names of the variables that are the keys in the linear and quadratic dictionaries can be simply obtained from $indexQUBO$.

2.7.2 Problem submission

The submission of a problem to a remote machine or to a local solver is done via a `Sampler` class. The connection to a quantum computer is done by creating an instance of the `DWaveSampler` class, optionally providing constraints for choosing the remote machine. For all the tests, the parameter for

Input: QUBO matrix Q of size $size(indexQUBO) \times size(indexQUBO)$, dicionary $posOfIndex$ with tuple representation of the QUBO variables as keys and the corresponding row/column of the QUBO matrix as values, number of bayesian variables n, penalty δ_{trans}
Result: QUBO matrix Q with values added according to the transitive hamiltonian H_{trans}

for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow i+1$ **to** n **do**
    for $k \leftarrow j+1$ **to** n **do**
      // $H^{(ijk)}_{trans} = \delta_{trans} \cdot (r_{ik} + r_{ij} \cdot r_{jk} - r_{ij} \cdot r_{ik} - r_{ik} \cdot r_{jk})$
      $r_{ij} \leftarrow$ ('r', i, j);
      $r_{ik} \leftarrow$ ('r', i, k);
      $r_{jk} \leftarrow$ ('r', j, k);
      // coefficient for r_{ik}
      $col \leftarrow row \leftarrow posOfIndex[r_{ik}]$;
      $Q[row][col] \leftarrow Q[row][col] + \delta_{trans}$;
      // coefficient for $r_{ij} \cdot r_{jk}$
      $row \leftarrow posOfIndex[r_{ij}]$;
      $col \leftarrow posOfIndex[r_{jk}]$;
      $Q[row][col] \leftarrow Q[row][col] + \delta_{trans}$;
      // coefficient for $r_{ij} \cdot r_{ik}$
      $row \leftarrow posOfIndex[r_{ij}]$;
      $col \leftarrow posOfIndex[r_{ik}]$;
      $Q[row][col] \leftarrow Q[row][col] - \delta_{trans}$;
      // coefficient for $r_{ik} \cdot r_{jk}$
      $row \leftarrow posOfIndex[r_{ik}]$;
      $col \leftarrow posOfIndex[r_{jk}]$;
      $Q[row][col] \leftarrow Q[row][col] - \delta_{trans}$;
    end
  end
end
return Q;

Algorithm 14: *addHtrans(Q, posOfIndex, n, δ_{trans})*

Input: QUBO matrix Q of size $size(indexQUBO) \times size(indexQUBO)$, dicionary $posOfIndex$ with tuple representation of the QUBO variables as keys and the corresponding row/column of the QUBO matrix as values, number of bayesian variables n, penalty $\delta_{consist}$
Result: QUBO matrix Q with values added according to the consistence hamiltonian $H_{consist}$

for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow i+1$ **to** n **do**
    // $H^{(ij)}_{consist} = \delta_{consist} \cdot (d_{ji} \cdot r_{ij} + d_{ij} - d_{ij} \cdot r_{ij})$
    $d_{ij} \leftarrow$ ('d', i, j);
    $d_{ji} \leftarrow$ ('d', j, i);
    $r_{ij} \leftarrow$ ('r', i, j);
    // coefficient for $d_{ji} \cdot r_{ij}$
    $row \leftarrow posOfIndex[d_{ji}]$;
    $col \leftarrow posOfIndex[r_{ij}]$;
    $Q[row][col] \leftarrow Q[row][col] + \delta_{consist}$;
    // coefficient for d_{ij}
    $col \leftarrow row \leftarrow posOfIndex[d_{ij}]$;
    $Q[row][col] \leftarrow Q[row][col] + \delta_{consist}$;
    // coefficient for $d_{ij} \cdot r_{ij}$
    $row \leftarrow posOfIndex[d_{ij}]$;
    $col \leftarrow posOfIndex[r_{ij}]$;
    $Q[row][col] \leftarrow Q[row][col] - \delta_{consist}$;
  end
end
return Q;

Algorithm 15: *addHconsist(Q, posOfIndex, n, $\delta_{consist}$)*

requiring the Pegasus topology was set, this way only the D-Wave Advantage computer was used. To submit the problem, the `sample()` method of the sampler is called, with the BQM encoding of the

Input: x solution vector with only the edge bits set, list *indexQUBO* containing the tuple representation of the QUBO variables used as indices for the QUBO matrix, dicionary *posOfIndex* with tuple representation of the QUBO variables as keys and the corresponding row/column of the QUBO matrix as values, number of bayesian variables n

Result: x solution vector with the best setting for the y and r bits (no penalties if possible, otherwise minimum penalty)

function *completeGraph(graph,n,succ)*:
  for $u \leftarrow 1$ **to** n **do**
    for $v \leftarrow u+1$ **to** n **do**
      // if there is no edge between u and v
      if not $(u \in graph[v])$ **and not** $(v \in graph[u])$ **then**
        // if u is not a successor of v add the edge (u, v)
        // else v comes before u, so to avoid cycles add the edge (v, u)
        if not $u \in succ[v]$ **then**
          $graph[u].append(v)$;
        else
          $graph[v].append(u)$;
        end
      end
    end
  end
  return s;
$nedges \leftarrow n * (n-1)$;
$countEdgesToVar \leftarrow Vector(n)$;
// Calculate the in-degree for each variable
for $e \leftarrow 1$ **to** $nedges$ **do**
  if $x[e] = 1$ **then**
    $dindex \leftarrow indexQUBO[e]$;
    // $dindex$ has the form $('name', i, j)$
    $j \leftarrow dindex[3]$;
    $countEdgesToVar[j] \leftarrow countEdgesToVar[j] + 1$;
  end
end
// Set the y bits so that $H^{(i)}_{max}$ is zero when X_i has m or less parents
for $i \leftarrow 1$ **to** n **do**
  $y_{i1} \leftarrow ('y', i, 1)$; $y_{i2} \leftarrow ('y', i, 2)$;
  $posy_{i1} \leftarrow posOfIndex[y_{i1}]$; $posy_{i2} \leftarrow posOfIndex[y_{i2}]$;
  $x[posy_{i1}] \leftarrow 0$; $x[posy_{i2}] \leftarrow 0$;
  if $countEdgesToVar[i] = 0$ **then**
    $x[posy_{i2}] \leftarrow 1$;
  else if $countEdgesToVar[i] = 1$ **then**
    $x[posy_{i1}] \leftarrow 1$;
  end
end
/* Set the r bits so that they are consistent with the edge bits and no cycles are formed
   (assuming the edge bits do not encode cycles already) */
$graph \leftarrow G(x, n)$; // Generate an adjacency vector from the edge bits
$succ \leftarrow succ(graph, n)$; // Generate a vector containing the successors for each variable
// Complete the graph to be an acyclic tournament (possible if there are no cycles already)
$completeGraph(graph, n, succ)$;
$numR \leftarrow \frac{n(n-1)}{2}$;
$posFirstR \leftarrow posOfIndex[('r', 0, 1)]$;
for $r \leftarrow posFirstR$ **to** $posFirstR + numR$ **do**
  $rindex \leftarrow indexQUBO[r]$;
  $i \leftarrow rindex[2]$; $j \leftarrow rindex[3]$;
  if $i \in succ[j]$ **then**
    $x[r] \leftarrow 0$;
  else
    $x[r] \leftarrow 1$;
  end
end
return x

Algorithm 16: *setBestParams(x, indexQUBO, posOfIndex, n)*

problem as parameter, along with the number of reads (samplings to perform) and the annealing time for every read. This call returns the results for all reads, from which the best result is extracted.

2.7.3 Embedding

Due to the sparseness of the annealer graph and the highly connected nature of the QUBO formulation of the BNSL problem, a direct representation is not possible. The Ocean library provides a solution to this problem via *minor embedding*. The idea is to use multiple qubits chained together that act as one, increasing the connectivity of the graph of the annealer at the price of reducing the logical amount of qubits, consequently reducing the size of the embeddable problems.
The usage of the minor embedding is performed by simply instantiating a sampler using the `EmbeddingComposite` class, passing as parameter an instance of a sampler like `DWaveSampler`. This way the sampler constructed by `EmbeddingComposite` automatically finds a minor embedding for the submitted BQM problem.

3 Tests and results

In this section, the problems and the generation of their datasets is presented, along with extensive tests to check the correctness of formulation and evaluate the performance of the quantum computer. The results of these tests are then thoroughly discussed.

3.1 Problems used

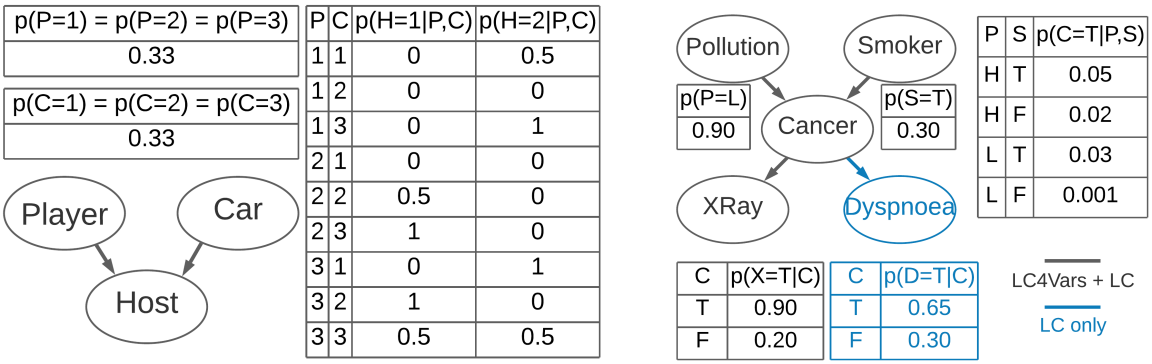

Figure 3.1: Monty Hall Problem (left) and Lung Cancer (right)

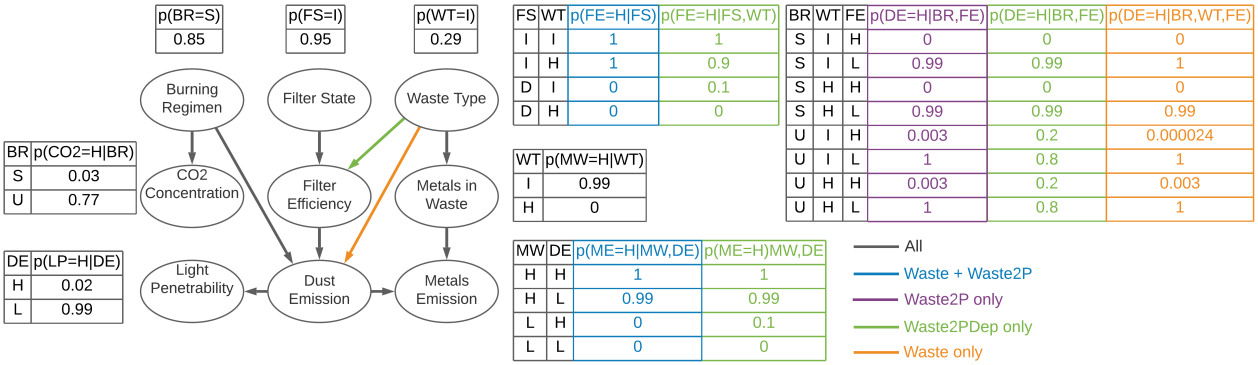

Figure 3.2: Waste

3.1.1 Monty Hall Problem

The Monty Hall Problem (MHP) was chosen as first as first problem, because of its simplicity and its popularity. Each variable has three states $\{1, 2, 3\}$, symbolising the number of the door. The Beyesian network of the problem (Figure 3.1) is composed of $n = 3$ variables, which is the minimum amount for the QUBO encoding. The reason behind this is that, with $n = 2$, the penalty $\delta_{consist}$ would become indipendent of δ_{trans} because of (2.3). With only $n = 1$ variable there would be no Bayesian network to learn.

3.1.2 Lung Cancer

To increase the number of variables, the Lung Cancer network was chosen. Each variable has two states that can either be $\{H, L\}$ high or low, or $\{T, F\}$ true or false. Figure 3.1 shows both the original network (LC) composed of $n = 5$ variables and a variation (LC4Vars) with $n = 4$ variables, obtained

by removing the "Dyspnoea" variable. Doing so makes it possible to gradually scale the problem size, which is useful for testing.

3.1.3 Waste

The Waste problem [5] was chosen for various reasons. Firstly, to reach the size limits of the embeddable problems. Secondly, to try to represent a problem containing variables with continuous states. Lastly, to test what happens when the structure of the network contains more parents than m allows. Six of the original Waste variables have continuous states. Since the QUBO encoding only works with discrete variables, a discretisation has to be applied. All the continuous variables are transformed into discrete variables of two states, with the following procedure. First, the lowest and highest values for the continuous variable are calculated by testing all possible settings of the variables it depends on. This is possible since the variables and the states are few. The middle point between the lowest and highest value is kept as a threshold that is used to discriminate between the two states. Then, for each state combination of the parent variables states, the mean and variance for the continuous variable are taken. Lastly, the probability of the Gaussian with this mean and variance having a value higher than the threshold is set to be the probability for the H state.
The resulting problem is kept with the name Waste. The discretisation previously described caused the loss of the edge between "Waste Type" and "Filter Efficiency", but the rest of the topology remained the same (Figure 3.2). Two variations of Waste are also introduced. Waste2P removes the edge between "Waste Type" and "Dust Emission" to respect the $m = 2$ maximum number of parents constraint. Waste2PDep is a variation of Waste2P that reintroduces the edge between "Waste Type" and "Filter Efficiency" that was lost in the discretization, by altering some probabilities.

3.2 Datasets generation

For each of the problems several datasets have been constructed, varying both in size and method of generation. Three dataset sizes N have been used and are 10^4 (10K), 10^5 (100K), 10^6 (1M). Two methods of generation have been used. The first consists in generating N examples making use of the `uniform()` function from Python's `random` library for obtaining a random outcome using the probabilities shown in Figures 3.1 and 3.2. The second method aims at generating a dataset with zero variance, with combinations of states appearing exactly the expected amount of times. These datasets have "Exp" in the name for distinction. To generate the expected datasets, the probability p of every combination of states of the variables is calculated, then $\lfloor N * p \rfloor$ examples with those states are saved in the dataset. For some probabilities p it is not possible to generate an integer number of examples and consequently the variance is not exactly zero, but given the size of the datasets and the very small imprecision, it is still a very good approximation.
For each problem, datasets have been generated combining all sizes and both methods. In addition, multiple datasets have been generated using the same size and the variability method for each problem, to provide a certain variation in the data.

3.3 Correctness of formulation

For the QUBO encoding to accurately represent the original BNSL problem, the α hyperparameters need to be chosen appropriately. With a suitable value for alpha, the value of the expected solution (Section 2.4) should be the best value among all the possible solutions. To check this, the expected solution needs to be computed starting from the edges of the BNSL graph and the minimum valued solution needs to be found with the exhaustive search (Section 2.5).
Since the objective is to learn the structure exclusively from a dataset, the values for α should be uninformative, i.e. they should not encode any information on the target Bayesian network. For this reason the first values chosen for α are $N/(r_i \cdot q_i)$ and 1 as proposed by Heckerman et al. [2]. The first has the effect of having the α hyperparameters related to a variable X_i and its parent set Π_i sum to the dataset size N. This tends to flatten the probabilities, since it adds in (1.6) the number of examples that would be found if all the variable and parent set states were uniformly distributed.
Setting all α to 1 does not yield the desired behaviour and setting them to $N/(r_i \cdot q_i)$ shows some partially good results for the smallest problem, but does not work as soon as the number of variables

increases.
Given the results, new values for α have been tested, with the idea of influencing the data as little as possible. Both $1/r_i$ and $1/(r_i \cdot q_i)$ have the desired behaviour and the latter has been choosen as the permanent setting. This setting has the effect of having the α hyperparameters related to a variable X_i and its parent set Π_i sum to 1.
The table 3.1 shows the results discussed here. The tests have been performed on datasets of different sizes, generated both with variability and the expected values.

Problem	$\alpha = N/(r_i \cdot q_i)$	$\alpha = 1$	$\alpha = 1/(r_i \cdot q_i)$	$\alpha = 1/r_i$
MHP	0.8	0.0	**1.0**	1.0
LC4Vars	0.0	0.0	**1.0**	1.0
LC	0.0	0.0	**1.0**	1.0

Table 3.1: Ratio between times in which the best solution is the expected solution and the number of tests, with different settings for α on different problems

3.4 Comparison

Several factors that influence the performance have been analyzed, regarding both the QUBO construction time and the quality of the solutions found with different search methods.

3.4.1 Dataset sizes

For all the problems, the impact that the dataset size has on the QUBO formulation time is analyzed. In accordance with the complexity $\mathcal{O}(n^3 N R_{max}^3)$ discussed in Section 2.6, the time for the computation increases linearly with the dataset size as shown in Table 3.2. Especially when computing the QUBO matrix for large problems, larger datasets sizes become increasingly prohibitive. Because of this and the fact that having a dataset size larger than $N = 10^4$ brings no improvement (Table 3.3), it is best to keep the dataset size limited.

Problem	$N = 10^4$	$N = 10^5$	$N = 10^6$
MHP	0.36	3.45	34.75
LC4Vars	0.53	4.93	48.78
LC	1.19	11.24	115.15
Waste	9.68	—	—
Waste2P	8.48	90.28	—
Waste2PDep	8.92	91.14	875.57

Table 3.2: Average time in seconds necessary to calculate the QUBO matrix Q, varying problem and dataset size

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

Table 3.3: Average solution value for Waste2PDep on expected datasets of different sizes

3.4.2 Number of reads and read time

When submitting a problem to D-Wave, additional parameters can be set. Two important parameters are the number of reads to be performed for the problem and the annealing time for each read. Table 3.4 reports an extensive test for these parameters. The maximum allowed number of reads is 10^4 and the maximum annealing time for a read is 2000μs, but there is another internal limit that prevents an annealing time larger than 999μs with 10^3 reads and prevents an annealing time larger than 99μs with 10^4 reads. From the results is clear that increasing either the number of reads or the annealing time provides better results. Increasing the reads and keeping the annealing time well under the joint limit already shows better results than less reads with the annealing time maximised. Using as many

reads as possible and only then maximising the annealing time to reach the joint limit, provides the best results.

	Annealing time per sample (μs)					
#reads	1	10	20	50	99	999
10^3	0.00	0.00	—	—	—	0.10
10^4	—	0.20	0.22	0.40	0.50	—

Table 3.4: Ratio between number of times the minimum is found and number of experiments on the LCExp10K dataset, with different settings for the number of reads and the read time

3.4.3 Performances

All methods for finding the minimum have been put to comparison for all the problems and the results are reported in Table 3.5. In addition to the exhaustive search (ES) and quantum annealing (QA), Simulated annealing (SA) has also been used. This method is implemented by simply using the `SimulatedAnnealingSampler` class provided by the Ocean D-Wave library. All these tests have been computed on expected datasets of size $N = 10^4$ and for QA an annealing time per read of 99μs has been used, providing the best settings.

ES outperforms both SA and QA for small problems of size $n = 3$ and $n = 4$, but quickly falls behind for $n = 5$ and it becomes unfeasible for $n \geq 6$ due to its $\mathcal{O}(n^2 2^{n^2})$ complexity discussed in Section 2.5. For the smallest problems with $n = 3$ and $n = 4$ variables, QA always manages to find the optimum solution. Moreover it finds the minimum several times, which suggests that fewer reads would be enough to obtain the same accuracy. From $n = 5$ and on, SA manages to obtain better results than QA, with only little increase in the required time. For $n = 5$ variables, QA still manages to find the mimimum half of the times, with the minimum occasionally appearing more than once. For the $n = 9$ problems neither SA nor QA ever finds the minimum, but on average the results are still good quality solutions.

The Waste problem is an exception. The best solutions found have a lower score than the expected solution. This always happens for SA and sometimes happens for QA. The reason behind it is that the expected solution contains a node with three parents. This penalises the solution and makes it possible to have better solutions that respect the $m = 2$ parent constraint.

		Exhaustive search		Simulated annealing			Quantum annealing			
n	Problem	Success rate	Search time(s)	Success rate	Average result	Anneal time(s)	Success rate	Average result	Anneal time(s)	Occour exp
3	MHP	1.00	0.51	1.00	1.0000	3.40	1.00	1.0000	1.76	215.90
4	LC4Vars	1.00	0.89	1.00	1.0000	5.52	1.00	1.0000	1.77	11.40
5	LC	1.00	134.81	1.00	1.0000	8.99	0.50	0.9987	1.80	0.60
9	Waste	—	—	0.00	1.0145	13.07	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	12.24	0.00	0.9763	2.14	0.00

Table 3.5: Comparison between exhaustive search, simulated annealing and quantum annealing performances, for all problems on expected datasets of size $N = 10^4$ and annealing time per sample of 99 μs

Table 3.6 shows the same QA tests with annealing time per sample of 99μs, compared with the same tests performed with annealing time per sample of 1μs. The higher annealing time provides higher success rate, and a higher number of occourences of the minimum solution for the $n = 4$ (LC4Vars) and $n = 5$ (LC) problems. For all the problems the increased annealing time per sample provides on average better results, with little additional time required.

SA does not limit the amount of reads. For this reason further tests have been made on Waste2PDep problem with the expected dataset, using 10^5 and 10^6 reads. The time necessary for the computation linearly increased, but the solutions showed no substantial improvement.

		Annealing time per sample 1μs				Annealing time per sample 99μs			
n	Problem	Success rate	Average result	Anneal time(s)	Occour exp	Success rate	Average result	Anneal time(s)	Occour exp
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	0.22	0.9979	1.01	0.24	0.50	0.9987	1.80	0.60
9	Waste	0.00	0.9619	1.16	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	0.9630	1.35	0.00	0.00	0.9763	2.14	0.00

Table 3.6: Comparison of quantum annealing performances with different annealing times per sample, for all the problems on expected datasets of size $N = 10^4$

3.5 Discussion

The state of the art quantum annealer D-Wave Advantage loses to a desktop computer, where the same problem, solved with simulated annealing, provides better results, with not much higher time requirements. There are multiple factors that lead to the poor results. Firstly, with respect to simulated annealing, the quantum computer is at a disadvantage: the simulated annealing works natively with the encoded QUBO problem, while the quantum annealer needs to embed it in its physical structure, further increasing the actual problem size and thus its difficulty. The embedding is necessary because of the highly connected nature of the QUBO formulation and the sparsity of the quantum annealer's physical graph. The need for the embedding also restricts the BNSL problem size even further. In fact, the size of the actual problem is increased with the embedding of the QUBO formulation, but the QUBO formulation already requires $\mathcal{O}(n^2)$ variables for n BNSL variables. For these reasons, for quantum annealing to be a valuable alternative, the best improvement that could be made is the increase of the connectivity of the physical graph. This is most likely harder than increasing the number of qubits, but is the only way of reducing the final problem size. Nevertheless over the last years D-Wave Systems has increased the qubit connectivity by a factor of 2.5, passing from the Chimera topology having 6 connections per qubit to the Pegasus architecture used by Advantage having 15 connections per qubit, so a further increase in the future does not seem unlikely.
The tests show an increase in both quantity and quality of the solutions when more samplings are performed and when the annealing time for each sample is increased. These parameters have a limit imposed by D-Wave Systems that only seems to be there for preventing the use of too much time on a single problem. If the limit could be removed or increased, better performances could be obtained at the cost of more machine time.

The proposed QUBO encoding, besides requiring $\mathcal{O}(n^2)$ variables, can be computationally demanding due to the complexity of $\mathcal{O}(n^3 N r_{max}^3)$ for the construction of the QUBO matrix. This makes it particularly prone to large calculation time when the number of Bayesian variables n or the number of their states is increased and it is also sensitive to the datset size N. In general the QUBO construction can easily become the most time consuming task if simulated annealing or quantum annealing are employed to search for the minimum. For the calculation of the complexities, the constraint $m = 2$ on the maximum number of parents was used, because with a larger number of parents the formulation would require even more QUBO variables ($\mathcal{O}(n^3)$ for $m = 3$) which would consequently increase the computation complexity. Limiting the number of parents to $m = 2$ has the result of having good QUBO encodings for sparse BNSL problems, while the degradation increases as the problems get denser. That said, for problems that can be well represented with this constraint, the proposed QUBO formulation for BNSL works, always having the best solution as the minimum and the invalid solutions penalised.

4 Conclusions

The original formulation proposed by O'Gorman et al. needs some practical adjustments to work. Firstly, the calculation of the local scores s necessitates of algebraic manipulations to avoid the direct calculation of the gamma function that would quickly yield out of range results, also due to the multiplication of several of its outputs. Then, suitable hyperparameters α need to be found for the encoding to function in the desired way. A positive aspect to notice is that, once a setting for the hyperparameters is found, it seems to work for any problem.

A framework for testing also needed to be put in place. Problems for this purpose have been found and replicated, and datasets have been generated from them. An initial testing of the correctness of the algorithm has also been done, using an exhaustive search strategy. With this initial setting, proper α values have been found respecting the requirement for the QUBO encoding to have the expected behaviour for all the problems it has been tested with. Finally, to make proper tests, the problems are submitted to the quantum annealer, making use of the D-Wave Ocean library. From these tests it has been observed that increasing both the number of reads that the annealer performs and the annealing time for each read increases the amount of occourences of the minimum for the problems in which it is found and the quality of the non-optimal solutions for the others. This increase, however, comes at the cost of more machine time spent on each task.

The overall complexity of the QUBO formulation is $\mathcal{O}(n^3 N r_{max}^3)$. This penalises larger problems, larger datasets and problems with variables having a larger amount of states. Since only problems having variables with discrete states can be represented, continuous states problems suffer further penalisation. This is because, to have an accurate discrete representation, continuous problems need many states for each variable, which means that there has to be a trade off between the accuracy of the representation and the efficiency of the QUBO computation.

Due to the maximum parent $m = 2$ constraint, only problems in which every variable has at most two parents can accurately be represented in this QUBO form. This contstraint is necessary, because increasing the number of allowed parents would immediatly raise the $\mathcal{O}(n^2)$ representation complexity and consequently the formulation complexity, significantly reducing the already limited range of tractable problems.

While the method presented for the structure learning of Bayesian networks can definetely work in theory, it does not so in practice. The current state of the art of the quantum technology is the main reason. Despite the great improvements in the last decade, it is still not able to compete with classical methods, at least with regards to this problem. Despite this, there could be room for improvement acting on the method and using the obtained information constructively. A first approach could be combining several solutions that the annealer provides to find an averaged solution. This is possible because the quantum annealer, although rarely finds the best solution for large problems, always finds good solutions. Another approach could be reducing the embedding size of the problem in the physical graph. To reduce the size, a more thorough and more expensive embedding algorithm must be used. This additional cost could be dampened by the fact that this specific QUBO formulation has a fixed structure for all BNSL problems of a certain size. This means that once the embedding for BNSL of size n is computed, it is possible to use this same embedding for all future BNSL problems of size n. To instead try to minimise the necessary time to construct the QUBO matrix, smaller datasets could be tested to find the minimum size that allows the learning of the structure. This makes sense given that no substantial increase or decrease in the encoding's perfomance has been found with the sizes used in the tests.

Bibliography

[1] O'Gorman B., Babbush R., Perdomo-Ortiz A., Aspuru-Guzik A., and Smelyanskiy V. Bayesian network structure learning using quantum annealing. *The European Physical Journal Special Topics*, 224(1):163–188, 2015.

[2] Heckerman D., Geiger D., and Chickering D. M. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243, 1995.

[3] Glover F., Kochenberger G., and Du Y. Quantum bridge analytics I: a tutorial on formulating and using qubo models. *4OR-Q J Oper Res*, 17:355–371, 2019.

[4] Aaronson S. BQP and the polynomial hierarchy. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 141–150, 2010.

[5] Bayes Server: bayesian network examples. https://www.bayesserver.com/examples/. Last access 22/06/2021.

[6] D-wave Systems: How D-Wave Systems work. https://www.dwavesys.com/quantum-computing/. Last access 22/06/2021.

Attachment A Implementation code

A.1 QUBOValues.py

Python library responsible for the calculation of the intermediate values necessary for the calculation of the QUBO matrix.

```
import math

def getValues(filename, alpha='1/(ri*qi)'):
  examples, n, states = getExamples(filename)
  #calculation of parentSets
  parentSets = calcParentSets(n)
  #calculation of w
  w = calcW(n,parentSets,states,examples,alpha=alpha)
  #calculation of delta
  delta = calcDelta(n,parentSets,w)
  #calculation of deltaMax
  deltaMax = calcDeltaMax(n,delta)
  #calculation of deltaTrans
  deltaTrans = calcDeltaTrans(n,delta)
  #calculation of deltaConsist
  deltaConsist = calcDeltaConsist(n,deltaTrans)
  return n, parentSets, w, deltaMax, deltaTrans, deltaConsist

def getExamples(filename):
  with open(filename,'r') as f:
    lines = f.read().splitlines()
    #get info from first line
    info = lines.pop(0).split(' ')
    n = int(info[0])
    #number of states for each variable
    states = [ int(si) for si in info[1:n+1]]
    #get examples
    examples = []
    for l in lines:
      ls = l.split(' ')
      examples += [[ int(x) for x in ls ]]
  return examples, n, states

#subsets with max size m=2
def calcSubSets(s):
  sSet = [()]
  #all individuals
  sSet += [(i,) for i in s]
  #all unordered pairs
```

```
  sSet += [(i,j) for i in s for j in s if i < j]
  return sSet

def calcParentSets(n):
  parentSets = []
  for i in range(n):
    maxSet = tuple( x for x in range(0,n) if x != i )
    parentSets.append(calcSubSets(maxSet))
  return parentSets

def calcAlpha(n, states, parentSets, N, alpha='1/(ri*qi)'):
  if alpha == '1/(ri*qi)':
    def ca(N,ri,qi): return 1/(ri*qi)
  elif alpha == '1':
    def ca(N,ri,qi): return 1
  elif alpha == 'N/(ri*qi)':
    def ca(N,ri,qi): return N/(ri*qi)
  elif alpha == '1/ri':
    def ca(N,ri,qi): return 1/ri
  alpha = []
  for i in range(n):
    alpha.append({})
    #for each valid parent set
    for parentSet in parentSets[i]:
      #generate alpha
      alpha[i][parentSet] = []
      qi = calcQi(parentSet,states)
      for j in range(qi):
        alpha[i][parentSet].append([])
        for k in range(states[i]):
          alpha[i][parentSet][j].append([])
          #initialize alpha according to the choice
          alpha[i][parentSet][j][k] = ca(N,states[i],qi)
  return alpha

def calcAlphaijSum(alpha, parentSet, i, j, states):
  sum = 0
  for alphaijk in alpha[i][parentSet][j]:
    sum += alphaijk
  return sum

def calcJthState(j, parentSet, states):
  #ASSUMPTION: all the variables have the same number of states,
  #if this is false, some combinations will be ignored
  p0 = parentSet[0]
  #j = (states[p0]^1)*sp0 + (states[p1]^0)*sp1
  #j = states[p0]*sp0 + sp1
  sp0 = j//states[p0]
  sp1 = j%states[p0]
  return sp0, sp1

def calcNijk(examples, parentSet, i, j, k, states):
  count = 0
```

```
  for example in examples:
    #variable i is in the k-th state
    if example[i] == k:
      #parent set is in the j-th state
      if parentSet == ():
        #empty parent set -> only has one state j=0
        if j == 0:
          count = count + 1
      elif len(parentSet) == 1:
        #one variable -> is that variable in its j-th state?
        if example[parentSet[0]] == j:
          count = count + 1
      else:
        #parent set has 2 variables
        sp0,sp1 = calcJthState(j,parentSet,states)
        p0 = parentSet[0]; p1 = parentSet[1]
        if example[p0] == sp0 and example[p1] == sp1:
          count = count + 1

  return count

def calcNijSum(examples, parentSet, i, j, states):
  sum = 0
  for k in range(states[i]):
    sum += calcNijk(examples,parentSet,i,j,k,states)
  return sum

def calcQi(parentSet,states):
  qi = 1
  for j in parentSet:
    qi *= states[j]
  return qi

def calcSi(i, parentSet, states, alpha, examples):
  qi = calcQi(parentSet,states)
  sum = 0
  for j in range(qi):
    alphaij = calcAlphaijSum(alpha,parentSet,i,j,states)
    Nij = calcNijSum(examples,parentSet,i,j,states)
    sum += math.lgamma(alphaij) - math.lgamma(alphaij + Nij)
    for k in range(states[i]):
      Nijk = calcNijk(examples,parentSet,i,j,k,states)
      sum += math.lgamma(alpha[i][parentSet][j][k] + Nijk) - math.lgamma(alpha[i
  return -sum

def calcWi(i, parentSet, s):
  if parentSet == ():
    return s[i][()]
  elif len(parentSet) == 1:
    return s[i][parentSet] - s[i][()]
  elif len(parentSet) == 2:
    p0 = parentSet[0]; p1 = parentSet[1]
    return s[i][parentSet] - s[i][(p0,)] - s[i][(p1,)] + s[i][()]
```

```
def calcS(n, states, parentSets, alpha, examples):
  s = []
  for i in range(n):
    s.append({})
    #for each valid parent set
    for parentSet in parentSets[i]:
      #calculate si
      s[i][parentSet] = calcSi(i,parentSet,states,alpha,examples)
  return s

def calcWFromS(n, parentSets, s):
  w = []
  for i in range(n):
    w.append({})
    for parentSet in parentSets[i]:
      #calculate wi
      w[i][parentSet] = calcWi(i,parentSet,s)
  return w

def calcW(n, parentSets, states, examples, alpha='1/(ri*qi)'):
  #calculation of alpha
  alpha = calcAlpha(n,states,parentSets,len(examples),alpha=alpha)
  #calculation of s
  s = calcS(n,states,parentSets,alpha,examples)
  print()
  #calculation of w
  w = calcWFromS(n,parentSets,s)
  return w

def calcDeltaji(j, i, w, parentSets, n):
  deltaPrimeji = -w[i][(j,)]
  #for all parentSets for i, {j,k}
  for parentSet in parentSets[i]:
    if len(parentSet) == 2:
      if j in parentSet:
        deltaPrimeji -= min(0,w[i][parentSet])
  return max(0,deltaPrimeji)

def getDelta(j, i, delta):
  posI = i if i < j else i-1
  return delta[j][posI]

def calcDelta(n, parentSets, w):
  delta = []
  for j in range(n):
    delta.append([])
    for i in range(n):
      if i != j:
        delta[j].append(calcDeltaji(j,i,w,parentSets,n))
  return delta

def calcDeltaMax(n, delta):
```

```
  deltaMax = []
  for i in range(n):
    maxDelta = 0
    for j in range(n):
      if i != j:
        maxDelta = max(maxDelta,getDelta(j,i,delta))
    #+1 to satisfy the > constraint
    deltaMax.append(maxDelta + 1)
  return deltaMax

def calcDeltaTrans(n, delta):
  #calculate max of delta
  maxDelta = 0
  for j in range(n):
    for i in range(n):
      if i != j:
        maxDelta = max(maxDelta,getDelta(j,i,delta))
  #calculate deltaTrans -> they are all the same
  #+1 to satisfy the > constraint
  deltaTrans = maxDelta + 1
  return deltaTrans

def calcDeltaConsist(n, deltaTrans):
  #need to calculate max between deltaTrans(i,j,k), but there is only one deltaT
  #+1 to satisfy the > constraint
  deltaConsist = (n-2)*deltaTrans + 1
  return deltaConsist
```

A.2 QUBOMatrix.py

Python library responsible for the construction of the QUBO matrix.

```
from QUBOValues import getValues
def calcQUBOMatrix(path,alpha='1/(ri*qi)'):
  #get and calculate values
  n, parentSets, w, deltaMax, deltaTrans, deltaConsist = getValues(path,alpha=alp
  #calculate d
  d = [ ('d',j,i) for j in range(n) for i in range(n) if j != i ]
  #calculate y (m=2)
  y = [ ('y',i,l+1) for i in range(n) for l in range(2)]
  #calculate r
  r = [ ('r',j,i) for j in range(n) for i in range(n) if j < i ]
  #calculate index array for rows and columns of the QUBO matrix
  indexQUBO = d + y + r
  iqLen = len(indexQUBO)
  #init QUBO matrix Q
  Q = [ [ 0. for j in range(iqLen) ] for i in range(iqLen) ]
  #calculate map for indexQUBO
  posOfIndex = calcPosOfIndex(indexQUBO)
  #add Hscore component to Q
  addHscore(Q, posOfIndex, n, parentSets, w)
  #add Hmax component to Q
  addHmax(Q,posOfIndex,n,deltaMax)
  #add Htrans component to Q
```

```
  addHtrans(Q, posOfIndex, n, deltaTrans)
  #add Hconsist component to Q
  addHconsist(Q, posOfIndex, n, deltaConsist)
  return Q,indexQUBO,posOfIndex,n

def addHscore(Q, posOfIndex, n, parentSets, w):
  for i in range(n):
    for parentSet in parentSets[i]:
      if parentSet != ():
        if len(parentSet) == 1:
          #elements of the diagonal
          j = parentSet[0]
          index = ('d',j,i)
          col = row = posOfIndex[index]
          Q[row][col] += w[i][parentSet]
        else:
          #elements outside the diagonal
          x = parentSet[0]; y = parentSet[1]
          indexR = ('d',x,i)
          indexC = ('d',y,i)
          row = posOfIndex[indexR]
          col = posOfIndex[indexC]
          Q[row][col] += w[i][parentSet]


def addHmax(Q, posOfIndex, n, deltaMax):
  for i in range(n):
    #Hmaxi(di,yi) = deltaMax[i] * (m - di - yi)**2
    toSquare = []
    #add m=2
    m = 2
    #add all dji
    for j in range(n):
      if j != i:
        toSquare.append(('d',j,i,-1))
    #add all yil for m=2
    toSquare.append(('y',i,1,-1))
    toSquare.append(('y',i,2,-2))
    #elements of the diagonal:
    #   square of elements (m excluded since it is a constant)
    #   double products with m
    for index in toSquare:
      col = row = posOfIndex[index[:3]]
      #add penalty multiplied by the square of the coefficient
      Q[row][col] += deltaMax[i] * (index[3]**2)
      #add penalty multiplied by m and the coefficient of the index
      Q[row][col] += deltaMax[i] * (2*m*index[3])
    #outside the diagonal: all pairs of indices
    for j in range(len(toSquare)):
      indexR = toSquare[j]
      row = posOfIndex[indexR[:3]]
      for k in range(j+1,len(toSquare)):
        indexC = toSquare[k]
```

```
        col = posOfIndex[indexC[:3]]
        #add penalty multiplied by the double product of the coefficients
        Q[row][col] += deltaMax[i] * (2*indexR[3]*indexC[3])

def addHtrans(Q, posOfIndex, n, deltaTrans):
  for i in range(n):
    for j in range(i+1,n):
      for k in range(j+1,n):
        #Htransijk(rij,rik,rjk) = deltaTrans * (rik + rij*rjk - rij*rik - rjk*ri
        rij = ('r',i,j)
        rik = ('r',i,k)
        rjk = ('r',j,k)
        #coeff for rik
        col = row = posOfIndex[rik]
        Q[row][col] += deltaTrans
        #coeff for rij*rjk
        row = posOfIndex[rij]
        col = posOfIndex[rjk]
        Q[row][col] += deltaTrans
        #coeff for rij*rik
        row = posOfIndex[rij]
        col = posOfIndex[rik]
        Q[row][col] += -deltaTrans
        #coeff for rjk*rik (rik comes before -> row)
        row = posOfIndex[rik]
        col = posOfIndex[rjk]
        Q[row][col] += -deltaTrans

def addHconsist(Q, posOfIndex, n, deltaConsist):
  for i in range(n):
    for j in range(i+1,n):
      #Hconsistij(dij,dji,rij) = deltaConsist*(dji*rij + dij - dij*rij)
      dij = ('d',i,j)
      dji = ('d',j,i)
      rij = ('r',i,j)
      #coeff for dji*rij
      row = posOfIndex[dji]
      col = posOfIndex[rij]
      Q[row][col] += deltaConsist
      #coeff for dij
      col = row = posOfIndex[dij]
      Q[row][col] += deltaConsist
      #coeff for dij*rij
      row = posOfIndex[dij]
      col = posOfIndex[rij]
      Q[row][col] += -deltaConsist

def calcPosOfIndex(indexQUBO):
  posOfIndex = {}
  for i in range(len(indexQUBO)):
    index = indexQUBO[i]
    posOfIndex[index] = i
  return posOfIndex
```

Attachment B Test data

B.1 Exhaustive search

n	problem	alpha	examples	optim	dataset	time	minY/expY
3	MHP	N/(ri*qi)	10000	—	MHPExp10K	00m02s	1.000000
3	MHP	N/(ri*qi)	100000	—	MHPExp100K	00m05s	1.000000
3	MHP	N/(ri*qi)	1000000	—	MHPExp1M	00m34s	1.000000
3	MHP	N/(ri*qi)	10000	—	MHP10K1	00m02s	1.000005
3	MHP	N/(ri*qi)	10000	—	MHP10K2	00m02s	1.000000
3	MHP	N/(ri*qi)	10000	—	MHP10K3	00m02s	1.000000
3	MHP	N/(ri*qi)	100000	—	MHP100K	00m05s	1.000000
3	MHP	N/(ri*qi)	1000000	—	MHP1M	00m36s	1.000000
3	MHP	1	10000	—	MHPExp10K	00m02s	1.000343
3	MHP	1	100000	—	MHPExp100K	00m05s	1.000049
3	MHP	1	1000000	—	MHPExp1M	00m33s	1.000006
3	MHP	1	10000	—	MHP10K1	00m02s	1.000396
3	MHP	1	10000	—	MHP10K2	00m02s	1.000357
3	MHP	1	10000	—	MHP10K3	00m02s	1.000356
3	MHP	1	100000	—	MHP100K	00m05s	1.000052
3	MHP	1	1000000	—	MHP1M	00m36s	1.000007
3	MHP	1/(ri*qi)	10000	—	MHPExp10K	00m02s	1.000000
3	MHP	1/(ri*qi)	100000	—	MHPExp100K	00m05s	1.000000
3	MHP	1/(ri*qi)	1000000	—	MHPExp1M	00m33s	1.000000
3	MHP	1/(ri*qi)	10000	—	MHP10K1	00m02s	1.000000
3	MHP	1/(ri*qi)	10000	—	MHP10K2	00m02s	1.000000
3	MHP	1/(ri*qi)	10000	—	MHP10K3	00m02s	1.000000
3	MHP	1/(ri*qi)	100000	—	MHP100K	00m05s	1.000000
3	MHP	1/(ri*qi)	1000000	—	MHP1M	00m36s	1.000000
3	MHP	1/ri	10000	y-r	MHPExp10K	00m02s	1.000000
3	MHP	1/ri	100000	y-r	MHPExp100K	00m05s	1.000000
3	MHP	1/ri	1000000	y-r	MHPExp1M	00m35s	1.000000
3	MHP	1/ri	10000	y-r	MHP10K1	00m02s	1.000000
3	MHP	1/ri	10000	y-r	MHP10K2	00m02s	1.000000
3	MHP	1/ri	10000	y-r	MHP10K3	00m02s	1.000000
3	MHP	1/ri	100000	y-r	MHP100K	00m05s	1.000000
3	MHP	1/ri	1000000	y-r	MHP1M	00m36s	1.000000

n	problem	alpha	examples	optim	dataset	time	minY/expY
4	LC4Vars	N/(ri*qi)	10000	—	LC4VarsExp10K	18m32s	1.027477
4	LC4Vars	N/(ri*qi)	10000	y	LC4VarsExp10K	00m06s	1.027477
4	LC4Vars	N/(ri*qi)	10000	y-r	LC4VarsExp10K	00m02s	1.027477
4	LC4Vars	N/(ri*qi)	100000	—	LC4VarsExp100K	19m43s	1.027595
4	LC4Vars	N/(ri*qi)	100000	y	LC4VarsExp100K	00m10s	1.027595
4	LC4Vars	N/(ri*qi)	100000	y-r	LC4VarsExp100K	00m06s	1.027595
4	LC4Vars	N/(ri*qi)	1000000	—	LC4VarsExp1M	18m58s	1.027605
4	LC4Vars	N/(ri*qi)	1000000	y	LC4VarsExp1M	00m53s	1.027605
4	LC4Vars	N/(ri*qi)	1000000	y-r	LC4VarsExp1M	00m49s	1.027605
4	LC4Vars	N/(ri*qi)	10000	y	LC4Vars10K1	00m06s	1.024918
4	LC4Vars	N/(ri*qi)	10000	y	LC4Vars10K2	00m06s	1.026049
4	LC4Vars	N/(ri*qi)	10000	y	LC4Vars10K3	00m06s	1.026708
4	LC4Vars	N/(ri*qi)	100000	—	LC4Vars100K	18m25s	1.027197
4	LC4Vars	N/(ri*qi)	100000	y	LC4Vars100K	00m10s	1.027197
4	LC4Vars	N/(ri*qi)	1000000	y	LC4Vars1M	00m54s	1.027634
4	LC4Vars	1	10000	y	LC4VarsExp10K	00m06s	1.002331
4	LC4Vars	1	100000	y	LC4VarsExp100K	00m10s	1.000163
4	LC4Vars	1	1000000	y	LC4VarsExp1M	00m54s	1.000009
4	LC4Vars	1	10000	y	LC4Vars10K1	00m06s	1.002707
4	LC4Vars	1	10000	y	LC4Vars10K2	00m06s	1.002405
4	LC4Vars	1	10000	y	LC4Vars10K3	00m06s	1.002376
4	LC4Vars	1	100000	y	LC4Vars100K	00m10s	1.000184
4	LC4Vars	1	1000000	y	LC4Vars1M	00m54s	1.000010
4	LC4Vars	1/(ri*qi)	10000	y	LC4VarsExp10K	00m06s	1.000000
4	LC4Vars	1/(ri*qi)	100000	y	LC4VarsExp100K	00m10s	1.000000
4	LC4Vars	1/(ri*qi)	1000000	y	LC4VarsExp1M	00m55s	1.000000
4	LC4Vars	1/(ri*qi)	10000	y	LC4Vars10K1	00m06s	1.000000
4	LC4Vars	1/(ri*qi)	10000	y	LC4Vars10K2	00m06s	1.000000
4	LC4Vars	1/(ri*qi)	10000	y	LC4Vars10K3	00m06s	1.000000
4	LC4Vars	1/(ri*qi)	100000	y	LC4Vars100K	00m10s	1.000000
4	LC4Vars	1/(ri*qi)	1000000	y	LC4Vars1M	00m55s	1.000000
4	LC4Vars	1/ri	10000	y-r	LC4VarsExp10K	00m02s	1.000000
4	LC4Vars	1/ri	100000	y-r	LC4VarsExp100K	00m06s	1.000000
4	LC4Vars	1/ri	1000000	y-r	LC4VarsExp1M	00m49s	1.000000
4	LC4Vars	1/ri	10000	y-r	LC4Vars10K1	00m02s	1.000000
4	LC4Vars	1/ri	10000	y-r	LC4Vars10K2	00m02s	1.000000
4	LC4Vars	1/ri	10000	y-r	LC4Vars10K3	00m02s	1.000000
4	LC4Vars	1/ri	100000	y-r	LC4Vars100K	00m06s	1.000000
4	LC4Vars	1/ri	1000000	y-r	LC4Vars1M	00m50s	1.000000

n	problem	alpha	examples	optim	dataset	time	minY/expY
5	LC	N/(ri*qi)	10000	y-r	LCExp10K	02m10s	1.031684
5	LC	N/(ri*qi)	100000	y-r	LCExp100K	02m19s	1.031860
5	LC	N/(ri*qi)	1000000	y-r	LCExp1M	03m59s	1.031876
5	LC	N/(ri*qi)	10000	y-r	LC10K1	02m07s	1.031071
5	LC	N/(ri*qi)	10000	y-r	LC10K2	02m06s	1.032102
5	LC	N/(ri*qi)	10000	y-r	LC10K3	02m09s	1.030286
5	LC	N/(ri*qi)	100000	y-r	LC100K	02m17s	1.031739
5	LC	N/(ri*qi)	1000000	y-r	LC1M	04m08s	1.031954
5	LC	1	10000	y-r	LCExp10K	02m07s	1.002164
5	LC	1	100000	y-r	LCExp100K	02m18s	1.000150
5	LC	1	1000000	y-r	LCExp1M	03m58s	1.000009
5	LC	1	10000	y-r	LC10K1	02m07s	1.001885
5	LC	1	10000	y-r	LC10K2	02m05s	1.002713
5	LC	1	10000	y-r	LC10K3	02m04s	1.002610
5	LC	1	100000	y-r	LC100K	02m18s	1.000160
5	LC	1	1000000	y-r	LC1M	04m01s	1.000014
5	LC	1/(ri*qi)	10000	y-r	LCExp10K	02m35s	1.000000
5	LC	1/(ri*qi)	100000	y-r	LCExp100K	02m17s	1.000000
5	LC	1/(ri*qi)	1000000	y-r	LCExp1M	04m03s	1.000000
5	LC	1/(ri*qi)	10000	y-r	LC10K1	02m08s	1.000000
5	LC	1/(ri*qi)	10000	y-r	LC10K2	02m06s	1.000000
5	LC	1/(ri*qi)	10000	y-r	LC10K3	02m09s	1.000000
5	LC	1/(ri*qi)	100000	y-r	LC100K	02m26s	1.000000
5	LC	1/(ri*qi)	1000000	y-r	LC1M	04m12s	1.000000
5	LC	1/ri	10000	y-r	LCExp10K	02m16s	1.000000
5	LC	1/ri	100000	y-r	LCExp100K	02m19s	1.000000
5	LC	1/ri	1000000	y-r	LCExp1M	03m57s	1.000000
5	LC	1/ri	10000	y-r	LC10K1	02m09s	1.000000
5	LC	1/ri	10000	y-r	LC10K2	02m08s	1.000000
5	LC	1/ri	10000	y-r	LC10K3	02m10s	1.000000
5	LC	1/ri	100000	y-r	LC100K	02m19s	1.000000
5	LC	1/ri	1000000	y-r	LC1M	04m06s	1.000000

B.2 Quantum annealing and simulated annealing

n	dataset	examples	method	reads	read t (μs)	classical t (s)	anneal t (s)	occour min	minY/expY
3	MHP10K1	10000	QA	10	20	0.362156	0.017941	1	0.999708
3	MHP10K1	10000	SA	10	—	0.360135	0.006716	1	0.999708
3	MHP10K1	10000	QA	100	20	0.363921	0.025795	5	1.000000
3	MHP10K1	10000	SA	100	—	0.362759	0.049065	1	1.000000
3	MHP10K2	10000	QA	10	20	0.374181	0.018060	1	0.999676
3	MHP10K2	10000	SA	10	—	0.368371	0.005559	1	0.999676
3	MHP10K2	10000	QA	100	20	0.365696	0.025402	5	1.000000
3	MHP10K2	10000	SA	100	—	0.361871	0.039010	1	1.000000
3	MHP10K3	10000	QA	10	20	0.378512	0.018088	2	0.999678
3	MHP10K3	10000	SA	10	—	0.365130	0.006013	1	0.999678
3	MHP10K3	10000	QA	100	20	0.372534	0.025316	4	1.000000
3	MHP10K3	10000	SA	100	—	0.360954	0.037873	1	1.000000
3	MHPExp10K	10000	QA	10	20	0.349854	0.017921	1	0.999633
3	MHPExp10K	10000	SA	10	—	0.339435	0.005542	1	0.999664
3	MHPExp10K	10000	QA	100	20	0.333640	0.025010	3	1.000000
3	MHPExp10K	10000	SA	100	—	0.345434	0.038173	1	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.707411	233	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.850209	153	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.755410	198	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.844210	178	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.741410	670	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.875009	137	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.759409	743	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.758210	137	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.652609	167	1.000000
3	MHPExp10K	10000	QA	10000	1	0.486059	0.808609	431	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.711004	252	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.865004	98	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.835803	145	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.710604	257	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.861805	226	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.623005	456	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.759005	93	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.751805	99	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.689405	46	1.000000
3	MHPExp10K	10000	QA	10000	99	0.372324	1.769405	487	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.417364	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.385839	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.386824	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.370190	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.373586	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.377526	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.370485	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.380858	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.453557	1	1.000000
3	MHPExp10K	10000	SA	10000	—	0.630718	3.449173	1	1.000000
3	MHP100K	100000	QA	10	20	3.429532	0.017931	1	0.999963
3	MHP100K	100000	SA	10	—	3.502270	0.005644	1	1.000000
3	MHP100K	100000	QA	100	20	3.421755	0.025447	1	1.000000
3	MHP100K	100000	SA	100	—	3.461259	0.038088	1	1.000000
3	MHP1M	1000000	QA	10	20	34.831013	0.017999	1	0.999996
3	MHP1M	1000000	SA	10	—	34.424582	0.006628	1	1.000000
3	MHP1M	1000000	QA	100	20	35.040288	0.025023	5	0.999996
3	MHP1M	1000000	SA	100	—	34.697142	0.037963	1	1.000000
4	LC4Vars10K1	10000	QA	100	20	0.537443	0.025707	2	0.998015
4	LC4Vars10K1	10000	SA	100	—	0.530938	0.059534	1	1.000000
4	LC4Vars10K1	10000	QA	1000	20	0.531983	0.112163	1	1.000000
4	LC4Vars10K1	10000	SA	1000	—	0.532473	0.560202	1	1.000000
4	LC4Vars10K2	10000	QA	100	20	0.543217	0.025782	1	0.997578
4	LC4Vars10K2	10000	SA	100	—	0.530686	0.067898	1	0.997578
4	LC4Vars10K2	10000	SA	100	—	0.533719	0.062354	1	1.000000
4	LC4Vars10K2	10000	QA	1000	20	0.527906	0.109403	3	1.000000
4	LC4Vars10K2	10000	SA	1000	—	0.532624	0.583813	1	1.000000
4	LC4Vars10K3	10000	QA	100	20	0.528559	0.027714	1	0.995724
4	LC4Vars10K3	10000	SA	100	—	0.532625	0.058939	1	1.000000
4	LC4Vars10K3	10000	QA	1000	20	0.529226	0.097721	4	1.000000
4	LC4Vars10K3	10000	SA	1000	—	0.531452	0.558011	1	1.000000

n	dataset	examples	method	reads	read t (μs)	classical t (s)	anneal t (s)	occour min	minY/expY
4	LC4VarsExp10K	10000	QA	10	20	0.508257	0.017992	1	0.997164
4	LC4VarsExp10K	10000	SA	10	—	0.530552	0.007963	1	0.997164
4	LC4VarsExp10K	10000	QA	100	20	0.528895	0.026918	1	0.997164
4	LC4VarsExp10K	10000	SA	100	—	0.528625	0.058721	1	1.000000
4	LC4VarsExp10K	10000	QA	1000	20	0.532579	0.123622	2	1.000000
4	LC4VarsExp10K	10000	SA	1000	—	0.533752	0.560213	1	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.815011	8	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.909010	9	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.822209	5	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.819408	1	0.997164
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.739807	10	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.834208	5	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.681008	4	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.803009	3	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.899410	2	1.000000
4	LC4VarsExp10K	10000	QA	10000	1	0.536397	0.801009	6	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.827002	7	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.827003	31	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.716205	17	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.695004	8	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.868604	2	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.710605	13	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.793805	3	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.701005	2	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.660605	19	1.000000
4	LC4VarsExp10K	10000	QA	10000	99	0.549742	1.875004	12	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.280223	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.353126	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.322288	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.514416	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.615415	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.555278	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.764129	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.659242	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.585171	1	1.000000
4	LC4VarsExp10K	10000	SA	10000	—	0.562042	5.564676	1	1.000000
4	LC4Vars100K	100000	QA	100	20	4.979415	0.026337	1	0.999637
4	LC4Vars100K	100000	SA	100	—	4.927170	0.065333	1	1.000000
4	LC4Vars100K	100000	QA	1000	20	4.948299	0.123100	1	1.000000
4	LC4Vars100K	100000	SA	1000	—	4.855013	0.574553	1	1.000000
4	LC4Vars1M	1000000	QA	100	20	48.755726	0.026413	1	1.000000
4	LC4Vars1M	1000000	SA	100	—	48.533559	0.060151	1	1.000000
4	LC4Vars1M	1000000	QA	1000	20	48.418332	0.124221	4	1.000000
4	LC4Vars1M	1000000	SA	1000	—	49.425066	0.565658	1	1.000000
5	LC10K1	10000	QA	1000	20	1.173840	0.126012	1	0.994997
5	LC10K1	10000	SA	1000	—	1.182128	0.846210	1	1.000000
5	LC10K1	10000	QA	10000	20	1.177951	1.002082	1	0.997614
5	LC10K1	10000	SA	10000	—	1.181461	8.550748	1	1.000000
5	LC10K2	10000	QA	1000	20	1.179590	0.130784	1	0.991980
5	LC10K2	10000	SA	1000	—	1.157821	0.790696	1	1.000000
5	LC10K2	10000	QA	10000	20	1.174526	1.159282	1	0.997547
5	LC10K2	10000	SA	10000	—	1.180107	7.867765	1	1.000000
5	LC10K3	10000	QA	1000	20	1.199878	0.113861	1	0.995873
5	LC10K3	10000	SA	1000	—	1.196721	0.355444	1	1.000000
5	LC10K3	10000	QA	10000	20	1.173996	1.008880	1	1.000000
5	LC10K3	10000	SA	10000	—	1.198699	4.409697	1	1.000000
5	LCExp10K	10000	QA	10	20	1.137277	0.018244	1	0.974264
5	LCExp10K	10000	SA	10	—	1.140076	0.012452	1	0.997402
5	LCExp10K	10000	QA	100	20	1.121770	0.027345	1	0.988939
5	LCExp10K	10000	QA	100	20	1.146159	0.026473	1	0.994030
5	LCExp10K	10000	SA	100	—	1.135664	0.095751	1	0.997402
5	LCExp10K	10000	SA	100	—	1.107063	0.093904	1	1.000000
5	LCExp10K	10000	SA	100	—	1.142254	0.119964	1	0.997402
5	LCExp10K	10000	QA	1000	10	1.195837	0.102925	1	0.996427
5	LCExp10K	10000	QA	1000	10	1.178002	0.098845	1	0.995027
5	LCExp10K	10000	QA	1000	10	1.180713	0.105130	1	0.997402
5	LCExp10K	10000	QA	1000	10	1.167101	0.113845	1	0.993081
5	LCExp10K	10000	QA	1000	10	1.283982	0.106086	1	0.997402
5	LCExp10K	10000	QA	1000	10	1.206849	0.109087	1	0.991914
5	LCExp10K	10000	QA	1000	10	1.180602	0.099525	1	0.997402
5	LCExp10K	10000	QA	1000	10	1.176487	0.112529	1	0.996427
5	LCExp10K	10000	QA	1000	10	1.199365	0.115728	1	0.996427
5	LCExp10K	10000	QA	1000	10	1.180974	0.118085	1	0.997402
5	LCExp10K	10000	QA	1000	20	1.134868	0.116976	1	0.992998
5	LCExp10K	10000	QA	1000	20	1.140937	0.106515	1	0.996427
5	LCExp10K	10000	QA	1000	999	1.179768	1.094603	1	1.000000
5	LCExp10K	10000	QA	1000	999	1.190121	1.108684	1	0.997402
5	LCExp10K	10000	QA	1000	999	1.180955	1.103161	1	0.997402
5	LCExp10K	10000	QA	1000	999	1.191405	1.088082	1	0.997402
5	LCExp10K	10000	QA	1000	999	1.199359	1.101284	1	0.997402
5	LCExp10K	10000	QA	1000	999	1.191024	1.105081	1	0.997402
5	LCExp10K	10000	QA	1000	999	1.180730	1.114081	1	0.997402
5	LCExp10K	10000	QA	1000	999	1.199097	1.096521	1	0.997402
5	LCExp10K	10000	QA	1000	999	1.197034	1.095163	1	0.995487
5	LCExp10K	10000	QA	1000	999	1.212552	1.096801	1	0.997402
5	LCExp10K	10000	SA	1000	—	1.108284	0.864804	1	1.000000
5	LCExp10K	10000	SA	1000	—	1.144178	0.868138	1	1.000000

n	dataset	examples	method	reads	read t (μs)	classical t (s)	anneal t (s)	occour min	minY/expY
5	LCExp10K	10000	QA	10000	1	1.261356	0.961411	1	0.994268
5	LCExp10K	10000	QA	10000	1	1.261356	0.955011	1	0.997402
5	LCExp10K	10000	QA	10000	1	1.261356	0.716610	1	0.997402
5	LCExp10K	10000	QA	10000	1	1.261356	0.871011	1	0.997402
5	LCExp10K	10000	QA	10000	1	1.261356	0.861811	1	1.000000
5	LCExp10K	10000	QA	10000	1	1.261356	0.978209	2	1.000000
5	LCExp10K	10000	QA	10000	1	1.261356	0.803011	1	1.000000
5	LCExp10K	10000	QA	10000	1	1.261356	0.870607	1	0.996427
5	LCExp10K	10000	QA	10000	1	1.261356	0.876609	1	0.997402
5	LCExp10K	10000	QA	10000	1	1.261356	0.806210	1	1.000000
5	LCExp10K	10000	QA	10000	10	1.201150	1.084206	1	0.997402
5	LCExp10K	10000	QA	10000	10	1.179093	0.829407	1	0.997402
5	LCExp10K	10000	QA	10000	10	1.191132	0.963007	1	1.000000
5	LCExp10K	10000	QA	10000	10	1.170084	0.872607	1	0.997402
5	LCExp10K	10000	QA	10000	10	1.181130	0.970605	1	0.997402
5	LCExp10K	10000	QA	10000	10	1.237088	0.945810	1	0.996427
5	LCExp10K	10000	QA	10000	10	1.175303	0.936207	1	0.997402
5	LCExp10K	10000	QA	10000	10	1.194095	0.969408	1	0.996427
5	LCExp10K	10000	QA	10000	10	1.175989	1.115407	1	0.997402
5	LCExp10K	10000	QA	10000	10	1.178887	0.871806	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.110277	0.980873	1	0.996427
5	LCExp10K	10000	QA	10000	20	1.144705	0.921473	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.144318	0.960077	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.139122	0.962072	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.150459	1.031276	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.138436	0.951679	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.175104	1.022480	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.179586	1.025279	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.144236	0.967283	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.144573	0.973280	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.141369	0.945479	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.155771	0.960876	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.143616	1.141673	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.144915	1.080074	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.160216	0.928277	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.141797	1.097671	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.141012	0.987280	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.207491	0.973282	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.177490	1.023683	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.150658	0.949277	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.301247	1.154873	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.294415	1.035283	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.245346	0.842273	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.254860	1.053273	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.233146	1.008877	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.267852	0.877879	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.231548	0.990881	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.252932	1.131670	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.280567	1.035674	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.253815	1.169277	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.239584	1.030878	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.242852	0.910277	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.226914	1.139277	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.251339	0.998487	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.304462	1.014080	2	1.000000
5	LCExp10K	10000	QA	10000	20	1.115940	0.933071	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.278901	0.877879	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.258630	0.983280	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.359669	1.025276	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.233397	1.126873	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.299235	0.929078	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.261896	1.026877	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.250909	1.160879	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.280070	0.987678	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.247256	1.078881	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.245278	1.031281	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.240388	0.988081	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.254415	0.857075	1	0.997402
5	LCExp10K	10000	QA	10000	20	1.174113	1.012079	2	0.997402
5	LCExp10K	10000	QA	10000	20	1.237807	0.969281	1	1.000000
5	LCExp10K	10000	QA	10000	20	1.241410	1.136873	1	0.997402
5	LCExp10K	10000	QA	10000	50	1.188859	1.287407	1	1.000000
5	LCExp10K	10000	QA	10000	50	1.182790	1.290607	1	1.000000
5	LCExp10K	10000	QA	10000	50	1.189070	1.347807	1	0.997402
5	LCExp10K	10000	QA	10000	50	1.178715	1.405008	1	1.000000
5	LCExp10K	10000	QA	10000	50	1.179907	1.394604	1	0.995487
5	LCExp10K	10000	QA	10000	50	1.170825	1.289408	2	0.997402
5	LCExp10K	10000	QA	10000	50	1.187135	1.421805	1	0.997402
5	LCExp10K	10000	QA	10000	50	1.193225	1.453409	1	0.997402
5	LCExp10K	10000	QA	10000	50	1.170651	1.289808	1	1.000000
5	LCExp10K	10000	QA	10000	50	1.192143	1.300606	1	0.997402
5	LCExp10K	10000	QA	10000	99	1.173548	1.680605	2	1.000000
5	LCExp10K	10000	QA	10000	99	1.169557	1.863008	1	0.997402
5	LCExp10K	10000	QA	10000	99	1.184225	1.837804	2	0.997402
5	LCExp10K	10000	QA	10000	99	1.188946	1.849009	1	0.997402
5	LCExp10K	10000	QA	10000	99	1.208245	1.805407	1	1.000000
5	LCExp10K	10000	QA	10000	99	1.194377	1.798605	1	1.000000
5	LCExp10K	10000	QA	10000	99	1.181210	1.791408	1	0.997402
5	LCExp10K	10000	QA	10000	99	1.179629	1.685807	1	0.997402
5	LCExp10K	10000	QA	10000	99	1.192189	1.878605	1	1.000000
5	LCExp10K	10000	QA	10000	99	1.180643	1.809808	1	1.000000

n	dataset	examples	method	reads	read t (μs)	classical t (s)	anneal t (s)	occour min	minY/expY
5	LCExp10K	10000	SA	10000	—	1.139265	8.595373	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	8.724242	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	8.977068	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	9.189863	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	8.951978	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	8.781477	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	9.048075	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	9.030483	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	9.013093	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	9.057225	1	1.000000
5	LCExp10K	10000	SA	10000	—	1.269766	9.084269	1	1.000000
5	LCExp10K	10000	SA	100000	—	1.150469	86.379691	1	1.000000
5	LC100K	100000	QA	1000	20	11.575061	0.110379	1	0.999127
5	LC100K	100000	SA	1000	—	11.608257	0.357252	1	1.000000
5	LC100K	100000	QA	10000	20	11.346828	0.926675	1	0.999677
5	LC100K	100000	SA	10000	—	11.344632	3.454631	1	1.000000
5	LCExp100K	100000	SA	100	—	11.008501	0.096412	1	0.999686
5	LCExp100K	100000	QA	1000	20	10.965920	0.113183	1	1.000000
5	LCExp100K	100000	SA	1000	—	11.013276	0.881921	1	0.999686
5	LCExp100K	100000	QA	10000	20	11.280494	0.983680	1	0.999686
5	LCExp100K	100000	SA	10000	—	10.973003	8.878658	1	1.000000
5	LC1M	1000000	QA	1000	20	115.316532	0.113854	1	0.999901
5	LC1M	1000000	SA	1000	—	116.065992	0.348941	1	1.000000
5	LC1M	1000000	QA	10000	20	114.628093	0.933477	1	1.000000
5	LC1M	1000000	SA	10000	—	114.579128	3.483646	1	1.000000
9	Waste10K1	10000	QA	10000	20	8.605131	1.413007	1	0.999409
9	Waste10K1	10000	QA	10000	20	8.444193	1.444203	1	0.983945
9	Waste10K1	10000	QA	10000	20	8.505734	1.282605	1	0.984453
9	Waste10K1	10000	QA	10000	20	8.499244	1.123408	1	0.985872
9	Waste10K1	10000	QA	10000	20	8.456811	1.596205	1	0.999416
9	Waste10K1	10000	QA	10000	20	8.457538	1.347006	1	0.987501
9	Waste10K1	10000	QA	10000	20	8.421549	1.537803	1	0.983228
9	Waste10K1	10000	QA	10000	20	10.587118	1.357402	1	0.982451
9	Waste10K1	10000	QA	10000	20	10.283708	1.491806	1	0.967886
9	Waste10K1	10000	QA	10000	20	10.315830	1.275807	1	0.997456
9	Waste10K1	10000	SA	10000	—	10.308187	15.588348	1	1.014124
9	Waste10K1	10000	SA	10000	—	10.347247	15.607900	1	1.014124
9	Waste10K1	10000	SA	10000	—	10.252288	15.776450	1	1.014113
9	Waste10K1	10000	SA	10000	—	10.261675	15.580355	1	1.014121
9	Waste10K1	10000	SA	10000	—	10.286404	15.640694	1	1.014117
9	Waste10K1	10000	SA	10000	—	10.361668	15.618543	1	1.014116
9	Waste10K1	10000	SA	10000	—	10.252029	15.841684	1	1.014113
9	Waste10K1	10000	SA	10000	—	10.316878	15.887953	1	1.014142
9	Waste10K1	10000	SA	10000	—	10.355615	15.877411	1	1.014132
9	Waste10K1	10000	SA	10000	—	10.297592	15.649133	1	1.014111
9	WasteExp10K	10000	QA	10000	1	8.998658	0.929811	1	0.972710
9	WasteExp10K	10000	QA	10000	1	8.998658	1.369009	1	0.925900
9	WasteExp10K	10000	QA	10000	1	8.998658	1.078210	1	0.972990
9	WasteExp10K	10000	QA	10000	1	8.998658	1.213009	1	0.957698
9	WasteExp10K	10000	QA	10000	1	8.998658	0.988606	1	0.940467
9	WasteExp10K	10000	QA	10000	1	8.998658	1.096616	1	0.984083
9	WasteExp10K	10000	QA	10000	1	8.998658	1.344600	1	0.971160
9	WasteExp10K	10000	QA	10000	1	8.998658	1.327807	1	0.983632
9	WasteExp10K	10000	QA	10000	1	8.998658	1.093811	1	0.954947
9	WasteExp10K	10000	QA	10000	1	8.998658	1.099011	1	0.954953
9	WasteExp10K	10000	QA	10000	99	8.764840	2.367401	1	0.968935
9	WasteExp10K	10000	QA	10000	99	8.764840	1.970604	1	0.981900
9	WasteExp10K	10000	QA	10000	99	8.764840	1.997405	1	0.984985
9	WasteExp10K	10000	QA	10000	99	8.764840	2.271404	1	0.971242
9	WasteExp10K	10000	QA	10000	99	8.764840	1.879005	1	1.002038
9	WasteExp10K	10000	QA	10000	99	8.764840	2.087803	1	1.012836
9	WasteExp10K	10000	QA	10000	99	8.764840	2.271801	1	0.998487
9	WasteExp10K	10000	QA	10000	99	8.764840	2.190200	1	0.996329
9	WasteExp10K	10000	QA	10000	99	8.764840	1.992613	1	0.996020
9	WasteExp10K	10000	QA	10000	99	8.764840	1.902207	1	0.985385
9	WasteExp10K	10000	SA	10000	—	8.436820	12.260654	1	1.014533
9	WasteExp10K	10000	SA	10000	—	8.389975	12.244263	1	1.014568
9	WasteExp10K	10000	SA	10000	—	9.144522	12.886598	1	1.014523
9	WasteExp10K	10000	SA	10000	—	9.144522	12.805718	1	1.014530
9	WasteExp10K	10000	SA	10000	—	9.144522	13.073703	1	1.014561
9	WasteExp10K	10000	SA	10000	—	9.144522	13.258027	1	1.014534
9	WasteExp10K	10000	SA	10000	—	9.144522	13.315627	1	1.014531
9	WasteExp10K	10000	SA	10000	—	9.144522	12.717648	1	1.014514
9	WasteExp10K	10000	SA	10000	—	9.144522	13.010751	1	1.014523
9	WasteExp10K	10000	SA	10000	—	9.144522	12.988084	1	1.014518
9	WasteExp10K	10000	SA	10000	—	9.144522	13.202679	1	1.014531
9	WasteExp10K	10000	SA	10000	—	9.144522	13.478939	1	1.014541
9	WasteExp10K	10000	SA	100000	—	8.576871	123.234959	1	1.014565
9	Waste2P10K1	10000	QA	10000	20	8.620387	1.311814	1	0.972727
9	Waste2P10K1	10000	QA	10000	20	8.478520	1.586600	1	0.970544
9	Waste2P10K1	10000	QA	10000	20	8.625004	1.314204	1	0.982166
9	Waste2P10K1	10000	QA	10000	20	8.450218	1.365407	1	0.949210
9	Waste2P10K1	10000	QA	10000	20	8.415698	1.336603	1	0.971093
9	Waste2P10K1	10000	QA	10000	20	8.454588	1.134605	1	0.972716
9	Waste2P10K1	10000	QA	10000	20	8.437244	1.148606	1	0.968795
9	Waste2P10K1	10000	QA	10000	20	8.611512	1.489395	1	0.984475
9	Waste2P10K1	10000	QA	10000	20	8.433655	1.264213	1	0.968474
9	Waste2P10K1	10000	QA	10000	20	8.457935	1.197003	1	0.974053
9	Waste2P10K1	10000	SA	10000	—	8.492096	12.462333	1	0.999979
9	Waste2P10K1	10000	SA	10000	—	8.498197	12.317298	1	0.999995
9	Waste2P10K1	10000	SA	10000	—	8.417107	12.389569	1	0.999984
9	Waste2P10K1	10000	SA	10000	—	8.459632	12.507718	1	1.000003
9	Waste2P10K1	10000	SA	10000	—	8.436825	12.393033	1	1.000002
9	Waste2P10K1	10000	SA	10000	—	8.439956	12.297119	1	0.999968
9	Waste2P10K1	10000	SA	10000	—	8.438682	12.405044	1	0.999964
9	Waste2P10K1	10000	SA	10000	—	8.507094	12.588365	1	0.999977
9	Waste2P10K1	10000	SA	10000	—	8.497457	12.359603	1	0.999983
9	Waste2P10K1	10000	SA	10000	—	8.476306	12.345614	1	0.999976

n	dataset	examples	method	reads	read t (μs)	classical t (s)	anneal t (s)	occour min	minY/expY
9	Waste2PExp10K	10000	SA	1000	—	8.544603	1.260173	1	0.999941
9	Waste2PExp10K	10000	SA	1000	—	8.458324	1.244875	1	0.999971
9	Waste2PExp10K	10000	SA	1000	—	8.472053	1.249800	1	0.999970
9	Waste2PExp10K	10000	SA	1000	—	8.365775	1.240123	1	0.999939
9	Waste2PExp10K	10000	SA	1000	—	8.395941	1.368983	1	0.999953
9	Waste2PExp10K	10000	QA	10000	1	8.533808	1.389809	1	0.929445
9	Waste2PExp10K	10000	QA	10000	1	8.533808	1.151012	1	0.968656
9	Waste2PExp10K	10000	QA	10000	1	8.533808	1.114208	1	0.968573
9	Waste2PExp10K	10000	QA	10000	1	8.533808	1.066216	1	0.931451
9	Waste2PExp10K	10000	QA	10000	1	8.533808	0.945411	1	0.941589
9	Waste2PExp10K	10000	QA	10000	1	8.533808	0.898611	1	0.955074
9	Waste2PExp10K	10000	QA	10000	1	8.533808	0.988208	1	0.956013
9	Waste2PExp10K	10000	QA	10000	1	8.533808	1.255410	1	0.926542
9	Waste2PExp10K	10000	QA	10000	1	8.533808	1.377407	1	0.956527
9	Waste2PExp10K	10000	QA	10000	1	8.533808	1.283014	1	0.938712
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.353003	1	0.973063
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.201006	1	0.957771
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.049407	1	0.969366
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.059808	1	0.987595
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.192613	1	0.958410
9	Waste2PExp10K	10000	QA	10000	99	8.975025	1.941803	1	0.982947
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.334603	1	0.969991
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.089815	1	0.983927
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.209803	1	0.986727
9	Waste2PExp10K	10000	QA	10000	99	8.975025	2.238211	1	0.984116
9	Waste2PExp10K	10000	SA	10000	—	9.019706	13.059167	1	0.999943
9	Waste2PExp10K	10000	SA	10000	—	9.019706	13.219645	1	0.999974
9	Waste2PExp10K	10000	SA	10000	—	9.019706	13.174889	1	0.999953
9	Waste2PExp10K	10000	SA	10000	—	9.019706	12.875711	1	0.999981
9	Waste2PExp10K	10000	SA	10000	—	9.019706	13.161383	1	0.999966
9	Waste2PExp10K	10000	SA	10000	—	9.019706	13.271872	1	0.999960
9	Waste2PExp10K	10000	SA	10000	—	9.019706	13.077865	1	0.999967
9	Waste2PExp10K	10000	SA	10000	—	9.019706	13.275262	1	0.999972
9	Waste2PExp10K	10000	SA	10000	—	9.019706	12.917117	1	0.999957
9	Waste2PExp10K	10000	SA	10000	—	9.019706	13.454678	1	0.999960
9	Waste2PExp100K	100000	QA	10000	20	89.990524	1.599413	1	0.956750
9	Waste2PExp100K	100000	QA	10000	20	87.068457	1.224201	1	0.984437
9	Waste2PExp100K	100000	QA	10000	20	91.531110	1.180199	1	0.984934
9	Waste2PExp100K	100000	QA	10000	20	91.864839	1.331403	1	0.970317
9	Waste2PExp100K	100000	QA	10000	20	90.366568	1.167796	1	0.969629
9	Waste2PExp100K	100000	QA	10000	20	90.366568	1.440198	1	0.974721
9	Waste2PExp100K	100000	QA	10000	20	90.366568	1.608202	1	0.958034
9	Waste2PExp100K	100000	QA	10000	20	90.366568	1.128201	1	0.967572
9	Waste2PExp100K	100000	QA	10000	20	90.366568	1.283009	1	0.957777
9	Waste2PExp100K	100000	QA	10000	20	90.366568	1.320599	1	0.958372
9	Waste2PExp100K	100000	SA	10000	—	92.430646	14.083311	1	0.999990
9	Waste2PExp100K	100000	SA	10000	—	91.721385	14.079207	1	0.999998
9	Waste2PExp100K	100000	SA	10000	—	91.986414	14.063060	1	0.999991
9	Waste2PExp100K	100000	SA	10000	—	91.776917	14.351061	1	0.999996
9	Waste2PExp100K	100000	SA	10000	—	91.554606	13.038172	1	0.999989
9	Waste2PExp100K	100000	SA	10000	—	90.504380	14.109298	1	0.999989
9	Waste2PExp100K	100000	SA	10000	—	91.148772	13.982602	1	0.999995
9	Waste2PExp100K	100000	SA	10000	—	93.371761	13.200915	1	0.999995
9	Waste2PExp100K	100000	SA	10000	—	83.743568	12.843314	1	0.999987
9	Waste2PExp100K	100000	SA	10000	—	83.661995	12.754342	1	0.999992
9	Waste2PExp100K	100000	SA	100000	—	91.315494	138.546413	1	0.999998
9	Waste2PDep10K1	10000	SA	1000	—	8.601953	1.293696	1	0.999901
9	Waste2PDep10K1	10000	QA	10000	20	9.170116	1.571799	1	0.977697
9	Waste2PDep10K1	10000	QA	10000	20	9.338558	1.233802	1	0.974654
9	Waste2PDep10K1	10000	QA	10000	20	9.371993	1.501804	1	0.984424
9	Waste2PDep10K1	10000	QA	10000	20	9.255645	1.187003	1	0.978694
9	Waste2PDep10K1	10000	QA	10000	20	9.413566	1.164206	1	0.966518
9	Waste2PDep10K1	10000	QA	10000	20	9.289297	1.122207	1	0.976461
9	Waste2PDep10K1	10000	QA	10000	20	9.280823	1.144605	1	0.964834
9	Waste2PDep10K1	10000	QA	10000	20	9.258405	1.470604	1	0.961837
9	Waste2PDep10K1	10000	QA	10000	20	9.315922	1.600999	1	0.968934
9	Waste2PDep10K1	10000	QA	10000	20	9.381468	1.417407	1	0.969673
9	Waste2PDep10K1	10000	SA	10000	—	8.522417	12.468923	1	0.999900
9	Waste2PDep10K1	10000	SA	10000	—	8.556592	12.460969	1	0.999910
9	Waste2PDep10K1	10000	SA	10000	—	9.224676	13.495750	1	0.999885
9	Waste2PDep10K1	10000	SA	10000	—	9.313079	13.518367	1	0.999871
9	Waste2PDep10K1	10000	SA	10000	—	9.319078	13.711563	1	0.999912
9	Waste2PDep10K1	10000	SA	10000	—	9.446271	13.751491	1	0.999910
9	Waste2PDep10K1	10000	SA	10000	—	9.256113	13.696978	1	0.999776
9	Waste2PDep10K1	10000	SA	10000	—	9.240760	13.890829	1	0.999773
9	Waste2PDep10K1	10000	SA	10000	—	9.370476	13.605993	1	0.999820
9	Waste2PDep10K1	10000	SA	10000	—	9.343248	13.613120	1	0.999820
9	Waste2PDep10K1	10000	SA	10000	—	9.165719	13.754115	1	0.999856
9	Waste2PDep10K1	10000	SA	10000	—	9.490840	13.555062	1	0.999857
9	Waste2PDep10K1	10000	SA	100000	—	8.745802	125.192117	1	0.999872
9	Waste2PDep10K1	10000	SA	100000	—	8.624878	125.278166	1	0.999922
9	Waste2PDep10K1	10000	SA	1000000	—	8.620946	1267.283534	1	0.999952

n	dataset	examples	method	reads	read t (μs)	classical t (s)	anneal t (s)	occour min	minY/expY
9	Waste2PDepExp10K	10000	SA	1000	—	8.428075	1.239364	1	0.999705
9	Waste2PDepExp10K	10000	SA	1000	—	8.545214	1.223667	1	0.999843
9	Waste2PDepExp10K	10000	QA	10000	20	8.434563	1.522016	1	0.978450
9	Waste2PDepExp10K	10000	QA	10000	20	8.485884	1.363633	1	0.957673
9	Waste2PDepExp10K	10000	QA	10000	20	17.598860	1.189631	1	0.972740
9	Waste2PDepExp10K	10000	QA	10000	20	8.548807	1.486426	1	0.949691
9	Waste2PDepExp10K	10000	QA	10000	20	8.345451	1.510818	1	0.953954
9	Waste2PDepExp10K	10000	QA	10000	20	8.462990	1.331637	1	0.972354
9	Waste2PDepExp10K	10000	QA	10000	20	8.375635	1.494033	1	0.979865
9	Waste2PDepExp10K	10000	QA	10000	20	8.338319	1.077228	1	0.949999
9	Waste2PDepExp10K	10000	QA	10000	20	8.834079	1.185233	1	0.951309
9	Waste2PDepExp10K	10000	QA	10000	99	8.524349	1.956193	1	0.979435
9	Waste2PDepExp10K	10000	QA	10000	99	8.417554	1.920195	1	0.969465
9	Waste2PDepExp10K	10000	QA	10000	99	8.488236	2.228599	1	0.990980
9	Waste2PDepExp10K	10000	QA	10000	99	8.534696	1.976590	1	0.978685
9	Waste2PDepExp10K	10000	QA	10000	99	8.620525	2.250199	1	0.984099
9	Waste2PDepExp10K	10000	QA	10000	99	8.675677	2.326599	1	0.971842
9	Waste2PDepExp10K	10000	QA	10000	99	8.615396	2.246195	1	0.974115
9	Waste2PDepExp10K	10000	QA	10000	99	8.595241	1.990200	1	0.988452
9	Waste2PDepExp10K	10000	QA	10000	99	8.431655	1.940599	1	0.976816
9	Waste2PDepExp10K	10000	QA	10000	99	8.584632	2.129401	1	0.966391
9	Waste2PDepExp10K	10000	QA	10000	99	8.360875	1.879803	1	0.956184
9	Waste2PDepExp10K	10000	QA	10000	99	8.761544	2.218200	1	0.978580
9	Waste2PDepExp10K	10000	QA	10000	99	8.510175	2.348601	1	0.978024
9	Waste2PDepExp10K	10000	QA	10000	99	8.566428	2.253002	1	0.962412
9	Waste2PDepExp10K	10000	QA	10000	99	8.310475	2.178596	1	0.976802
9	Waste2PDepExp10K	10000	QA	10000	99	8.478947	2.401397	1	0.992603
9	Waste2PDepExp10K	10000	QA	10000	99	8.477078	2.358997	1	0.979315
9	Waste2PDepExp10K	10000	QA	10000	99	8.345435	2.166598	1	0.987317
9	Waste2PDepExp10K	10000	QA	10000	99	8.472615	1.972199	1	0.970648
9	Waste2PDepExp10K	10000	QA	10000	99	8.455641	2.052598	1	0.963561
9	Waste2PDepExp10K	10000	SA	10000	—	8.404960	12.302949	1	0.999838
9	Waste2PDepExp10K	10000	SA	10000	—	8.444766	12.182779	1	0.999855
9	Waste2PDepExp10K	10000	SA	100000	—	8.477795	122.249096	1	0.999939
9	Waste2PDepExp10K	10000	SA	1000000	—	8.376327	1245.177095	1	0.999931
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.446599	1	0.962539
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.400204	1	0.933030
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.461804	1	0.967949
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.199005	1	0.969667
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.329399	1	0.956477
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.472599	1	0.960055
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.233004	1	0.970359
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.566198	1	0.971531
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.257003	1	0.950580
9	Waste2PDepExp100K	100000	QA	10000	20	91.946108	1.366204	1	0.962640
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	14.058488	1	0.999978
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.992015	1	0.999834
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.908090	1	0.999997
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.882160	1	0.999981
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.819517	1	0.999956
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.900489	1	0.999972
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.973490	1	0.999966
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.934883	1	0.999957
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.921525	1	0.999960
9	Waste2PDepExp100K	100000	SA	10000	—	90.335706	13.809688	1	0.999977
9	Waste2PDepExp1M	1000000	QA	10000	20	920.127600	1.161409	1	0.971643
9	Waste2PDepExp1M	1000000	QA	10000	20	920.127600	1.223009	1	0.967097
9	Waste2PDepExp1M	1000000	QA	10000	20	920.127600	1.185802	1	0.948361
9	Waste2PDepExp1M	1000000	QA	10000	20	920.127600	1.129001	1	0.968616
9	Waste2PDepExp1M	1000000	QA	10000	20	920.127600	1.442607	1	0.980377
9	Waste2PDepExp1M	1000000	QA	10000	20	831.010458	1.429812	1	0.963692
9	Waste2PDepExp1M	1000000	QA	10000	20	831.010458	1.133003	1	0.962586
9	Waste2PDepExp1M	1000000	QA	10000	20	831.010458	1.167414	1	0.957515
9	Waste2PDepExp1M	1000000	QA	10000	20	831.010458	1.612205	1	0.964473
9	Waste2PDepExp1M	1000000	QA	10000	20	831.010458	1.346215	1	0.966705