The Journal of Systems & Software 201 (2023) 111683

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Dynamic analysis of quantum annealing programs☆

Ricardo Pérez-Castillo [a,b,*], Luis Jiménez-Navajas [b], Mario Piattini [b]

[a] Faculty of Social Sciences & IT, University of Castilla-La Mancha, Avda. Real Fábrica de Seda, s/n, 45600, Talavera de la Reina, Spain
[b] Information Technology & Systems Institute (ITSI), University of Castilla-La Mancha, Paseo de la Universidad 4 13071, Ciudad Real, Spain

ARTICLE INFO

Article history:
Received 4 June 2022
Received in revised form 15 March 2023
Accepted 21 March 2023
Available online 24 March 2023

Keywords:
Quantum annealing
Quantum software engineering
D-wave
Dynamic analysis
Reverse engineering
Knowledge discovery metamodel

ABSTRACT

Quantum software engineering is emerging as a relevant field, as it deals with the challenges of producing the new quantum software, whose adoption is steadily increasing. Quantum annealing software has gained a certain market penetration, demonstrating a good performance for optimization problems. However, there are no reverse engineering techniques with which to discover the underlying optimization problem definitions (the Hamiltonian functions to be minimized). Problem definitions are, in turn, dynamically defined using classical software, and can evolve over time, which make their accurate comprehension and abstract representation difficult. This paper, therefore, presents a dynamic analysis technique for D-Wave (Python) programs with which to reverse Hamiltonian expressions, and which are additionally represented according to the Knowledge Discovery Metamodel. The usage of this standard makes it possible to represent the reversed Hamiltonians in combination with other parts of classical–quantum software systems. In order to facilitate its adoption, the proposed technique has been empirically validated through a case study with 27 D-Wave programs that demonstrates its effectiveness and efficiency. The main implication of this research is that the proposed technique helps modernize quantum annealing software alongside hybrid software systems.

1. Introduction

Quantum Computing (QC) has now gained a great deal of relevance thanks to the promising computational power that makes it possible to deal with problems that are impossible with classical computing in a reasonable timeframe (Córcoles et al., 2020; Bozzo-Rey et al., 2019). The present decade (2020s) has been called the "quantum decade", in which "quantum computing is poised to expand the scope and complexity of the business problems we can solve" (IBM, 2021), thus offering a true "quantum advantage".

Quantum computing is based on the obscure principles of quantum mechanics, such as superposition and entanglement. It is currently possible to distinguish two main quantum computing paradigms: quantum gate computing and adiabatic QC, also known as Quantum Annealing (QA) (McGeoch, 2020). On the one hand, gate-based computing is also known as universal quantum computing because it can, theoretically, address all the target problems (since it is Turing-complete), while quantum annealers are used only to address optimization problems. On the other hand, most of the gate-based quantum computers are also known as NISQ (Liu and Zhou, 2020) (Noisy Intermediate-Scale Quantum) devices and are, to a certain degree, still limited (Córcoles et al., 2020), while quantum annealing computers work with a built-in solution for a specific purpose and have achieved a certain market penetration, as is the case of D-Wave (Kahn, 2021; Dridi and Alghassi, 2017). Nevertheless, this paradigm still receives a certain amount of criticism, since some authors argue that "quantum annealers have been applied mostly to cases where the annealing is stochastic, that means it might be relatively easy for a classical computer to simulate what the quantum annealer is doing" (Preskill, 2018).

Both national governments (QURECA, 2021) and IT companies (including large IT companies, such as IBM, Google, Amazon or Intel, and start-ups, such as Rigetti or IonQ) have recognized the potential of QCs (Gill et al., 2022). This is, among other things, owing to the fact that QC has many applications in almost every sector (IBM, 2021; Allende López and Da Silva, 2019): security (Wallden and Kashefi, 2019), Artificial Intelligence (Zhang and Ni, 2020; Garg and Ramakrishnan, 2020), Natural Language Processing (Meichanetzidis et al., 2020); financial services (Egger et al., 2020) and economy (The Economist, 2021), chemistry (Cao et al., 2019), medicine (Cao et al., 2018), supply chains and logistics (Savoie, 2021), agriculture (Ou et al., 2022), transportation (Cooper, 2021), etc.

☆ Editor: Lingxiao Jiang.

* Corresponding author at: Faculty of Social Sciences & IT, University of Castilla-La Mancha, Avda. Real Fábrica de Seda, s/n, 45600, Talavera de la Reina, Spain.
E-mail addresses: ricardo.pdelcastillo@uclm.es (R. Pérez-Castillo), luis.jimeneznavajas@uclm.es (L. Jiménez-Navajas), mario.piattini@uclm.es (M. Piattini).

https://doi.org/10.1016/j.jss.2023.111683
0164-1212/

QC has gained a great deal of relevance in software engineering (Piattini et al., 2020a; De Stefano et al., 2022) owing to the fact that all potential applications need quantum software (Mueck, 2017). Moreover, until a few years ago, quantum algorithms for such radically different machines were coded by following ad hoc techniques (almost manually). However, the advent of quantum software engineering (Piattini et al., 2021; Zhao, 2020; Gemeinhardt et al., 2021) has led to a growing demand for quantum software to be produced in a more systematic manner while meeting quality and other budget and time constraints (Piattini et al., 2020b; aoun et al., 2021). In fact, "the lack of engineering and design methodologies for software development and verification" is one of the major barriers to the adoption of QC in industry (Awan et al., 2022).

One of the software engineering processes that has been revisited for QC is software modernization (Pérez-Castillo et al., 2021c). The evolution of quantum software must be considered as the operation in which classical and quantum software are combined through their integration into large complex systems known as hybrid software systems. The software modernization of these systems is an open challenge and needs to deal with the software evolution of hybrid systems according to certain well-known software engineering practices (automation, ease of comprehension, abstracting high-level designs (Pérez-Castillo et al., 2021a), preserving embedded business knowledge (Pérez-Castillo et al., 2021b), quality assurance (Cruz-Lemus et al., 2021), service orientation (Garcia-Alonso et al., 2022), etc.).

Of all the aforementioned open issues, this research focuses on how to extract the quantum knowledge embedded in the source code. This is essential as regards achieving abstract representations of quantum software with which to facilitate the evolution of hybrid software systems. The reverse engineering of quantum software has, to date, been addressed for gate-based QC (Pérez-Castillo et al., 2021c,b). Nevertheless, there is only a very limited amount of work regarding the reverse engineering of QA programs. This paper, therefore, presents a reverse engineering technique based on a dynamic analysis of Python code in order to extract a definition of the optimization problem (i.e., Hamiltonians). This problem is not trivial, since Hamiltonian expressions are usually dynamically built in the source code, which makes it difficult to know the exact expression at design time. Although programming languages allow to statically define optimization problems through hardcoded values, real-life problems commonly consider target optimization objectives and constraints based on external sources (e.g., files, databases, webservices). Hence, knowing the outgoing Hamiltonian being executed is hard, at least if developers use static analysis tools or they only relay on the existing documentation (which can turn out of date).

The knowledge extracted by dynamic analysis is then modeled according to the Knowledge Discovery Metamodel (KDM), which is an ISO/IEC standard (OMG, 2016). The outgoing KDM model can be integrated with other parts of larger hybrid software systems in a common, abstract, and technological-independent manner. As a result, the evolution of QA programs can be conducted in combination with other parts of hybrid software systems by following the Model-Driven Engineering (MDE) principles.

We present a multi-case study in order to show that the dynamic analysis of QA programs is effective and theoretically scalable to larger programs, which demonstrates the applicability and facilitates the adoption of the proposed solution

The remainder of the paper is structured as follows. Section 2 presents the state of the art regarding quantum computing and the software modernization of hybrid software systems. Section 3 provides a detailed explanation of the proposed technique for the dynamic analysis of QA programs. Section 4 shows the case study carried out in order to validate the proposed technique. Finally, Section 5 provides our conclusions and suggests future research.

2. Background

This section presents the theoretical and motivational background to this research. Section 2.1 depicts and illustrates how QA algorithms work and how the underlying programs are coded. Finally, Section 2.2 shows the importance of software modernization from/toward hybrid software systems.

2.1. Quantum annealing software

QA consists of a group of randomized algorithms that provide good heuristics for the solution of hard optimization problems. According to D-Wave (D-Wave, 2021), QA "outperforms other approaches such as gate model when it comes to complex optimization problems. This is because annealing avoids the significant pre-processing overhead associated with QAOA/gate-based approaches, is much more tolerant of errors and noise, and can scale to enterprise problem size".

The physics-based foundations of QA algorithms consist of the propensity of physical systems to minimize their free energy. Quantum annealing is free energy minimization in a quantum system. As stated previously, QA algorithms attempt to compute an exact or approximate solution of an optimization problem encoded as a Hamiltonian function (Calude et al., 2015). This signifies that, while universal quantum software is built through the combinations of quantum gates in quantum circuits, the effort of developing QA programs lies in the Hamiltonian definition, which groups the optimization goals and a set of constraints. Hamiltonians are usually defined and coded as Binary Quadratic Models (BQMs) in main QA vendors (e.g., D-Wave). BQMs can in turn be defined using Ising or QUBO models (the latter tend to be favored by computer scientists and software engineers, while the former are preferred by physicists) (Calude et al., 2015; Glover et al., 2018).

BQMs are unconstrained and have binary variables. Thus, BQMs are typically used for applications that optimize over decisions that could either be true or false. Apart from BQM, there exist additional models like Discrete Quadratic Models (DQM), which are also unconstrained but use discrete variables; and Constrained Quadratic Models (CQM) which can define constrains and consider real, integer and binary variables. Despite these models are alternative ways of defining optimization problems, this paper focuses on BQM (QUBO compliant) since it has been extensively used in the literature and can be used to "solve numerous categories of optimization problems including important instances of network flows, scheduling, max-cut, max-clique, vertex cover and other graph and management science problem" (Lewis and Glover, 2017).

Fig. 1 provides an example of a QA program and how it is defined using the aforementioned models. First, the QUBO matrix (a) represents the variables managed in a specific problem, which represents specific optimization goals and constraints according to the equivalent BQM (b). The underlying Hamiltonian function to be minimized is in (c). This problem is coded in a simple Python program (d) that uses a D-Wave sampler to compute an optimal solution.

Several QA computers (such as D-Wave) now exist that have achieved a certain presence in the global market, and various real applications have demonstrated their feasibility (Kahn, 2021; Dridi and Alghassi, 2017). In most cases, Hamiltonians (Ising or QUBO models) are defined manually by experts and are then maintained in the same way. Other semi-automatic approaches attempt to dynamically define Hamiltonians on the basis of constraints and variables that can evolve over time (Hevia et al., 2021). All this information is managed by means of classical code in combination with Hamiltonian expressions managed in

(a)QUBO matrix

	A	B	C	K
A	0	0	2	-2
B	0	1	-2	0
C	0	0	0	0
K	0	0	0	1

$$\min (B + K + 2 \cdot A \cdot C - 2 \cdot A \cdot K - 2 \cdot B \cdot C)$$

(b) $BQM : \{(A, C) : 2, (A, K) : -2, (B, B) : 1, (B, C) : -2, (K, K) : 1\}$

(c) $H = 0.5 \cdot b + 0.5 \cdot k + a \cdot c - a \cdot k - b \cdot c$

(d) Python (D-Wave) code

```
# Import the functions and packages that are used
from dwave.system import EmbeddingComposite, DWaveSampler
# Define the problem as a Python dictionary
Def problemDefinition():
    a = {('B','B'): 1,
         ('K','K'): 1,
         ('A','C'): 2,
         ('A','K'): -2,
         ('B','C'): -2}
return a
# Define the sampler to run the problem
sampler = EmbeddingComposite(DWaveSampler())

# Run the problem on the sampler and print the results
sampleset = sampler.sample_qubo(problemDefinition(),num_reads = 10, label='QUBO Example')
print(sampleset)
```

Fig. 1. Example of a QA program definition.

D-Wave code (D-Wave, 2023). Moreover, the underlying expressions embedded in the source code are sometimes computed dynamically by considering external information, e.g., configuration files, databases or webservices. In these cases, actual optimization objectives and constraints cannot be known at design time, which makes dynamic analysis necessary.

2.2. Modernization from/toward quantum–classical software

QC is not currently thought of as an all-in-one technology. A consequence of this is that QC could be suitable for solving certain kinds of problems, but not for addressing every single one. This is owing to the following reasons: (i) QC suffers from the same specific limitations as classical computing (Aaronson, 2008), and (ii) if the task is quite simple, there is no reason to use QC when classical computers can still perform successfully, as QC is a more expensive technology. Modern information systems will, therefore, probably integrate classical and quantum software (Pérez-Castillo et al., 2021c) in so-called hybrid software systems. These systems support some business operations implemented with classical programming languages which perform calls to quantum algorithms that can address certain otherwise intractable problems. In these hybrid systems, classical software runs in classical computers (the controller or driver part), while quantum software runs in quantum computers (the responder part) that are typically located in the cloud.

According to McCaskey et al. (2018), hybrid software systems confront obstacles regarding code portability, tool integration, program validation, and the orchestration of workflow development. Another important challenge is the modernization of hybrid information systems. In Pérez-Castillo et al. (2021c), reengineering and software modernization is revisited in order to deal with the problems associated with the expected quantum computing migrations and the upcoming coexistence of classical and quantum software. This work states the need for the reverse engineering of quantum programs (Pérez-Castillo et al., 2021b), the integration of information (from quantum and classical software) into a common representation, automatic transformations (Jiménez-Navajas et al., 2021), and a high-level design for the target hybrid systems (Pérez-Castillo et al., 2021a). In this context, the problem of modeling quantum programs has been addressed in Ali and Yue (2020). All these related works focus on gate-based quantum software, while QA programs have not been considered in the context of the modernization of hybrid software systems.

With regard to QA programs, the process of defining QA optimizing problems (cf. Section 2.1) poses two interesting challenges for quantum software evolution:

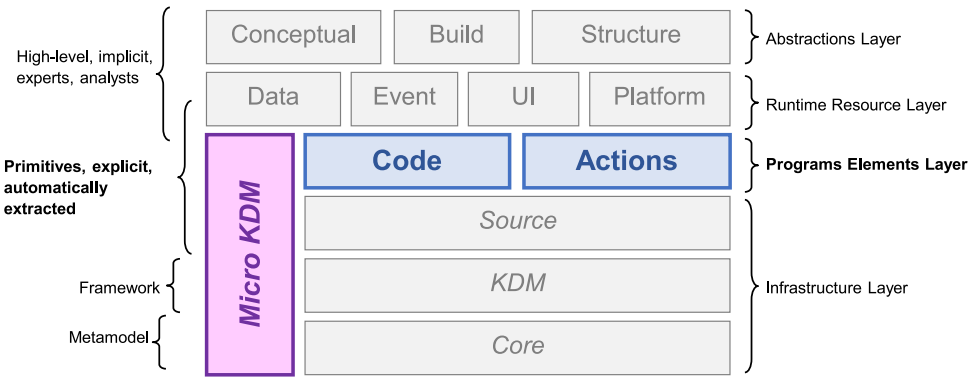

Fig. 2. Layers, packages, and concerns in KDM.

- Once the code supporting the Hamiltonian definition evolves, it is difficult to know the exact expression for the optimization problem, and it will, therefore, require a certain manual effort to comprehend only the outgoing expression and how this expression is programmatically built. The software maintenance of these Hamiltonian expressions is consequently a complex task.
- Even if the Hamiltonian expressions are clear, QA algorithms can be integrated into larger hybrid software systems. The evolution of QA algorithms cannot, therefore, take place in isolation, since some data managed by the classical software can be used dynamically to compute certain constraints or variables related to the Hamiltonian definitions in code, QUBO, Ising, etc.

With regard to the first challenge, despite the fact that there is some preliminary research concerning the reverse engineering of gate-based QC (Pérez-Castillo et al., 2021c,b), there is limited work regarding the reverse engineering of Hamiltonian expressions in QA. Tools such as *PyQUBO* (L. Recruit Communications Co., 2023) automatically transform different Hamiltonian representations among each other (e.g., BQM, QUBO or Ising). Unfortunately, these transformations do not support the reverse engineering of the hybrid (classical–quantum) code required in order to gather Hamiltonians. However, a Hamiltonian open quantum system toolkit (HOQST) was proposed in Chen and Lidar (2022), which integrates a simulator and other tools in order to investigate the open quantum system dynamics in Hamiltonian quantum computing. Despite the value of this toolkit, it does not specifically cover the reverse engineering of Hamiltonian expressions. In Kang et al. (2016), there are some mathematical methods with which to simplify the Hamiltonian or to generate it from other knowledge, although this software engineering technique cannot be employed to directly manage quantum software.

With regard to the second challenge, this paper advocates the use of KDM (OMG, 2016), as has occurred in other previous works (Pérez-Castillo et al., 2021b; Jiménez-Navajas et al., 2021). This standard enables the application of the MDE (Model-Driven Engineering) principles to quantum software engineering, as stated by several authors (Gemeinhardt et al., 2021; Ali and Yue, 2020). Nonetheless, KDM has not been used to represent Hamiltonian expressions to date. KDM is a standard metamodel framed in the Architecture-Driven Modernization (ADM) initiative (OMG, 2021). ADM is a software evolution paradigm that considers traditional reengineering processes by following the MDE principles. In this context, the KDM metamodel "is used to represent all the software artifacts involved (i.e., source code, databases, user interactions, etc.), and it achieves this in an integrated and technological-independent manner" (Pérez-Castillo et al., 2021c). Once it has been applied to hybrid software systems, it is possible to have a common KDM repository that is gradually populated with knowledge discovered through the analysis of different artifacts in combination, e.g., QA programs, quantum circuits and classical software. As a result, KDM can be used as an abstraction mechanism with which to represent hybrid information systems as a whole, and not represent only their source code.

The KDM metamodel is divided into four layers, which represent both the physical and logical software artifacts of information systems at several abstraction levels (Pérez-Castillo et al., 2011). Each layer is organized into packages that define a set of metamodel elements; the purpose of these packages is to represent a specific independent facet of knowledge related to LIS as architectural viewpoints (see Fig. 2, in which the highlighted boxes are the KDM packages considered within the scope of this research (cf. Section 3.3)).

3. Dynamic analysis of quantum annealing programs

The goal of this research is to deal with the two challenges described previously. The purpose of the dynamic analysis of QA programs is, therefore, twofold:

- To provide a reverse engineering technique based on the dynamic analysis of Python code (D-Wave Ocean SDK) that is able to abstract and generate the underlying Hamiltonian expressions. The abstract representation can, therefore, contribute to the software maintenance and evolution of hybrid information systems that include QA programs.
- To represent the Hamiltonian expressions according to the KDM standard (OMG, 2016). The outgoing KDM model can be integrated with other parts of larger hybrid information systems in a common abstract representation based on that standard. The evolution of QA programs can, therefore, be conducted in combination with other parts of hybrid information systems without having to pay attention to the specific technological details.

The overall process of the proposed dynamic analysis technique is summarized in Fig. 3. First, some traces must be added to the Python code in order to enable the execution traces to be gathered during program execution (cf. Section 3.1). The dynamic analysis script developed then analyzes the execution traces and considers some of the main D-Wave Ocean SDK samplers in order to generate the Hamiltonian expressions (cf. Section 3.2). Finally, the KDM model is generated from the Hamiltonian expression (cf. Section 3.3). The whole script can be accessed in GitHub (Pérez-Castillo, 2022b).

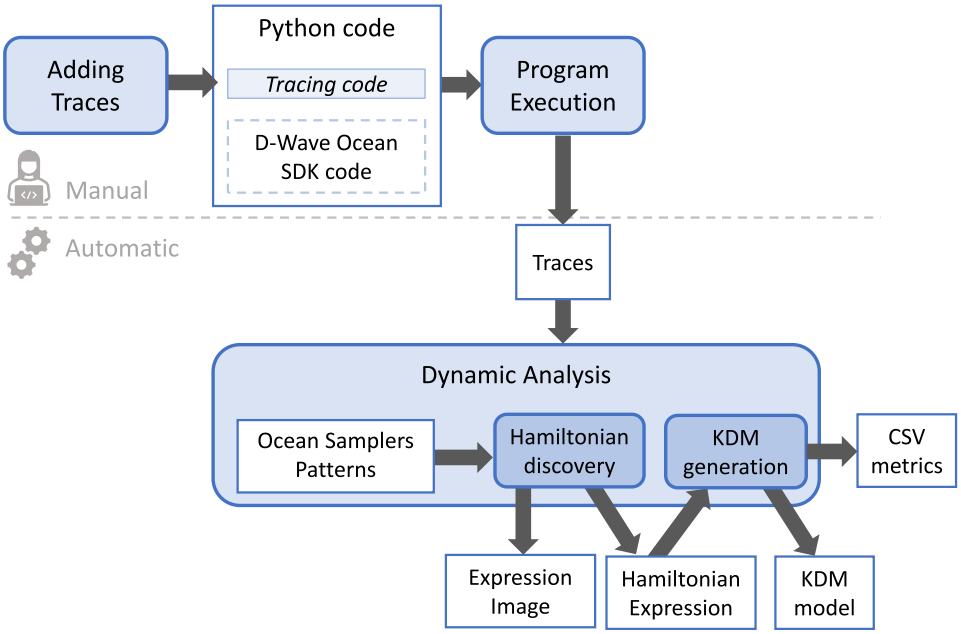

Fig. 3. Dynamic analysis technique process for QA programs.

```
''' Lines for dynamic analysis of dwave functions '''
import sys
from dwave_reverse.DwaveReverse import DwaveReverse
sys.settrace(DwaveReverse.traceit)
```

Fig. 4. Statements with which to enable dynamic analysis.

3.1. Tracing python code

The first step is the addition of the tracing code to the existing Python code in order to enable the collection of information during execution. This task consists of manually adding the lines shown in Fig. 4. The class *dwave_reverse. DwaveReverse* contains all the Python code that supports the dynamic analysis.

The Python tracing infrastructure is based on the function *sys.settrace()*. This capability facilitates the setting up of the system's trace function, which allows the implementation of Python source code debuggers. The *settrace()* function is intended only for the implementation of debuggers, profilers or coverage tools. Its behavior is part of the Python implementation platform, rather than part of the language definition, signifying that it may not be available in all Python implementations. This function is thread-specific and must register a trace function, which will be executed for every code statement. In our proposal, the function developed has been denominated as a *'traceit'*, as defined in the *DwaveReverse* class. The strategy used for the dynamic analysis forces the addition of tracing code within Python code. This may prevent this technique from being used in some cases owing to, for example, teams being unwilling to modify their source code. Despite this minor drawback, we believe that this strategy (based on the standard Python tracing mechanism) provides the greatest amount of useful information during the execution of QA programs. We additionally consider that this manual intervention is minimally invasive, since it is necessary only to add two import clauses and activate the built-in tracing mechanism of Python. The main source code of QA programs, therefore, remains immutable.

3.2. Reverse engineering of Hamiltonians

Having dynamically collected the traces for all the statement, the first point is to filter in only the call statements, and then those source code lines with invocations to D-Wave functions. After that, the complexity of the reverse engineering technique lays on the determination of the different D-Wave solvers' patterns and problems definitions from the executed Python code. The collected information is then used to abstract the respective BQM problem definition (also as a QUBO expression), which is eventually transformed into a Hamiltonian expression.

The reverse engineering employed in order to discover Hamiltonian expressions is supported by the *traceit* function that is used as a parameter in *sys.settrace*. The *traceit* function must be defined with at least three parameters (see Fig. 5):

- ***Frame.*** This is the current stack frame, i.e., the context of the QA program execution.
- ***Event***. This is a string representing the type of event that has been traced. These types can be: *call*, *line*, *return*, *exception*, or *opcode*. This technique operates when the 'call' event is triggered, and in particular, when a function in the D-Wave sampler is called.
- ***Arg.*** This depends on the event type and contains further information.

The *traceit* function intuitively attempts to recognize calls (see line 8) to different D-Wave samplers (see lines 11 to 22) and, depending on this function, it will gather the respective information required to build the Hamiltonian. A sampler or solver in D-Wave Ocean SDK is each of the specific classes that accept problem definitions (BQM, QUBO, Ising models) and compute solutions by following different strategies. Some of the commonly used Ocean

```
1 @staticmethod
2 def traceit(frame, event, arg):
3     function_code = frame.f_code
4     function_name = function_code.co_name
5     lineno = frame.f_lineno
6     vars = frame.f_locals
7
8     if event == 'call':
9         QUBO = None
10        type = None
11        if frame.f_code.co_name in ('sample',) and vars['self'].__class__.__module__ in
          ("dwave.system.composites.embedding", "dwave.system.samplers.leap_hybrid_sampler",
          "neal.sampler", "hybrid.reference.kerberos", "hybrid.core"):
12            type = 'QUBO'
13            time_start = timer()
14            QUBO = DwaveReverse.getQUBO(DwaveReverse.getQUBOfromBQM, vars)
15        elif frame.f_code.co_name in ('sample_qubo',) and
          vars['self'].__class__.__module__.__contains__("dwave"):
16            type = 'TupleWithoutOffset'
17            time_start = timer()
18            QUBO = DwaveReverse.getQUBO(DwaveReverse.getQUBOfromTupleWithoutOffset, vars)
19        elif frame.f_code.co_name in ('sample_ising',) and
          vars['self'].__class__.__module__.__contains__("dwave"):
20            type = 'Ising'
21            time_start = timer()
22            QUBO = DwaveReverse.getQUBO(DwaveReverse.getQUBOfromIsing, vars)
23
24        if QUBO isnotNone:
25            class_name = vars['self'].__class__.__qualname__ if 'self' invars.keys() else ''
26            function_line = str(frame.f_lineno)
27            trace_name = class_name + "." + function_name + "." + function_line + "_" +
              datetime.now().strftime("%Y%m%d_%H%M%S")
28
29            H, variables, coefficients = DwaveReverse.quboToH(QUBO)
30
31            time = timer()
32            DwaveReverse.generateHinTextFile(H, trace_name)
33            time_H_text = timer() - time
34
35            time = timer()
36            if len(coefficients) <150:
37                Hm = DwaveReverse.generateHinImage(H, trace_name)
38            time_H_image = timer() - time
39
40            time_H_total = timer() - time_start
41
42            time = timer()
43            KDMGenerator().generateKDM(H, class_name, function_name, function_line)
44            time_KDM = timer() - time
45
46            DwaveReverse.saveMetrics(ntpath.basename(sys.argv[0]), [class_name, function_name,
              function_line, type, len(variables), len(coefficients), time_H_text, time_H_image,
              time_H_total, time_KDM])
47
48    return DwaveReverse.traceit
```

Fig. 5. Python function with which to dynamically trace QA programs.

samplers are *DWaveSampler*, *ExactSolver*, *SimulatedAnnealingSampler*, *LeapHybridSampler*, among others.

The *traceit* function (see lines 11, 15 and 19 in Fig. 5) then recognizes various D-Wave functions, but could be extended in the future to recognize additional samplers or solvers. Although there are more D-Wave solvers that are not considered in these lines, this illustrates how to gather information from three different problem definitions: BQM, tuple (with no offset), and Ising model. Other problem definition models such as DQM or CQM are outside of the scope of this research since we consider BQM as the most widely used model in the literature (Lewis and Glover, 2017). Despite this design decision, this technique could be easily extended in the future to support those alternative models. The *traceit* function consequently translates any problem definition into QUBO, and this is eventually transformed into the Hamiltonian expression (see lines 24 to 46). In these lines, the Hamiltonian expression is generated in text and image mode, and the KDM generation function is also called (cf. Section 3.3). Time and other relevant measures are taken in those lines to be written in a csv file in line 48 (see Fig. 5).

The transformation from the QUBO to a Hamiltonian is almost direct (see Fig. 6). The *quboToH* function simply takes coefficients from each pair, isolated variables and composes the expression. It should also detect the coefficient if it is the linear term of the polynomial (see lines 19 to 25 in Fig. 6) or the quadratic term (see lines 26 to 30). After computing the raw polynomial, it is simplified in line 41 with the *simplify* function at *sympy*, an external module. The use of this simplification signifies that some special characters must be avoided, which is done by using various regex-based replacements (see for example lines 17, 21, 29 and 30 in Fig. 6). The output of this function is a tuple with three elements: (i) the Hamiltonian expression as a string; (ii) the list of the involved variables; and (iii) the list of coefficients (see line 45.

```
@staticmethod
def quboToH(Q):
    H = ''
    coefficients = []
    variables = set()
    first = True
    for pair in Q[0]:
        if Q[0][pair] == 0:
            continue
        if not first and Q[0][pair] >0:
            H += '+'
        first = False

        coeff = Q[0][pair]/2
        coefficients.append("{:.2f}".format(coeff))
        H += ("{:.2f}".format(coeff) + "*")
        regex = r"[\(|\)|\[|\]|\{|\}| ]"

        if pair[0] == pair[1]:
            var = 'a' + (str(pair[0]).lower() if isinstance(pair[0], str) else str(pair[0]))
            var = re.sub(regex, "", var.replace(",", "").replace("_", ""))
            symbols(var)
            variables.add(var)
            H += '(1-'+var+')'
        else:
            var1 = 'a' + (str(pair[0]).lower() if isinstance(pair[0], str) else str(pair[0]))
            var2 = 'a' + (str(pair[1]).lower() if isinstance(pair[1], str) else str(pair[1]))
            var1 = re.sub(regex, "", var1.replace(",", "").replace("_", ""))
            var2 = re.sub(regex, "", var2.replace(",", "").replace("_", ""))

            symbols(var1)
            symbols(var2)
            variables.add(var1)
            variables.add(var2)

            H += ('(1-'+var1+')*(1-' + var2+')')

    H += (("+" + "{:.2f}".format(Q[1])) if Q[1] >0 else "")
    try:
        H=simplify(H)
    except:
        print('Error in simplification!')
    H = 'H = ' + str(H)
    return H, variables, coefficients;
```

Fig. 6. Function employed to transform QUBO into a Hamiltonian.

```
def getQUBOfromIsing (vars):
    h = vars['h']
    J = vars['J']
    QUBO = utilities.ising_to_qubo(h, J, 0);
    return QUBO
```

Fig. 7. Function employed to obtain a QUBO from Ising model.

Before transforming QUBO into a Hamiltonian, the major contribution lies in the collection of information from different problem definitions (see lines 14, 18, 22 and 26 in Fig. 5), i.e., the gathering of QUBO definitions by means of the *getQUBO* function. The proposal defines *getQUBO* as an overloaded function that accepts as parameters a specific function plus the traced information (*vars*), which varies depending on the D-Wave sampler.

As an example, Fig. 7 shows the *getQUBOfromIsing* function that is used as a parameter in line 14 (see Fig. 5). Functions passed as parameters of the *getQUBO* function employ, if possible, some D-Wave utility functions to perform a transformation. In the example in Fig. 7, the function employs *utilities.ising_to_qubo* to generate the QUBO model.

3.3. Generation of KDM models

Once the Hamiltonian expression has been built, this information is used to generate a KDM model (see Fig. 5). This function is defined in the 'KDMGenerator' class (see the whole script in Pérez-Castillo (2022b)). A KDM model is stored as an *.XMI file, signifying that the script is simply devoted to building an XML file containing the information required to represent the Hamiltonian expression previously built.

Fig. 8 provides an excerpt of the *generateKDM* function to create the KDM model according to the KDM specification. The root element is a *Segment* element, which is created in line 28, which then defines a *CodeModel*. A *CodeModel* represents the source python file analyzed, which could define several optimization problems, i.e., various Hamiltonians. Thus, the script defines a *CodeAssembly* plus a *CallableUnit* for each reversed Hamiltonian (see lines 39–40). Then, the left part of the Hamiltonian ('H = ') is defined as an expression into the *CallableUnit*, by using an *ActionElement* (see line 46).

Having defined the overall schema for the KDM model, the script then tokenizes the Hamiltonian expression through the sums and by means of the terms involved in each sum (those terms use variables and coefficients). Thus, sums are represented in KDM as *ActionElement*, variables with *StorableUnit*, and coefficients with *Value* elements according to the KDM specification. This part of the script can be accessed in Pérez-Castillo (2022b).

To continue with the running example, Fig. 9 shows the KDM model that is built from the Hamiltonian example introduced in

```
1  def generateKDM (self, H, class_name, function_name, function_line):
2      print (H)
3      self.resetId()
4      if not os.path.isdir(KDMGenerator.KDM_FOLDER):
5          os.makedirs(KDMGenerator.KDM_FOLDER)
6      file_path = KDMGenerator.KDM_FOLDER + "/" + ntpath.basename(sys.argv[0]) + ".kdm"
7      ns = {
8          'xmi': 'http://www.omg.org/spec/XMI/20110701',
9          'action': 'http://www.omg.org/spec/KDM/20160201/action',
10         'code': 'http://www.omg.org/spec/KDM/20160201/code',
11     }
12     for prefix, uri in ns.items():
13         ET.register_namespace(prefix, uri)
14     ET.register_namespace('kdm', 'http://www.omg.org/spec/KDM/20160201/kdm')
15     segment = None
16     codeAsembly = None
17     if os.path.exists(file_path):
18         lastModTime = datetime.fromtimestamp(os.path.getmtime(file_path))
19         target_date = datetime.now() - timedelta(minutes=15)
20         if lastModTime<target_date:
21             print('>>>deleting file')
22             os.remove(file_path)
23     if os.path.exists(file_path):
24         tree = ET.parse(file_path)
25         root = tree.getroot()
26         codeAsembly = root[0][0]
27     else:
28         segment = ET.fromstring('<kdm:Segment
           xmlns:kdm="http://www.omg.org/spec/KDM/20160201/kdm"></kdm:Segment>')
29         tree = ET.ElementTree(segment)
30         segment.set('name', class_name + "" + datetime.now().strftime("%Y%m%d_%H%M%S"))
31         for prefix, uri in ns.items():
32             segment.set('xmlns:' + prefix, uri)
33         eId = self.getId();
34         codeModel = ET.SubElement(segment, "model", {'xmi:id':eId, 'xmi:type':'code:CodeModel'})
35         self.elementsMap[codeModel] = eId
36         eId = self.getId();
37         codeAsembly = ET.SubElement(codeModel, "codeElement", {'xmi:id':'id.' + eId,
           'xmi:type':'code:CodeAsembly'})
38         self.elementsMap[codeAsembly] = eId
39     eId = self.getId();
40     callableUnit = ET.SubElement(codeAsembly, "codeElement", {'xmi:id':eId,
       'xmi:type':'code:CallableUnit', 'kind':'regular', 'name':class_name + "." + function_name + "."
       + function_line})
41     self.elementsMap[callableUnit] = eId
42     eId = self.getId();
43     entryFlow = ET.SubElement(callableUnit, "entryFlow", {'xmi:id':eId, 'from':eId})
44     self.elementsMap[entryFlow] = eId
45     eId = self.getId();
46     actionElement = ET.SubElement(callableUnit, "codeElement", {'xmi:id':eId,
       'xmi:type':'action:ActionElement', 'kind':'compound', 'name':'H'})
47     self.elementsMap[actionElement] = eId
48     entryFlow.set('to', self.getElementId(actionElement))
49     eId = self.getId();
50     source = ET.SubElement(actionElement, "source", {'xmi:id':eId, 'snippet':str(H)})
51     self.elementsMap[source] = eId
```

Fig. 8. Function employed to generate a KDM model to represent Hamiltonians.

Fig. 1. Since the Hamiltonian expression is related to a call to one of the D-Wave samplers or solvers, a *CallableUnit* element is created in order to represent the Hamiltonian (see line 4 in Fig. 9). This callable unit could, therefore, be referenced from other elements in the system (classical or quantum parts) represented in the KDM model. The callable unit is created within a *CodeAsembly*, which is in turn created within a *CodeModel* element (see lines 2 and 3). These represent the Python file analyzed and could group various Hamiltonian expressions. The code model can be integrated into a KDM repository and establish relationships with other parts of larger systems.

The Hamiltonian expression is then represented in the subsequent lines in the KDM model (see Fig. 9). The formal representation of mathematical expressions is not the intended purpose of KDM. However, it is able to represent this kind of expressions, since it can be coded in information systems according to most of the programming languages. The Hamiltonian expressions are, therefore, represented in KDM by using *Code* and *Action* packages (see program elements layer in Fig. 2) and the *Micro KDM* package (see bottom-left part of Fig. 2). According to KDM, macro actions as defined in *Code* and *Action* packages are treated as a container that owns certain "micro actions" with predefined semantics, and precise semantics of the "macro action" is, therefore, defined. Micro KDM actions are aligned with the ISO/IEC 11 404 datatypes (operations on primitive datatypes and access to complex datatypes).

A Hamiltonian expression can be viewed as a sequence of summands that combine the variables with a specific coefficient. This sequence is, therefore, represented as a set of sums, and each sum as a set of multiplications. Multiplications are defined as a *CodeElement* using the micro action *kind* = *"Multiply"* (see lines 8 and 58 in Fig. 9). Sums are represented in an accumulative manner (see line 65 in Fig. 9). These are, therefore, also *CodeElements* with the micro action *kind* = *"Add"* or *"Subtract"* (depending on the coefficient).

```
<kdm:Segment xmlns:kdm="http://www.omg.org/spec/KDM/20160201/kdm"
xmlns:action="http://www.omg.org/spec/KDM/20160201/action"
xmlns:code="http://www.omg.org/spec/KDM/20160201/code"
xmlns:xmi="http://www.omg.org/spec/XMI/20110701"
name="EmbeddingComposite.sample_qubo.266">
  <model xmi:id="id.1" xmi:type="code:CodeModel">
    <codeElement xmi:id="id.id.2" xmi:type="code:CodeAsembly">
      <codeElement kind="regular" name="DWAVE_FUNCTION" xmi:id="id.3"
      xmi:type="code:CallableUnit">
        <entryFlow from="id.4"to="id.5" xmi:id="id.4"/>
        <codeElement kind="compound" name="H" xmi:id="id.5" xmi:type="action:ActionElement">
          <source snippet="H=1*b+1*k+2*a*c-2*a*k-2*b*c" xmi:id="id.6"/>
          <codeElement kind="Multiply" name="1*b" xmi:id="id.7" xmi:type="action:ActionElement">
            <codeElement from="id.7"to="id.8"xmi:id="id.9" xmi:type="action:Reads"/>
            <codeElement from="id.7"to="id.10"xmi:id="id.11" xmi:type="action:Reads"/>
            <codeElement from="id.7"to="id.12"xmi:id="id.13" xmi:type="action:Writes"/>
          </codeElement>
          <codeElement name="1" xmi:id="id.8" xmi:type="code:Value"/>
          <codeElementname="b" xmi:id="id.10" xmi:type="code:StorableUnit"/>
          <codeElement kind="register" name="multiply_1*b "xmi:id="id.12"
          xmi:type="code:StorableUnit"/>
          . . .
          <codeElement kind="Multiply" name="2*b*c" xmi:id="id.50"
          xmi:type="action:ActionElement">
            <codeElement from="id.50" to="id.26" xmi:id="id.51" xmi:type="action:Reads"/>
            <codeElement from="id.50" to="id.10" xmi:id="id.52" xmi:type="action:Reads"/>
            <codeElement from="id.50" to="id.30" xmi:id="id.53" xmi:type="action:Reads"/>
            <codeElement from="id.50" to="id.54" xmi:id="id.55" xmi:type="action:Writes"/>
          </codeElement>
          <codeElement kind="register" name="multiply_2*b*c" xmi:id="id.54"
          xmi:type="code:StorableUnit"/>
          <codeElement kind="Subtract" name="+1*b+1*k+2*a*c-2*a*k-2*b*c" xmi:id="id.56"
          xmi:type="action:ActionElement">
            <codeElement from="id.56" to="id.48" xmi:id="id.57" xmi:type="action:Reads"/>
            <codeElement from="id.56" to="id.54" xmi:id="id.58" xmi:type="action:Reads"/>
            <codeElement from="id.56" to="id.59" xmi:id="id.60" xmi:type="action:Writes"/>
          </codeElement>
          <codeElement kind="register" name="accumulated_+1*b+1*k+2*a*c-2*a*k-2*b*c"
          xmi:id="id.59" xmi:type="code:StorableUnit"/>
        </codeElement>
      </codeElement>
    </codeElement>
  </model>
</kdm:Segment>
```

Fig. 9. Example of KDM model for the Hamiltonian expression in Fig. 1.

4. Case study

This section presents a detailed multi case study. The term case study is currently a misleading and misused term in software engineering. According to Wohlin (Wohlin, 2021), a case study is *"an empirical investigation of a case, using multiple data collection methods, to study a contemporary phenomenon in its real-life context, and with the investigator(s) not taking an active role in the case investigated"*. Thus, the *contemporary phenomenon* and the rationale behind this study is, therefore, to demonstrate the feasibility of the proposed dynamic analysis technique as regards retrieving and abstracting Hamiltonian expressions from QA programs. The real-life context refers to real-life Python QA programs available in the D-Wave repository. Finally, researchers are not directly involved in the study as occurs in action research.

This section is structured as follows: Section 4.1 explains all the details of the case study design, which is necessary in order to ensure a certain rigor and facilitates its replication. Section 4.2 then provides an analysis of the results of the study, along with an interpretation of those results. Finally, Section 4.3 evaluates the validity of the study.

4.1. Case study design

The case study was designed and conducted by following the protocol proposed by Runeson et al. (2012). The following subsections are structured according to that protocol.

4.1.1. Rationale and objectives

The object of the study is a dynamic analysis technique with which to generate Hamiltonian expressions and its representation in KDM models from quantum annealing programs. The purpose is, meanwhile, to evaluate its effectiveness and efficiency with real-life quantum annealing programs.

The design of the case study is, according to Yin's classification (Yin, 2013), an embedded multi-case study. It is multi-case since it considers several quantum programs, which are treated as different cases. Some programs could, therefore, consider more than one Hamiltonian expression, and these expressions are consequently considered to be the unit of analysis in this study.

4.1.2. Research questions and hypotheses

Having defined the object and purpose of the study, we now present the aspects with which to evaluate effectiveness. First, RQ1 attempts to answer whether the Hamiltonian expressions can be fully and accurately retrieved from QA programs. Second, RQ2 focuses on assessing whether the KDM models generated as an abstract representation of Hamiltonians are built consistently, and finally, RQ3 analyzes whether the dynamic analysis technique (for both the retrieval of the Hamiltonians and the generation of the KDM models) is scalable to different QA programs and larger Hamiltonian expressions.

RQ1. Are Hamiltonian expressions effectively retrieved from the execution of quantum annealing programs?

Table 1
Variables definition.

Concept	Measure	Scope	Definition	Range
Effectiveness	Hamiltonian generated?	Unit of Analysis	Is the H. expression retrieved?	x = {0,1}
	Valid Hamiltonian?	Unit of Analysis	Is the H. expression accurate?	x = {0,1}
	KDM generated?	Unit of Analysis	Is the KDM model generated?	x = {0,1}
	Valid KDM model?	Unit of Analysis	Is the KDM model valid and compliant?	x = {0,1}
Efficiency	LoC	Case	Number of lines of code in the QA program	$x \in \mathbb{N}^+$
	#coefficients	Unit of Analysis	Number of coefficients in the H. expression	$x \in \mathbb{N}^+$
	#variables	Unit of Analysis	Number of variables in the H. expression	$x \in \mathbb{N}^+$
	Problem type	Unit of Analysis	How the optimization problem is defined in the QA program	X = {BQM, Ising, tuple}
	Hamiltonian generation time	Unit of Analysis	Time spent retrieving each H. expression	$x \in \mathbb{R}^+$
	KDM generation time	Case	Time spent generating the KDM model	$x \in \mathbb{R}^+$
	QPU Time	Unit of Analysis	Access time required for Quantum Processing Unit.	$x \in \mathbb{R}^+$
	Service Time	Unit of Analysis	The total time to pass through the D-Wave system.	$x \in \mathbb{R}^+$

RQ2. Are the extracted Hamiltonian expressions represented in KDM models with adequate accuracy and completion levels?

RQ3. Is the dynamic analysis of quantum annealing programs scalable?

4.1.3. Concepts and measures

There are two main concepts to be assessed: effectiveness and efficiency. Effectiveness is associated with RQ1 and RQ2, while efficiency serves to evaluate RQ3. We define several measures for each concept.

With regard to effectiveness, in the case of retrieving Hamiltonian expressions, this means that the expressions are fully represented and represent what is actually dynamically composed in the quantum annealing programs. With regard to KDM model generations, effectiveness is more closely related to the completeness of the Hamiltonian representation according to the KDM standard, and full compliance with it.

Table 1 shows all the variables defined for the case study. The measures associated with this concept are four Boolean variables determining: (i) whether the Hamiltonian expression was generated; (ii) whether those expressions actually represent what is embedded in the QA program; (iii) whether the KDM model was generated, and (iv) whether the model file represents a valid KDM. The scope for these variables is the specific QA problem, i.e., for each Hamiltonian that is coded or used in a QA program file. In order to discover whether or not a Hamiltonian is valid, the reversed expression is used to code an equivalent program that is then executed. The execution results of the original and generated programs are then compared in order to verify that both solutions are equivalent, and the Hamiltonian can, therefore, be considered valid.

With regard to the efficiency, this is related to the time spent on different tasks involved in the dynamic analysis of source code and the generation of the KDM models. First, in order to evaluate the scalability, some independent variables are used to quantify the size: the number of lines of source code in a QA program, along with the number of coefficients and variables involved in a Hamiltonian expression (see Table 1). The type of problem definition is also registered in case it affects the time variables. Four different timing measures are then collected (see Table 1): (i) the time spent retrieving the Hamiltonian expression; (ii) the time required to generate the KDM model (which is unique for each QA program); (iii) the QPU time spent resolving each QA program; and (iv) service time, i.e., the whole time spent on D-Wave machines which, apart from the QPU time, includes preparation, QPU queue, post-processing, among others. These two times are registered in order to compute the actual time spent by the dynamic analysis technique under study. The internet latency is not within the scope of this study, since it can vary for different executions.

4.1.4. Case selection

The case selection consists of the application of four criteria, SC1 to SC4. SC1 selects functional D-Wave QA programs. This is defined in order to consider only functional QA programs specially produced for D-Wave platform. In order to achieve this, a systematic search was performed in the open repository of D-Wave examples at https://github.com/dwave-examples. SC2 selects QA programs coded in Python in an attempt to filter out some examples written as interactive *Jupyter* notebooks and selects only QA programs coded in Python. SC3 considers optimization problems defined as Binary Quadratic Models (BQM). It filters in those programs whose optimization problems are defined as Binary Quadratic Models (BQMs), since the scope of the proposed dynamic analysis technique covers only this. Alternative problem definitions, such as Constrained Quadratic Models (CQMs) or Discrete Quadratic Models (DQMs), have been excluded from this case study. Finally, SC4 selects programs without supporting experiments, signifying that programs that support the execution of multiple QA programs that are automatically built based on information in different configuration files are discarded. These kinds of programs are used for experimentation, but generate multiple artificial QA programs that do not represent real programs. The result of applying the selection criteria provided us with a total of 27 QA programs. Table 2 lists the cases selected with (i) the identifier; (ii) project name (when added to the D-Wave examples URL, this provides the project's URL in which a brief description can be found); (iii) the main programming file(s) dynamically analyzed, and (iv) the problem definition type.

4.1.5. Data collection procedure

The overall data collection procedure consists of three main phases with various steps (see Fig. 10). The main phase consists of performing the dynamic analysis for the cases under study, and basically has the following main steps: (i) cloning/downloading the QA programs from the D-Wave examples repository in GitHub (check final repository (Pérez-Castillo, 2022a)); (ii) adding trace statements, as depicted in Section 3, and (iii) the execution of the QA programs. Various data artifacts are built as a result of this first phase:

- A Hamiltonian expression PNG file. This is a graphical image containing the mathematical expression.
- A Hamiltonian expression TXT file. This is the version of the Hamiltonian expression in plain text, which can be used in the following phases.
- A KDM file. This is the KDM model generated according to the proposed technique.
- A Metrics CSV file. For this case study, the Python scripts developed were modified in order to generate some useful metrics, such as the number of variables or coefficients, and

Table 2
QA programs under study.

ID	Project name	Program file	P. type
C1		Basic_Programs.general_program_qubo.py	Tuple
C2		Basic_Programs.general_program_ising.py	Ising
C3	simple-ocean-programs	BQM_Functionality.bqm_conversion.py	Ising
C4		BQM_Functionality.bqm_offsets.py	BQM
C5		BQM_Functionality.general_program_bqm.py	BQM
C6	maximum-cut	maximum_cut.py	Tuple
C7		maximum_cut_ising.py	Ising
C8	nurse-scheduling	nurse_scheduling.py	BQM
C9	satellite-placement	satellite.py	BQM
C10	job-shop scheduling	demo.py	BQM
C11	antenna-selection	antennas.py	BQM
C12	mutual-information-feature-selection	titanic.py	BQM
C13	ev-charger-placement	demo.py	BQM
C14	maze	demo.py	BQM
C15	factoring	demo.py	BQM
C16	clustering	clustering.py	BQM
C17	circuit-fault-diagnosis	demo.py	ISING
C18	Graph Partitioning	graph_partitioning.py	tuple
C19	Map coloring	map_coloring.py	BQM
C20	Pipelines	pipelines.py	BQM
C21	n-queens	n_queens.py	BQM
C22	sudoku	sudoku.py	BQM
C23	structural-imbalance	structural_imbalance.py	BQM
C24	reservoir-management	demo.py	BQM
C25	Line-up-optimization	lineup_optim.py	BQM
C26	qboost (blobs)	demo.py	BQM
C27	frequency-selection	frequency.py	BQM

various timing metrics. The metrics generated for each program are then integrated into the global case study dataset.
- Job information in D-Wave Leap. This refers to the information automatically generated after executing QA programs in D-Wave hardware and stored in the D-Wave Leap page. This information is useful for gathering execution time metrics.
- A D-Wave problem inspector HTML file. This is the HTML file automatically generated by D-Wave (on demand) and is used to show the detailed graphical results of QA programs. This is useful for the subsequent comparison of results.

In the second phase, we performed a round-trip execution with the Hamiltonian expressions discovered in order to answer research question RQ2, which concerns the effectiveness of the proposed technique. In the first step, the reversed Hamiltonian was, therefore, used to automatically build a QA program by employing a Python script that we had developed. In the second step, the program was executed again, and the results were captured in a new D-Wave problem inspector HTML file (see the snapshot example in Fig. 11). This was then compared with the previous one in order to verify whether the results were the same. For this comparison, the outgoing solutions obtained for both executions were compared. These solutions are reported in different ways depending on the optimization problem, and how it is coded.

The last phase concerns the collection and integration of all the relevant information (see Fig. 10). This phase consists of three main tasks:

- All the metrics in the CSV files, generated during dynamic analysis, are integrated into the common case study dataset.
- The execution jobs in D-Wave Leap are inspected in order to collect all the execution times. These are time metrics related to the D-Wave systems and help discard this time regarding the time really spent on dynamic analysis.
- The results of the round-trip executions in the D-Wave problem inspector are compared with the original results.

This results in the generation of a whole dataset for the case, which can be accessed in (Pérez-Castillo et al., 2022). A summary of this dataset is provided in Table 3, which shows data according to the variables defined in Table 1.

4.1.6. Data analysis

The data analysis process employs two different types of analyses to answer the research questions.

- Descriptive statistics (using numerical and graphical methods) are first used as a qualitative analysis for '*explanation building*' (Yin, 2014). With regard to RQ1 and RQ2, the associated measures and their aggregated values are quantitatively assessed. In the case of RQ3, descriptive statistics are preliminarily used to check the scalability. The relationship between size and generation time variables is, therefore, examined using a scatterplot.
- Regression model: The relationships among the variables are computed by means of this analysis. This is used only to answer RQ3. The linear regression model considers the generation time measures previously depicted as the dependent variable and the size measure as the independent variable. The correlation coefficient ρ (which is between -1 and 1) is computed by applying Pearson's rank correlation test. Pearson's rho (ρ) is the degree to which the actual values of the dependent variable are near to the predicted values. It should be noticed that a linear relationship reported by the correlation test is not enough to ensure the scalability. The scalability must be judged through the outgoing equation in combination. This is owing to the fact that, for example, the time could grow up fast for larger programs even if the relationship is linear, which will not ensure a practical scalability.

4.1.7. Quality control and assurance

We have considered three mechanisms with which to ensure the appropriate level of quality in this case study:

- Some external peers were asked to review a draft of the case study design.
- We conducted a pilot case study to assess a preliminary case study design. The pilot study analyzed 9 cases: programs C1to C9. This pilot study was used to assess whether the data collection and analysis procedures worked as expected according to the preliminary design.

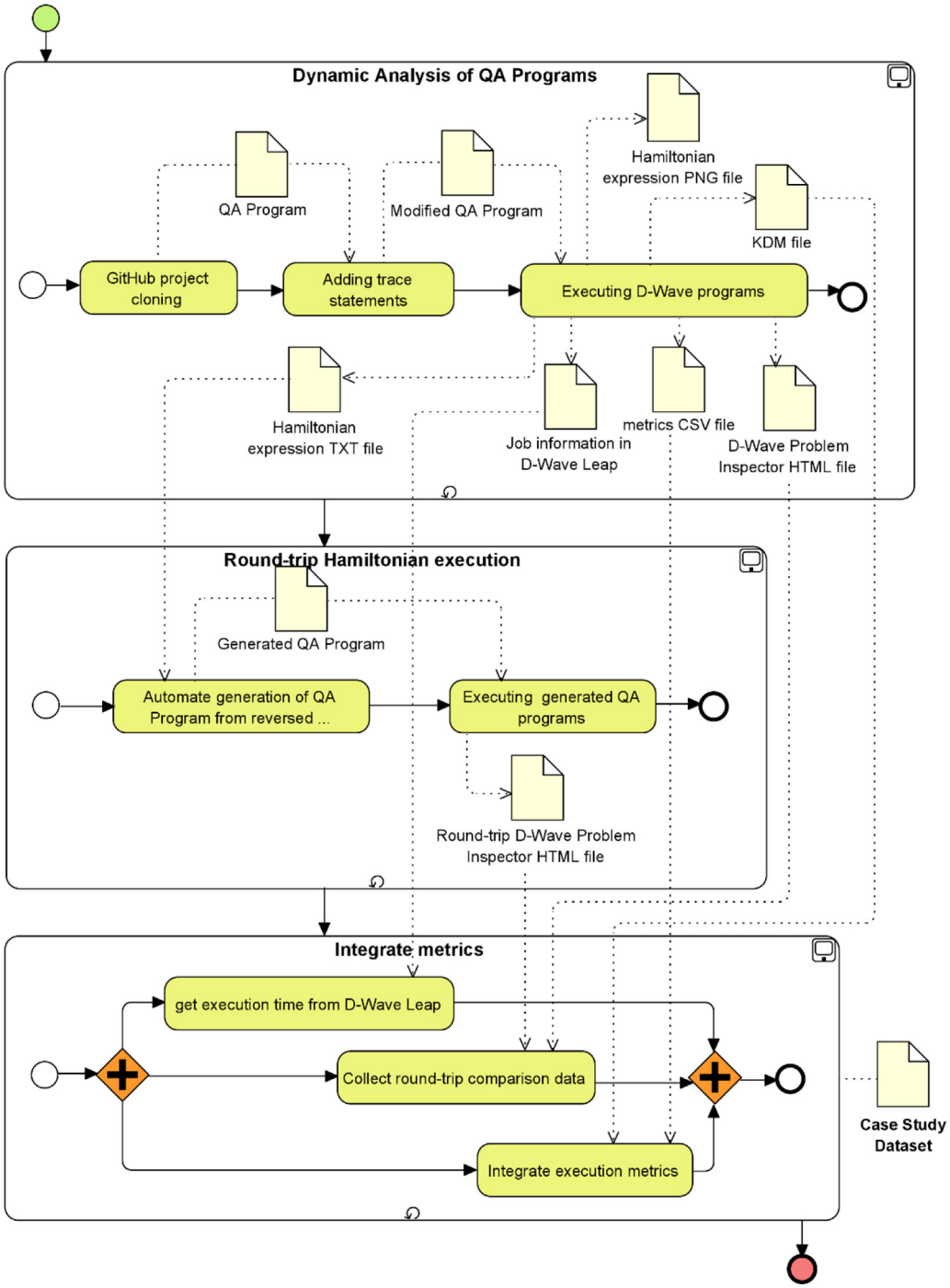

Fig. 10. Data collection procedure.

- When carrying out the case study, the actual progress was registered and compared with the planned progress in order to detect any possible deviations with regard to the case study design. A thorough review was also carried out after the data collection steps for all the execution of QA programs.

4.2. Analysis and interpretation

This section provides the main insights extracted from the multi-case study and attempts to answer the research questions defined.

4.2.1. RQ1. Effectiveness as regards discovering Hamiltonians

Fig. 12 summarizes the success rate for the generation of the Hamiltonians and KDM models, which can also be checked in the aggregated values at the bottom of Table 3. With regard to RQ1, the success factor for Hamiltonian generation was 81%.

These results signify that only five (5) out of 27 programs (C9, C13, C22, C24, C27) could not be discovered. After inspecting the execution logs for these five (5) programs, it was discovered that the failure occurred at the same point: the polynomial simplification. The script developed in Python attempts to transform all the different kinds of problem models (QUBO, Ising, Tuple, etc.) into a BQM. The script, therefore, considers all the arrays of variables and the coefficient matrix, and composes a Hamiltonian

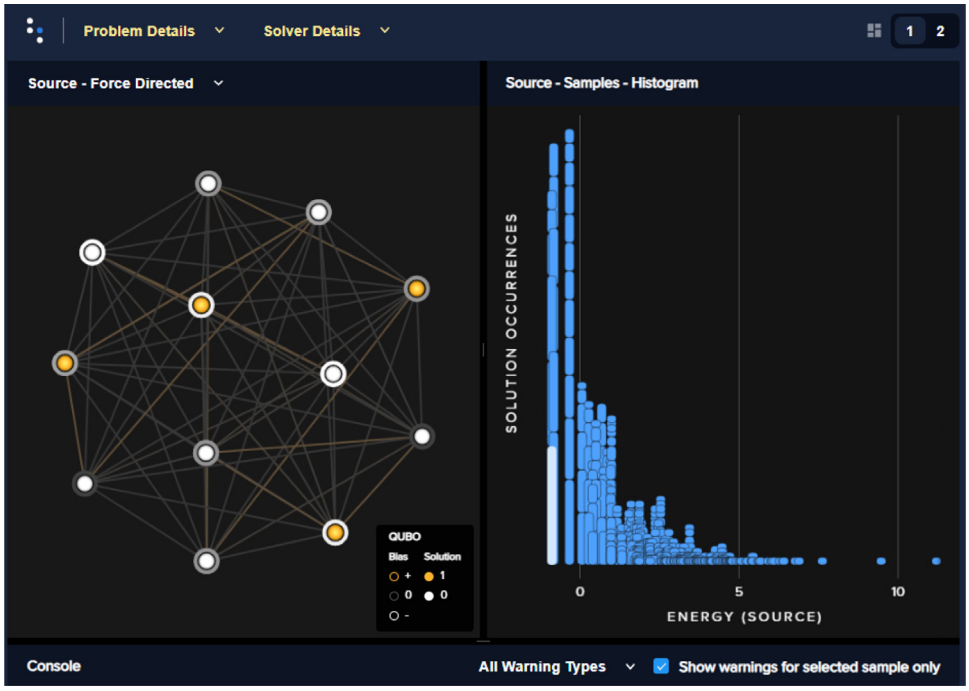

Fig. 11. D-Wave problem inspector example for clustering.py (C16).

Table 3
Case study dataset.

ID	Ham. Gen.?	Valid Ham.?	KDM Gen.?	Valid KDM?	LoC	#Coeff	#Var	Ham. Gen. T (ms)	KDM Gen. T (ms)	QPU time (ms)	Service time (ms)
C1	✓	✓	✓	✓	49	5	4	278	16	0.91	39
C2	✓	✓	✓	✓	48	5	4	247	8	0.25	32
C3	✓	✓	✓	✓	48	5	4	242	8	0.99	38
C4	✓	✓	✓	✓	51	5	4	252	11	0.88	35
C5	✓	✓	✓	✓	43	5	4	252	10	0.96	39
C6	✓	✓	✓	✓	90	11	5	318	8	0.110	37
C7	✓	✓	✓	✓	86	5	4	308	9	1.04	39
C8	✓	✗	✓	✓	274	231	33	4173	105	80.00	3080
C9	✗	✗	✗	✗	126	24310	220				
C10	✓	✗	✓	✓	84	29	11	595	16	111.00	147
C11	✓	✓	✓	✓	70	16	7	380	11	0.90	161
C12	✓	✓	✓	✓	196	36	8	720	14	0.62	657
C13	✗	✗	✗	✗	309	24531	221				
C14	✓	✓	✓	✓	59	73	25	1125	22	115.92	153
C15	✓	✓	✓	✓	186	77	21	1333	24	9.39	42
C16	✓	✗	✓	✓	152	78	12	1319	21	107.84	144
C17	✓	✓	✓	✓	141	88	29	1355	27	124.06	158
C18	✓	✓	✓	✓	77	820	40	21336	953	150.30	184
C19	✓	✓	✓	✓	123	190	52	3282	85	13.14	50
C20	✓	✓	✓	✓	74	16	7	376	11	1.15	43
C21	✓	✗	✓	✓	229	92	16	1524	54	102.00	3101
C22	✗	✗	✗	✗	211	9495	686				
C23	✓	✓	✓	✓	105	709	238	20260	819	19.56	56
C24	✗	✗	✗	✗	282	55161	669				
C25	✓	✗	✓	✓	153	982	53	28025	1581	2836.00	80821
C26	✓	✗	✓	✓	149	55	10	427	10	94.00	78085
C27	✗	✗	✗	✗	194	46038	736				
Total	22	16	22	22							
% Total	81%	59%	81%	81%							
% Rel.	81%	73%	100%	100%							
Mean					134	6040	116	4006	174	171.45	7598
St. Dev					78	14495	220	7979	405	597.53	23278

with several terms. The whole expressions that include terms that repeat variables are then simplified by means of *SymPy*, which is an external Python library. During simplification, huge expressions, therefore, cause an exception to the script, and it skips the simplification. This means that the KDM models are not generated in these exceptional cases. The five (5) cases have a huge number of variables and coefficients, which vary from 686 variables and 9,495 coefficients for C22 to 736 variables and 46,038 coefficients for C27.

Despite this issue, the Hamiltonian expressions that are not simplified are still generated and are mathematically equivalent. However, we agree that these non-simplified expressions could be less readable and, therefore, useless. Nevertheless, we believe that 81% represents a high success rate as regards ensuring that

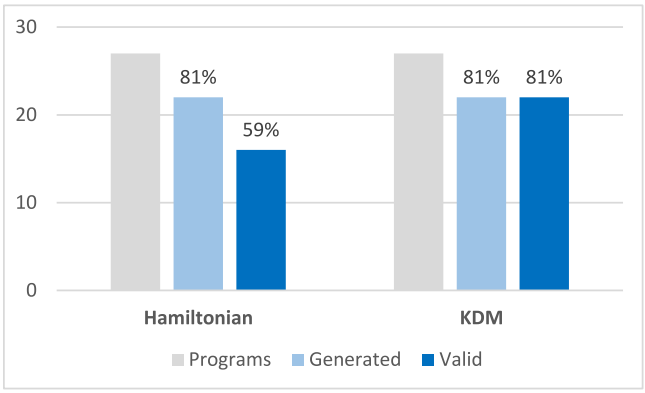

Fig. 12. Summary of effectiveness for Hamiltonian and KDMmodels.

the Hamiltonian expressions can be generated using the proposed dynamic analysis technique.

In order to complete the answer to the research question (how many of the generated Hamiltonian are valid expressions) the expressions were automatically transformed into BQM problem definitions and re-executed (as mentioned before). The results show that 59% (about the total number of programs) were valid, as the solutions obtained for the optimization problems were clearly equivalent. If we discard those five (5) programs that it was not possible to fully extract, the relative percentage is 73%. These results allow us to state (with some hesitation) that the technique is able to obtain valid Hamiltonians.

Upon inspecting the incorrect Hamiltonians (those that reported different solutions with regard to those from the original program) we discovered that there were six (6) in total (C8, C10, C16, C21, C25, C26). In these programs, the solution reported in the re-execution was not significantly different, but we have been unable to attain any strong evidence with which to confirm the equivalent solutions. Although we believe that the round-trip strategy is a good approximation, it is not perfect in QA programs, since the intrinsic non-deterministic nature of programs does not ensure the same solutions for the same optimization problem. It is also important to consider another fact in this assessment strategy: the re-execution was automatically coded under the same laboratory conditions, i.e., the same D-Wave solver, fixed numbers of reads, etc. This configuration varies for all the different programs. As a result, these factors could explain those 6 Hamiltonian that obtained different solutions, although this should be validated in further studies.

In summary, and based on the results obtained, RQ1 can be positively answered, i.e., the proposed dynamic analysis technique is able to generate Hamiltonian expressions from the execution of QA programs in an effective manner.

4.2.2. RQ2. Effectiveness as regards modeling Hamiltonians in KDM

With regards to RQ2, Table 3 and Fig. 12 also summarize the success rate for the generation of KDM models. In this case, 81% of the underlying Hamiltonians in the QA programs were transformed into KDM models. In this case, the only programs that it was not possible to model were those that failed during the simplification of polynomials (see Section 4.2.1). If this detail is ignored, 100% of the Hamiltonians that were fully generated were modeled in KDM. With regard to the validity of the KDM models, 100% of the models generated were determined as being fully compliant with the KDM model (based on an XML/Metamodel checker).

Keeping these results in mind, it is also possible to provide RQ2 with a positive response, i.e., the Hamiltonian expressions extracted can be represented according to the KDM standard and persisted in XML models with an adequate accuracy and completion level.

4.2.3. RQ3. Efficiency and scalability

The objective of RQ3 is to evaluate the efficiency of the proposal and, since there is no previous knowledge regarding similar techniques with which to compare it, in order to assess the scalability of the technique so as to ensure that it could work for larger models.

Fig. 13 shows a box plot in which the magnitude of the time spent generating Hamiltonians and KDM models can be compared with the QPU and Service Time that are really employed in D-Wave systems. While the KDM generation time can be discarded in comparison with the QPU and service time (an average of 174 ms (see Table 3)), the Hamiltonian generation time is higher in comparison with the others. The Hamiltonian generation takes an average of 4006 ms (4 s), although this time varies from 0.2 s to 28 s for larger problems. It should be noted that most of the time is spent simplifying polynomials. We consider that this time

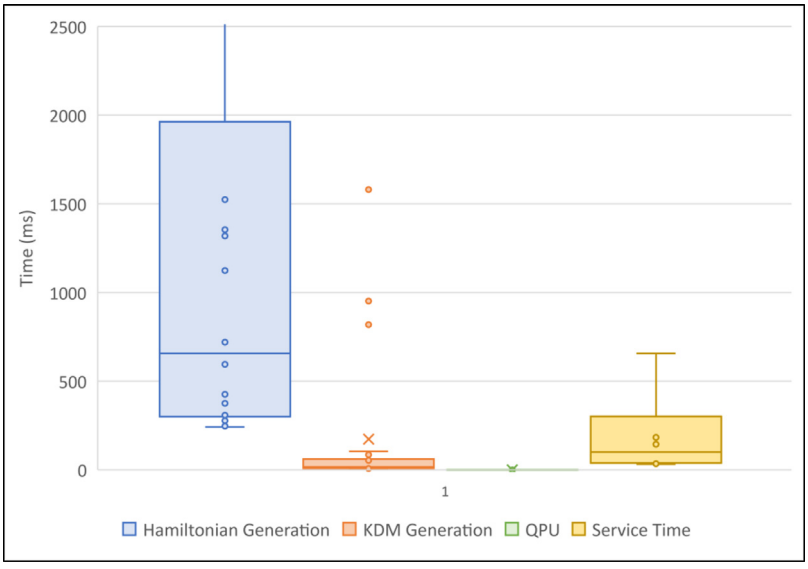

Fig. 13. Box plot for timing variables.

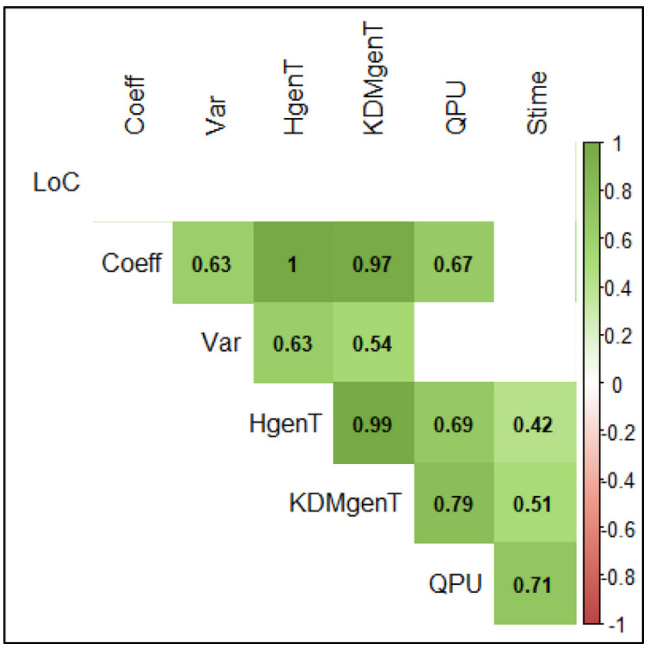

Fig. 14. Correlation matrix for timing and sizing variables.

is acceptable, since this magnitude is still manageable, and can even be discarded for smaller problems.

The temporal complexity for Hamiltonian generation can be theoretically explained through a factor that clearly influences the execution time. The mentioned critical part of the whole process, the transformation from QUBO to Hamiltonians and its simplification, depends on the number of factors, which will determine the number of terms included in the Hamiltonian equation. So, the expected complexity is $O(n)$ where n is the number of factors defined in the QUBO formulation. For every Hamiltonian term, the time spent is on the order of some milliseconds. This leads to overall time measures about milliseconds for Hamiltonians with hundreds of terms, or a few seconds for those Hamiltonians with more than a thousand of terms.

Another part of the answer to RQ3 is the assessment of scalability. In this regard, it is first necessary to discover which of the variables related to the size act as the independent variable affecting the time. Fig. 14 shows a matrix containing the Pearson correlations for each pair of size and timing variables. The important variables in this case are Hamiltonian and KDM generation time, which are strongly correlated with the number of coefficients (1 and 0.97 respectively) and, to a lesser extent, with the number of variables (0.63 and 0.54 respectively). As expected, the number of lines of source code does not correlate with the time. This is because there are sometimes too many lines of source code to build a smaller optimization problem (with few variables and coefficients), while on other occasions the opposite occurs.

The correlation values in Fig. 14 can be analyzed in depth using the scatter plots presented in Fig. 15, which provides a representation of trend lines (regarding the number of coefficients and variables) for the Hamiltonian and KDM generation times. It confirms that the actual important size measure is the number of coefficients. This can be explained by the fact that coefficients in some respects represent constraints that are added to the optimization problem. A small problem in terms of the number of variables could, therefore, result in a more complex one if many coefficients (constraints) are considered. Moreover, the number of coefficients has a direct impact on the number of outgoing terms in the raw polynomial that need to be simplified.

In order to conclude the analysis of RQ3, we believe the time spent obtaining the Hamiltonian expressions and the respective KDM models is feasible. Moreover, the scalability to larger problems is ensured, since there is a linear relationship between the number of coefficients and the time spent on generations. However, as explained in RQ1, it should be noted that there is a performance cliff for large problem definitions (around 200 variables and 9000 coefficients) in which simplifications crash and prevent the generation of the reduced expression. This fact prevents to clearly state the technique is scalable. Hence, RQ3 could consequently be answered in a half-positive manner. Despite the aforementioned problem for the largest programs, it should be taken into account that the simplification depends on an external simplification library. This is not, therefore, dealt with in this research, since it is not considered to be within its scope.

4.3. Validity evaluation

This section provides an evaluation of validity, analyzes threats to the validity and discuss mitigation actions and recommendations for future replications of this study.

4.3.1. Construct validity

This aspect refers to "to what extent the operational measures that are studied really represent what the researcher has in mind and what is investigated according to the research questions" (Runeson et al., 2012). Apart from the measures that simply denote whether a Hamiltonian or a KDM has been built, we defined the respective variables in order to assess whether or not these models were valid. The models in question were defined during the design phase, and an explanation of how to evaluate them was provided. The size of the optimization problems was also analyzed by using different variables, such as the number of variables, coefficients, and lines of code.

4.3.2. Internal validity

This validity aspect refers to the external third factors that may affect some other factors being investigated. No causal relationships were examined in this study. Only some correlations for timing variables were examined, and all the potential size variables were examined (e.g., discarding the number of lines of code). Apart from this, it should be considered that most of the programs used were from the D-Wave repository, in which programs are expected to be well documented and follow code standards. However, third-party programs could not follow the same best practices, and thus it could affect other factors (e.g., the capability to be analyzed among other). Although this is a threat to the validity, there are a limited number of such annealing programs available to be freely accessed. Although such annealing programs could be provided by companies, the dissemination of the source code is usually restricted. As a result, the usage of those programs will lead to a poor reliability and replicability of the case study.

4.3.3. External validity

This aspect evaluates to what extent it is possible to generalize the conclusions, and to what extent the conclusions are applicable to relevant for other cases with common characteristics (Runeson et al., 2012). The goal of this study was to generalize the main insights into all QA programs. However, there are various quantum programming languages that support this kind of programs, and the technique was specially designed for Python code and, in particular, for D-Wave solvers. This is consequently a threat to the generalizability of the results. Nonetheless, it would have been very ambitious to cover several programming languages and different quantum annealer systems. This is because dynamic

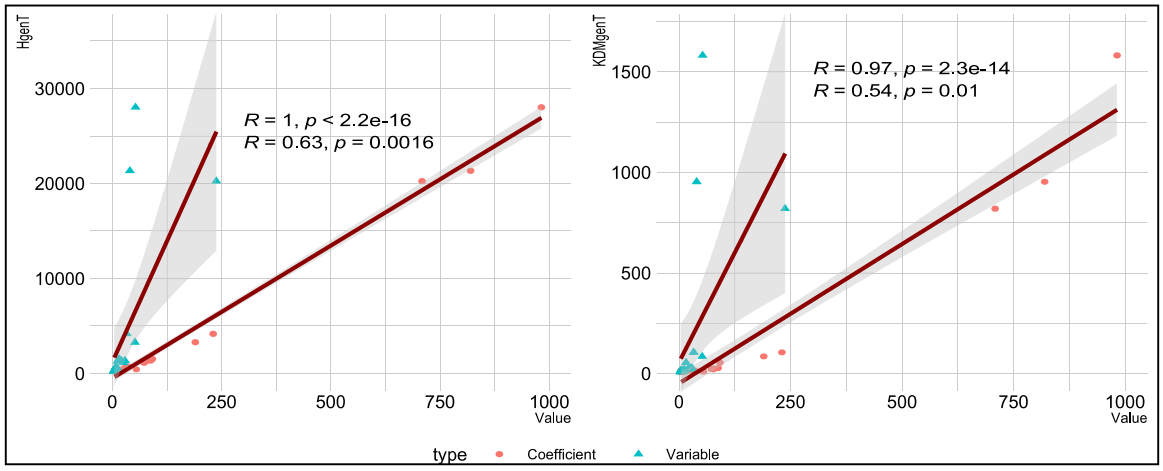

Fig. 15. Scatter plots for Hamiltonian (left) and KDM (right) generation time compared to the number of variables and coefficients in problem definitions.

analysis techniques are, in general, highly dependent on various technical details apart from the obvious ones (e.g., tracking mechanisms, threading, etc.). This could be mitigated in the future by means of a replication of the study with similar dynamic analysis techniques that cover other kinds of systems.

Another threat to the external validity is the size of the problem definition with which the technique has been validated. In this study, in terms of the number of variables and coefficients, the highest cases analyzed had 686 variables and 55 161 coefficients. However, those cases that it was possible to fully analyze using the technique had 238 variables and 982 coefficients. The generalization of the results should, therefore, take this threshold into account.

Finally, it should be noticed that the correlation and scatter plots used to evaluate the hypothesized linear relationship (between the size/complexity of programs and the execution time) has been calculated with a limited number of measures. Hence, further replications by considering additional programs will improve the insights provided in this study.

4.3.4. Reliability

This is concerned with "to what extent the data and the analysis are dependent on the specific researchers" (Runeson et al., 2012). As mentioned in the construct validity section, the definition of measures with which to assess the validity of Hamiltonians relies on a round-trip re-execution strategy, in which synthetic programs were automatically generated from the Hamiltonian and run again. This made it possible to reduce the researchers' subjective point of view as regards the validity of the Hamiltonians obtained. This strategy can lead to false negatives due to the non-deterministic nature of QA programs. Even so, we believe this is better than other alternatives to determine the effectiveness like, for example, those using test suites based on a golden standard. This is because, the expected value to be compared with, is unknown and difficult to be defined. In most of the cases, the definition of the expected Hamiltonian would require the manual intervention by experts, which will result on higher error-proneness. Apart from the difficulty to find and engage experts who were/are involved in the respective QA program. In future replications, a complimentary validation strategy could be implemented by executing the Hamiltonians in a classical way, e.g., through a simulated annealing algorithm. Those additional result comparisons could be considered as ulterior proof of validity.

Moreover, in order to ensure the replicability of the study, the whole code repository (Pérez-Castillo, 2022a) and the case study dataset (Pérez-Castillo et al., 2022) are available for the research community.

5. Conclusions and future work

Quantum software engineering is emerging as a strong means to deal with challenges related to producing quantum software in a systematic manner and with sufficient quality levels. There are various new challenges owing to the nature of quantum software, because their foundations are, to a certain extent, counterintuitive for software engineers. Together with those new challenges, most of the challenges that software engineers had to confront with classical software in the past are also of relevance for quantum software. In this regard, software evolution is a critical challenge, and various open questions need to be addressed in the near future. For example, how quantum software is integrated with classical software; how both kinds of software operate in combination, or how these hybrid software systems are maintained, among others.

Although the research community has now begun to tackle some of these questions, initial efforts have been focused on universal quantum software, while quantum annealing software has not been considered. As a consequence, this research is focused on the reverse engineering of QA programs and its modeling in order to contribute with the modernization from/toward hybrid software systems. The reverse engineering technique, which is based on dynamic analysis, is able to generate the Hamiltonian expressions and a respective abstract model based on KDM, an international ISO/IEC standard. The main implications of this research are the following:

- In comparison with the manual comprehension of QA programs (and how optimization problems are defined), the automatic generation of the underlying Hamiltonian expressions can aid experts in this comprehension. Moreover, reverse engineering can reduce the time and error-proneness of manual comprehension from scratch.
- The abstract representations based on KDM make it possible to integrate and use the knowledge of QA programs in the software modernization process according to the MDE principles. Additionally, the usage of KDM (which is an international ISO/IEC standard) ensures its adoption by industry.
- The case study in which the proposed technique has been applied to 27 D-Wave programs. This research demonstrates the effectiveness and efficiency of this technique. The empirical validation, therefore, provides sufficient evidence of its applicability which, in turn, facilitates its adoption.

With regard to the limitations of this research, it should be noted that, as with any reverse engineering technique, it suffers

from a semantic loss. For example, the outgoing Hamiltonian might sometimes turn out to be useless owing to the fact that the most important thing is to know the meaning of variables and constraints rather than a long mathematical expression. Even in these cases, we argue that the Hamiltonian expressions generated are still useful as black-box pieces that can be integrated into the target systems during modernization toward hybrid software systems. In this scenario, and thanks to the usage of dynamic analysis, this guarantees that the Hamiltonian is accurate.

Future work in this open research will focus on several areas: more general aspects (long/medium term) and specific issues (short-term). With regard to the general aspects, we intend to investigate how to use these reversed Hamiltonians, i.e., how to integrate them with classical and/or universal quantum software. In this case, we shall research how Hamiltonians can be restructured and enriched with further semantics. Beyond achieving the expression of a QA Hamiltonian, we shall investigate whether this is sufficient to achieve the end-goal of the software modernization of hybrid software systems. Despite the fact that KDM models have been chosen to represent Hamiltonians at higher abstraction levels, we shall explore alternative models that could be used to modernize hybrid software systems.

With regard to the specific issues (short-term), we shall address the limitations discovered after conducting the case study. For example, the simplification of larger Hamiltonian expressions; or the support of alternative problem definitions apart from BQM, such as discrete and constrained quadratic models; among others.

CRediT authorship contribution statement

Ricardo Pérez-Castillo: Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Luis Jiménez-Navajas:** Validation, Data curation, Writing – review & editing, Visualization. **Mario Piattini:** Conceptualization, Investigation, Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ricardo Perez-Castillo reports financial support was provided by University of Castilla-La Mancha.

Data availability

Code is in GitHub, validation data is in Zenodo.

Acknowledgments

This work is part of Grant PID2019-104791RB-I00 (SMOQUIN Project) funded by MCIN/AEI/, Spain 10.13039/501100011033; Grant PDC2022-133051-I00 (QU-ASAP Project) funded by MCIN/AEI/10.13039/501100011033 and by the "European Union NextGenerationEU/PRT; and project "QHealth: Quantum Pharmacogenomics Applied to Aging", 2020 CDTI Missions Programme, funded by MICINN, Spain.

References

Aaronson, S., 2008. The limits of quantum. J. Sci. Am. 298 (3), 62–69.

Ali, S., Yue, T., 2020. Modeling quantum programs: challenges, initial results, and research directions. In: Presented at the Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software, Virtual, USA. http://dx.doi.org/10.1145/3412451.3428499, [Online].

Allende López, M., Da Silva, M.M., 2019. Quantum Technologies: Digital Transformation, Social Impact, and Cross-Sector Disruption. Inter-American Development Bank, [Online]. Available https://publications.iadb.org/en/quantum-technologies-digital-transformation-social-impact-and-cross-sector-disruption.

aoun, M.R.E., Li, H., Khomh, F., Openja, M., 2021. Understanding quantum software engineering challenges an empirical study on stack exchange forums and GitHub issues. In: presented at the 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), 27 Sept.-1 Oct. 2021.

Awan, U., Hannola, L., Tandon, A., Goyal, R.K., Dhir, A., 2022. Quantum computing challenges in the software industry, a fuzzy AHP-based approach. Inf. Softw. Technol. 147, 106896. http://dx.doi.org/10.1016/j.infsof.2022.106896.

Bozzo-Rey, M., Longbottom, J., Müller, H.A., 2019. Quantum computing: challenges and opportunities. In: presented at the Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, Toronto, Ontario, Canada.

Calude, C.S., Calude, E., Dinneen, M.J., 2015. Guest column: Adiabatic quantum computing challenges. J. SIGACT News 46 (1), 40–61. http://dx.doi.org/10.1145/2744447.2744459.

Cao, Y., Romero, J., Aspuru-Guzik, A., 2018. Potential of quantum computing for drug discovery. IBM J. Res. Dev. 62 (6), 6:1–6:20. http://dx.doi.org/10.1147/JRD.2018.2888987.

Cao, Y., et al., 2019. Quantum chemistry in the age of quantum computing. Chem. Rev. 119 (19), 10856–10915. http://dx.doi.org/10.1021/acs.chemrev.8b00803.

Chen, H., Lidar, D.A., 2022. Hamiltonian open quantum system toolkit. Commun. Phys. 5 (1), 112. http://dx.doi.org/10.1038/s42005-022-00887-2.

Cooper, C.H.V., 2021. Exploring potential applications of quantum computing in transportation modelling. IEEE Trans. Intell. Transp. Syst. 1–9. http://dx.doi.org/10.1109/TITS.2021.3132161.

Córcoles, A.D., et al., 2020. Challenges and opportunities of near-term quantum computing systems. Proc. IEEE 108 (8), 1338–1352. http://dx.doi.org/10.1109/JPROC.2019.2954005.

Cruz-Lemus, J.A., Marcelo, L.A., Piattini, M., 2021. Towards a Set of Metrics for Quantum Circuits Understandability. Cham.

D-Wave, 2021. How D-wave systems work. https://www.dwavesys.com/learn/quantum-computing/, (Accessed 04/08/2021, 2021).

D-Wave, 2023. Dwave Ocean SDK.

De Stefano, M., Pecorelli, F., Di Nucci, D., Palomba, F., De Lucia, A., 2022. Software engineering for quantum programming: How far are we? J. Syst. Softw. 190, 111326. http://dx.doi.org/10.1016/j.jss.2022.111326.

Dridi, R., Alghassi, H., 2017. Prime factorization using quantum annealing and computational algebraic geometry. Sci. Rep. 7 (1), 43048. http://dx.doi.org/10.1038/srep43048.

Egger, D.J., et al., 2020. Quantum computing for finance: State-of-the-art and future prospects. IEEE Trans. Quantum Eng. 1, 1–24. http://dx.doi.org/10.1109/TQE.2020.3030314.

Garcia-Alonso, J., Rojo, J., Valencia, D., Moguel, E., Berrocal, J., Murillo, J.M., 2022. Quantum software as a service through a quantum API gateway. IEEE Internet Comput. 26 (1), 34–41. http://dx.doi.org/10.1109/MIC.2021.3132688.

Garg, S., Ramakrishnan, G., 2020. Advances in quantum deep learning: An overview. arXiv preprint, arXiv:.04316.

Gemeinhardt, F., Garmendia, A., Wimmer, M., Towards model-driven quantum software engineering. In: presented at the 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), 1-2 June 2021.

Gill, S.S., et al., 2022. Quantum computing: A taxonomy, systematic review and future directions. 52, (1), pp. 66–114. http://dx.doi.org/10.1002/spe.3039.

Glover, F., Kochenberger, G., Du, Y., 2018. A tutorial on formulating and using QUBO models. arXiv preprint, arXiv:.11538.

Hevia, J.L., Murina, E., Peterssen, G., Piattini, M., 2021. A new path to create solutions for quantum annealing problems. J. Quantum Inf. Sci. 11 (3), 112–123. http://dx.doi.org/10.4236/jqis.2021.113009.

IBM, 2021. The Quantum Decade. a Playbook for Achieving Awareness, Readiness, and Advantage. IBM Institute for Business Value, [Online]. Available https://www.ibm.com/thought-leadership/institute-business-value/report/quantum-decade.

Jiménez-Navajas, L., Pérez-Castillo, R., Piattini, M., 2021. KDM to UML Model transformation for quantum software modernization. In: presented at the International Conference on the Quality of Information and Communications Technology (QUATIC'21), Virtual.

Kahn, J., 2021. D-Wave Unveils Its Most Powerful Quantum Computer to Date. Fortune, https://fortune.com/2020/09/29/d-wave-5000-qubit-quantum-computer/, (Accessed 2021).

Kang, Y.-H., Chen, Y.-H., Wu, Q.-C., Huang, B.-H., Xia, Y., Song, J., 2016. Reverse engineering of a Hamiltonian by designing the evolution operators. Sci. Rep. 6 (1), 30151. http://dx.doi.org/10.1038/srep30151.

L. Recruit Communications Co., 2023. PyQUBO. Recruit Communications Co., Ltd. https://pyqubo.readthedocs.io/en/latest/, (Accessed 2023).

Lewis, M., Glover, F., 2017. Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis. Networks 70 (2), 79–97. http://dx.doi.org/10.1002/net.21751.

Liu, J., Zhou, H., 2020. Reliability modeling of NISQ- era quantum computers. In: presented at the 2020 IEEE International Symposium on Workload Characterization (IISWC), 27-30 Oct. 2020.

McCaskey, A., Dumitrescu, E., Liakh, D., Humble, T., 2018. Hybrid programming for near-term quantum computing systems. In: presented at the 2018 IEEE International Conference on Rebooting Computing (ICRC), 7-9 Nov. 2018.

McGeoch, C.C., 2020. Theory versus practice in annealing-based quantum computing. Theoret. Comput. Sci. 816, 169–183. http://dx.doi.org/10.1016/j.tcs.2020.01.024.

Meichanetzidis, K., Gogioso, S., De Felice, G., Chiappori, N., Toumi, A., Coecke, B., 2020. Quantum natural language processing on near-term quantum computers. arXiv preprint, arXiv:.04147.

Mueck, L., 2017. Quantum software. Nature 549 (7671), 171. http://dx.doi.org/10.1038/549171a.

OMG, 2016. Architecture-Driven Modernization (ADM): Knowledge Discovery Meta-Model. KDM, OMG, v1.4. https://www.omg.org/spec/KDM/1.4/PDF. [Online]. Available https://www.omg.org/spec/KDM/1.4/PDF.

OMG, 2021. ADM Task Force by Object Management Group. Object Management Group, https://www.omg.org/adm/, (Accessed 29/12/2021, 2021).

Ou, C.-H., Jiang, D.-J., Chen, C.-Y., Yu, L.-P., Chang, C.-R., 2022. Smart agriculture decision making scheme using quantum annealing. In: 2022 IEEE International Conference on Quantum Computing and Engineering (QCE). Broomfield, CO, USA, pp. 862–863. http://dx.doi.org/10.1109/QCE53715.2022.00145.

Pérez-Castillo, R., 2022a. D-wave examples for dynamic analysis of quantum annealing programs. https://github.com/ricpdc/dwave_reverse/tree/case_study/dwave_examples, (Accessed 2022).

Pérez-Castillo, R., 2022b. Ricpdc/dwave_reverse repository. https://github.com/ricpdc/dwave_reverse, (Accessed 03/01/2022).

Pérez-Castillo, R., de Guzmán, I.G.-R., Piattini, M., 2011. Knowledge discovery metamodel-ISO/IEC 19506: A standard to modernize legacy systems. Comput. Stand. Interfaces 33 (6), 519–532. http://dx.doi.org/10.1016/j.csi.2011.02.007.

Pérez-Castillo, R., Jiménez-Navajas, L., Mario, P., 2022. Dataset for Dynamic Analysis of Quantum Annealing Programs (0.1) [Data Set]. Zenodo, http://dx.doi.org/10.5281/zenodo.6546492, (Accessed 2022).

Pérez-Castillo, R., Jiménez-Navajas, L., Piattini, M., 2021a. Modelling quantum circuits with UML. In: presented at the 43rd ACM/IEEE International Conference on Software Engineering Workshops. 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), Virtual (originally in Madrid, Spain), May (2021) 25-28, 2021, 2.

Pérez-Castillo, R., Jiménez-Navajas, L., Piattini, M., 2021b. Qrev: migrating quantum code towards hybrid information systems. Softw. Qual. J. http://dx.doi.org/10.1007/s11219-021-09574-x.

Pérez-Castillo, R., Serrano, M.A., Piattini, M., 2021c. Software modernization to embrace quantum technology. Adv. Eng. Softw. 151, 102933. http://dx.doi.org/10.1016/j.advengsoft.2020.102933.

Piattini, M., Peterssen, G., Pérez-Castillo, R., 2020a. Quantum computing: A new software engineering golden age. J. SIGSOFT Softw. Eng. Notes 45, 12–14. http://dx.doi.org/10.1145/3402127.3402131.

Piattini, M., Peterssen, G., Serrano, M.A., Hevia, J.L., Pérez-Castillo, R., 2021. Towards a quantum software engineering. IT Prof. 23 (1), 62–66. http://dx.doi.org/10.1109/MITP.2020.3019522.

Piattini, M., et al., 2020b. The talavera manifesto for quantum software engineering and programming. In: Presented at the QANSWER 2020. Quantum SoftWare Engineering & Programming, Talavera de la Reina, 1. [Online]. Available http://ceur-ws.org/Vol-2561/paper0.pdf.

Preskill, J., 2018. Quantum computing in the NISQ era and beyond. Quantum 2 (aug), 79. http://dx.doi.org/10.22331/q-2018-08-06-79.

QURECA, 2021. Overview on quantum initiatives worldwide. https://www.qureca.com/overview-on-quantum-initiatives-worldwide/, (Accessed 06/08/2021, 2021).

Runeson, P., Host, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons.

Savoie, C., 2021. How quantum computers could cut millions of miles from supply chains and transform logistics. In: Forbes Technology Council. Forbes, Feb 5, 2021. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2021/02/05/how-quantum-computers-could-cut-millions-of-miles-from-supply-chains-and-transform-logistics/.

The Economist, 2021. Wall street's latest shiny new thing: quantum computing. Dec 19th 2020. [Online]. Available: https://www.economist.com/finance-and-economics/2020/12/19/wall-streets-latest-shiny-new-thing-quantum-computing.

Wallden, P., Kashefi, E., 2019. Cyber security in the quantum era. J. Commun. ACM 62 (4), 120. http://dx.doi.org/10.1145/3241037.

Wohlin, C., 2021. Case study research in software engineering—It is a case, and it is a study, but is it a case study? Inf. Softw. Technol. 133, 106514. http://dx.doi.org/10.1016/j.infsof.2021.106514.

Yin, R.K., 2013. Case Study Research: Design and Methods, fifth ed. SAGE Publications, London, p. 312.

Yin, R.K., 2014. Case Study Research: Design and Methods, fifth ed. Sage publications.

Zhang, Y., Ni, Q., 2020. Recent advances in quantum machine learning. 2, (1), e34. http://dx.doi.org/10.1002/que2.34.

Zhao, J., 2020. Quantum software engineering: Landscapes and horizons. arXiv preprint, arXiv:2007.07047v1.