POLITECNICO DI TORINO
Repository ISTITUZIONALE

Improving the exploitability of Simulated Adiabatic Bifurcation through a flexible and open-source digital architecture

Original
Improving the exploitability of Simulated Adiabatic Bifurcation through a flexible and open-source digital architecture / Volpe, Deborah; Cirillo, Giovanni; Zamboni, Maurizio; Graziano, Mariagrazia; Turvani, Giovanna. - In: ACM TRANSACTIONS ON QUANTUM COMPUTING. - ISSN 2643-6817. - ELETTRONICO. - (2024). [10.1145/3665281]

Availability:
This version is available at: 11583/2989404 since: 2024-06-10T19:58:11Z

Publisher:
ACM

Published
DOI:10.1145/3665281

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
ACM postprint/Author's Accepted Manuscript, con Copyr. autore

© Volpe, Deborah; Cirillo, Giovanni; Zamboni, Maurizio; Graziano, Mariagrazia; Turvani, Giovanna 2024. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in ACM TRANSACTIONS ON QUANTUM COMPUTING, http://dx.doi.org/10.1145/3665281.

(Article begins on next page)

06 August 2024

Improving the exploitability of Simulated Adiabatic Bifurcation through a flexible and open-source digital architecture

DEBORAH VOLPE, Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Italy
GIOVANNI CIRILLO, Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Italy
MAURIZIO ZAMBONI, Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Italy
MARIAGRAZIA GRAZIANO, Politecnico di Torino Facoltà di Ingegneria, Torino, Italy
GIOVANNA TURVANI, Politecnico di Torino, Torino, Italy

Combinatorial Optimization (CO) problems exhibit exponential complexity, constraining classical computers from providing fast and satisfactory outcomes. Quantum Computers (QCs) can effectively find optimal or near-optimal solutions by exploring the solutions space of a problem encoded in a qubits system, exploiting principles of quantum mechanics. However, non-idealities and high costs limit their availability. These can be overcome by emulating QCs on cheaper and more accessible classical computing platforms, like Field-Programmable Gate Arrays (FPGAs).

This article presents a digital architecture, implementing the Ising-compatible Simulated Adiabatic Bifurcation algorithm. It mimics the quantum adiabatic evolution of a network of non-linear Kerr oscillators. The architecture, described in VHDL and targeting FPGAs, consists of processing elements for computing the Kerr oscillators' evolution, a set of units considering their Ising-related interactions and an evolution variables update unit. The proposed approach includes a speedup-targeting approximation of the algorithm, a method for handling single-variable constraints, and a software model that allows architecture customization for specific problems. Tests were conducted using an Altera Cyclone V SoC with FPGA logic and the Nios II processor for interface purposes. The results demonstrate the functionality of the architecture and its scalability with the problem size, making it suitable for real-world applications.

Additional Key Words and Phrases: Ising machine, QUBO, optimization, Kerr oscillator, Simulated Bifurcation, Adiabacity

Just Accepted

1 INTRODUCTION

Quantum computing research is achieving new results — in terms of algorithm definitions, applications, and device engineering — every day. Regardless of the paradigm considered (quantum circuit model [40] or quantum annealing [28]), the state-of-the-art already presents theoretical proofs of the potential computational advantages of quantum computation, particularly in applications involving large-scale data processing such as optimization and machine learning [3, 14, 16, 35, 40]. However, it should be noted that these proofs are based on worst-case complexities and experimental evidence of advantage is yet to be fully established. Some limitations must be considered when quantum computers are going to be used. First, quantum hardware fabrication, control, and maintenance are costly nowadays. For this reason, companies manufacturing hardware — which can be either big

Authors' Contact Information: Deborah Volpe, Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Italy; e-mail: deborah.volpe@polito.it; Giovanni Cirillo, Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Piemonte, Italy; e-mail: giovanniamedeo.cirillo@st.com; Maurizio Zamboni, Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Piemonte, Italy; e-mail: maurizio.zamboni@polito.it; Mariagrazia Graziano, Politecnico di Torino Facoltà di Ingegneria, Torino, Piemonte, Italy; e-mail: mariagrazia.graziano@polito.it; Giovanna Turvani, Politecnico di Torino, Torino, Piemonte, Italy; e-mail: giovanna.turvani@polito.it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2024 Copyright held by the owner/author(s).

ACM 2643-6817/2024/5-ART
https://doi.org/10.1145/3665281

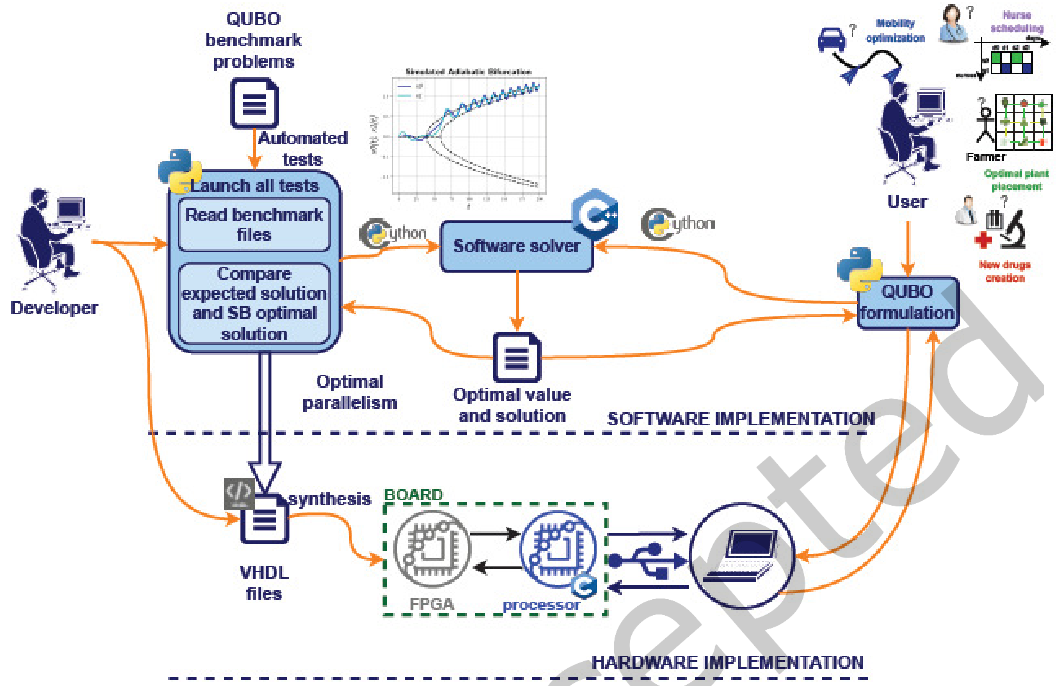


Fig. 1. Overview of the article. Software implementations evaluate the SBa proposed acceleration by exploiting a benchmark set of max-cut, knapsack, and traveling salesman problems. This analysis also allows the verification of the feasibility of the hardware in fixed-point numbers representation, providing the opportunity to choose the proper data parallelism. The VHDL description of the algorithm is then synthesized on an Altera Cyclone V FPGA interfacing with the Nios II processor. The processor aims to send the problem's matrix and algorithm parameters to the FPGA and collect the results. Both the software models and the hardware description are open source, allowing users to generate QUBO problems and solve them with at least one among software or hardware implementations of the algorithm.

Just Accepted

companies like IBM, Google, and Intel or startups like Pasqal, Rigetti, IonQ, QuEra, and D-Wave Systems — give the possibility to access their devices via cloud (both with and without fees). Moreover, **hardware non-ideality phenomena** like decoherence and relaxation [8, 40], unwanted qubits interactions or excitation [13, 44, 57] and connectivity limitations [33, 58, 64] can significantly affect the reliability of the results.

These issues could partially limit the exploitability of quantum hardware to real-world problems, particularly for companies and academic institutes unable to access a real quantum computer or afford usage fees. A compromise between quantum computers' physical limitations and computational advantages can be achieved through their **classical emulation**, belonging to a branch of the so-called **quantum-inspired computing**, where quantum phenomena *inspire* the definition of procedures accelerating specific tasks, such as solving Combinatorial Optimization (CO) problems [5]. Hardware non-idealities would not affect emulated qubits, and they can be managed on established computing platforms like **Field-Programmable Gate Arrays** (**FPGAs**) or **Graphics Processing Units** (**GPUs**). Since these are more affordable than current quantum computers, this approach can open the way for developing custom applications based on the programming of *on-premises* quantum-inspired machines.

Lessons learned from the design of classical computing systems can be leveraged to improve quantum computing systems, for example, in applications like circuit compilation [33] or to implement efficient and reliable quantum-inspired application-specific classical computers. Hardware emulators are expected to be preferable to software-based simulators in approximating quantum phenomena because the parallel nature of quantum computation can be emulated more accurately and flexibly [30]. Very-large-scale Integration (VLSI) design methodologies for Digital Signal Processing [41] can be considered a good starting point for designing quantum emulators since these are usually based on models and algorithms involving arithmetic operations.

This article presents a quantum-inspired digital solver for Ising models based on the Simulated Adiabatic Bifurcation (SBa) algorithm [21], which emulates the evolution of a quantum system. Therefore, the proposed solver can also be employed with problems described with the Quadratic Unconstrained Binary Optimization (QUBO) formalism [17], which is equivalent to the Ising model. To the best of our knowledge, the presented architecture will be the **first open-source implementation of the SBa algorithm**, and it is characterized by a **flexible description**, allowing the synthesis of the most suitable hardware in terms of **dimension** and **data parallelism** for the problems of interest. Unlike most of the SBa machines, specifically designed for solving max-cut problems, which involve only the interaction term, the presented implementation **can also address problems including the external field contribution of the Ising model**. Another peculiarity of the proposed architecture is its **near-memory** approach, and an **approximation of the original algorithm proposed** to enhance solver speed.

Moreover, the SBa machine was inserted and developed inside the context of the toolchain shown in Figure 1. The optimized solver is the hardware implementation (bottom part of the figure), corresponding to a digital circuit described in VHDL and synthesized on FPGA. Additionally, a software implementation of the same is available to provide users with higher flexibility in terms of employable backends for solving Ising/QUBO problems and to serve as a reference functional model for the hardware implementation. As mentioned, the hardware description is generic to allow the choice of the best parallelism and size of the machine for the user's purpose. The FPGA used to perform tests is put on a board also supporting synthesis of the Nios II processor, currently employed for interfacing purposes, which could also be used to calibrate some parameters of the custom hardware.

The goals of this article are outlined in the following list:

- to provide a comprehensive overview of SBa algorithm encompassing all the main concepts behind this quantum system emulation mechanism;
- to assess the SBa performance with generic Ising problems involving constraints on both single spins and pairs, which can be considered closer to real-world scenarios;
- to explore potential methods for an efficient SBa implementation of Ising problems with h vector;
- to propose and validate an SBa approximation achieving a higher degree of parallelization without significantly affecting the quality of the solutions obtained;
- to present open-source software and hardware implementations, aiming to facilitate the comprehension of the SBa algorithm among a wider community of users.

The article is organized as follows. Section 2 presents the theoretical foundations, focusing particularly on the Ising model and its relation with the QUBO model, the benchmark problems considered, and introduces of the Simulated Adiabatic Bifurcation (SBa) algorithm. Section 3 introduces the proposed architecture and the toolchain where it is embedded, highlighting the novelty introduced by this work. Section 4 reports and examines the results obtained.. Finally, in Section 5, conclusions are drawn, and future perspectives are discussed.

2 THEORETICAL FOUNDATIONS

This section presents the Ising formulation, which is the native model of simulated bifurcation machines and many other classical, quantum, and quantum-inspired solvers, and its relation with the more popular QUBO formulation (Section 2.1). Moreover, the problems considered for benchmarking the proposed software and

Just Accepted

hardware implementation are introduced (Section 2.1.2).
Furthermore, Section 2.2 provides a detailed presentation of the Simulated Adiabatic Bifurcation (SBa) algorithm.

2.1 Optimization Problems formalism

2.1.1 Ising model. The **Ising model** [38, 49] is a **physical-mathematical** model of ferromagnetism employed in statistical mechanics for the description of the magnetic properties of a material. It describes a system of interacting **atomic spins** modelled as dipoles arranged in a lattice, where each spin can assume one of two discrete states, depending on its orientation: **+1** (**spin-up**) or **-1** (**spin-down**). The following Hamiltonian can describe this model:

$$H(\mathbf{s}) = \frac{1}{2} \sum_{i=0}^{N_{\text{SPIN}}-1} \sum_{j=0, j\neq i}^{N_{\text{SPIN}}-1} J_{ij} s_i s_j + \sum_{i=0}^{N_{\text{SPIN}}-1} h_i s_i \,, \tag{1}$$

where N_{SPIN} is the number of spins, s_i is the i^{th} spin, J_{ij} is the **interaction coefficient** between the i^{th} and the j^{th} spins and h_i is the **external magnetic field coefficient** of the i^{th} spin. In simulated bifurcation machine, a **symmetric** J matrix is considered, hence $J_{ij} = J_{ji}$.
The Ising Hamiltonian comprises two types of interaction:

- Interaction terms J, whose weight and sign determine whether neighbouring spin pairs prefer aligned or anti-aligned (specifically, ferromagnetic and anti-ferromagnetic) orientation. The sum of all pair contributions yields the overall interaction energy.
- External field h, where the sign determines the preferred orientation of a spin (up or down) and the weight of the energy contribution of a single spin in the system's energy.

Any system composed of paired elements that can be modelled as spins can be described using this model. It is categorized **1D**, **2D**, **3D**, or **fully connected**, depending on the number of interacting neighbours for each spin. The first identifies systems with two interacting neighbours, while four and six interacting spins characterize 2D and 3D structures, respectively. In a fully connected system, each spin interacts with all the others.
Recently, Ising formulation has been extensively used for describing **combinatorial optimization** problems. In particular, the optimal solution of the problem corresponds to the **ground state** of the Ising Hamiltonian.
Moreover, this formulation is the **native** model for **quantum annealers** and other **quantum and quantum-inspired optimization solvers** and is perfectly equivalent to the **Quadratic Unconstrained Binary Optimization** (**QUBO**) formulation. The latter involves **unipolar binary variables** — i.e., which can assume only 0 and 1 values —, as implied by the term **Binary** in the acronym. The term **Quadratic** refers to the highest power applied to these variables, allowing the description of combinatorial **Optimization** through the following objective function:

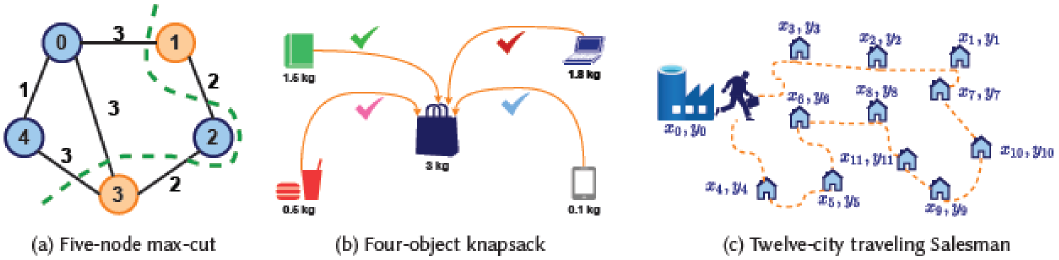
$$\text{Obj}(c, a_i, b_{ij}, x_i) = c + \sum_i x_i \cdot a_i + \sum_{i<j} b_{ij} \cdot x_i x_j \,, \tag{2}$$

where $x_i \in [0, 1]$ is a binary variable, $x_i x_j$ is a coupler that allows two variables to influence each other, a_i is a weight or bias associated with a single variable (analogous to Ising h_i), b_{ij} is a strength which controls the influence of variables i and j (analogous to Ising J_{ij}), and c is an offset, which can be neglected during the optimization.
It can also be expressed as:

$$\text{minimize/maximize} \quad y = f(\mathbf{x}) = \mathbf{x}^t \cdot Q \cdot \mathbf{x} \,, \tag{3}$$

where $\mathbf{x}$ is a vector of binary variables (*e.g.* [0,1,1,0,1]) and Q is a square matrix of constants, depending on the problem. The matrix Q can be **symmetric** or in **upper triangular form**.

Just Accepted



(a) Five-node max-cut (b) Four-object knapsack (c) Twelve-city traveling Salesman

Fig. 2. Optimization problems considered for benchmarking.

Just Accepted

As in Ising model, the variable constraints are not explicitly considered in this formulation (as the term **Unconstrained** in the acronym suggests), but they can be evaluated by exploiting proper penalty functions:

$$\text{minimize/maximize} \quad y = f(\mathbf{x}) + \lambda g(\mathbf{x}), \tag{4}$$

where λ is a positive parameter to be multiplied by the constraint function or **penalty function** $g(\mathbf{x})$, whose sizing is critical, as discussed in [17, 60].
Moving a problem from QUBO formulation to Ising, and vice versa, is always possible by exploiting the following relation:

$$q_i = \frac{1 + s_i}{2}, \tag{5}$$

and its counterpart:

$$s_i = 2q_i - 1. \tag{6}$$

Knowing how to move from one formulation to the other is valuable. Indeed, the Ising model, as mentioned, is the **native formulation for many solvers** (classical, quantum, and quantum-inspired), which are consequently defined as **Ising machines**. At the same time, in the literature, there are many QUBO descriptions of real-world problems [15, 24, 45, 48], and the quadratic penalty functions associated with several constraints can be easily written with known formulas [17]. In general, describing an optimization problem with unipolar binary variables is expected to be easier and more intuitive than doing the same with bipolar variables. Therefore, to simplify the formulation of the optimization problem of interest, it is common to write the target problem according to the QUBO formulation and then translate it to the Ising formulation to exploit a quantum or quantum-inspired solver QUBO-to-Ising translation, and vice versa, can be automatically done with Python libraries like qubovert [26], PyQUBO [25, 63] and dimod [2].

2.1.2 Benchmark problems considered. In this paragraph, the benchmark problems considered for the proposed simulated bifurcation machine are presented. They were chosen because they are sufficiently different to approach the solver corner cases and to verify their effectiveness in a real-world context where the problems' characteristics vary. Specifically, it is crucial to assess the algorithm's performance considering problems with and without the external field contribution (h) associated with single-variable constraints. Moreover, to stimulate the algorithm effectively, it is important to consider problems where the external field predominates over the interaction coefficients, thus presenting a scenario opposite to that for which the approach is designed. Furthermore, exploring problems with different ranges of coefficient values and energy profile characteristics is necessary to evaluate the exploration capabilities of the presented approaches in multiple scenarios. As discussed in the following sections, the chosen benchmark meets these characteristics.

Each QUBO formulation is written and converted to Ising formulation through qubovert library for each type of benchmark problem. For some of them also, sets of problems available online were exploited.

Max-cut. **Max-cut** [12, 22] (Figure 2(a)) is one of the most known CO problems because it can be exploited to describe **several real-world** problems in network design, statistical physics, VLSI design, and circuit layout design [36]. Its goal is **partitioning a graph into two complementary subsets, S and $\overline{S}$, maximizing the cut**, i.e. the **sum of edges joining the two sets**. In its QUBO formulation, **a binary variable is required for each node**, which assumes value one if the node belongs to subset S and 0 otherwise. The cut can be represented as the quantity $\epsilon_{(i,j)} = x_i + x_j - 2x_i x_j$, whose value equals one if the edge (i, j) is in the cut and zero otherwise. Taking into account all the edges, the obtained cost function is the following:

$$\text{Maximize} \quad y = \sum_{(i,j)\in E} w_{i,j}\epsilon_{(i,j)} = \sum_{(i,j)\in E} w_{i,j} \cdot (x_i + x_j - 2x_i x_j) \tag{7}$$

where $w_{i,j}$ is the weight of the edge that connects the i^{th} and the j^{th} node.
Furthermore, the max-cut problem has two peculiarities. The first regards the **energy profile**, which is **symmetric**; this means that, since the obtained two subsets are interchangeable, a solution and its complement (e.g. [0,1,1,0,1] and [1,0,0,1,0]) have the same energy, so they are equivalent. The second concerns the corresponding Ising formulation, whose Hamiltonian is the following:

$$H_{\text{max-cut}} = -\sum_{(i,j)\in E} w_{i,j} s_i s_j \,. \tag{8}$$

Note that **the external field contribution h is not present**. This is the most relevant characteristic of the problem and well justifies the fact that this problem is usually employed as a benchmark of the consolidated SBa solver in the state-of-the-art, not considering single-spin variables.
In this work, a set of randomly generated max-cut problems of different sizes (from 3 to 1000 nodes) and the well-known G-set max-cut problems, whose dimensions range from 800 to 20000 nodes, were considered for benchmarking purposes. Randomly generated max-cut problems were considered, as they allow fine-grained variations in problem size, a feature often absent in online problems. Moreover, these problems have the flexibility of edge weights that can take integer values within the range [0, 10], which is in contrast with the restricted set of [-1, 0, 1] in the G-Set. The maximum size of the generated problems is constrained to ensure the estimation of the reference optimal values.

Knapsack. The **knapsack** problem [11] (Figure 21) aims to define the **best subset of objects** belonging to a set X. Each object is characterized by a preference parameter p_i and a weight w_i and the optimization problem seeks the **subset maximizing the total score**:

$$P = \sum_{i\in\text{subset}} p_i \,, \tag{9}$$

while not exceeding a certain weight threshold $W_{\mathbf{max}}$:

$$\sum_{i=1}^{\dim(X)} w_i x_i \leq W_{\max} \,, \tag{10}$$

When p_i are all equal, the problem is reduced to selecting the highest possible number of objects to put in the bag without exceeding the maximum weight.
Auxiliary variables are required to represent the **inequality constraint** in QUBO formulation, the number of

Just Accepted

which depends on the maximum weight [16, 17, 59]. Therefore, the final cost function is as follows:

$$f_{\text{knapsack}}(\mathbf{x}) = f_{\text{inequality}}(\mathbf{x}) - \sum_{i} p_i x_i \,. \tag{11}$$

The knapsack problem is widespread because it can be **adapted for writing many resource selection optimizations**, e.g., in industrial environments. Moreover, to additionally prove the importance of these problems in a real-world scenario, recent attempts to reformulate knapsack QUBO problems, tailored for specific solvers, have been proposed to improve execution reliability in terms of time and quality of the obtained solutions. For example, [42] proposes reformulations targeting the Quantum Approximate Optimization Algorithm (QAOA), compliant with the quantum circuit model, e.g. in terms of choice of the penalty constants and removal of slack variables.

This problem was chosen as a benchmark due to its importance in the actual application context and its difference with respect to the max-cut problem; for example, **it presents the contribution of the external field**.

In the case of the knapsack problem, the external field stands out as the primary contribution, evident from its formulation. In fact, the single-variable contributions are given by the linear combination of the preference factor $-p_i x_i$ (Equation 11) and the weight constraints $w_i^2 x_i^2 - 2W_{\max} w_i x_i = (w_i^2 - 2W_{\max} w_i) x_i$ (Equation 10), while the constraints on pairs are given by $-2w_i w_j x_i x_j$. Since $2W_{\max} w_i$ is expected to be greater than w_i^2 and $2w_i w_j$, the overall single-variable constraints reasonably provide the most significant contribution to the problem.

This article considers a set of randomly generated problems of different sizes (from 2 to 103 binary variables) and the 0/1 Knapsack set, whose sizes range from 8 to 110 binary variables. Randomly generated knapsack problems were considered as they allow fine-grained variations in problem size, not present in the state-of-the-art benchmarks. Nevertheless, as evaluations require estimating the expected optimal values, the maximum size considered for the generated problems is constrained.

Traveling salesman. The **Traveling Salesman Problem** (TSP) [6, 31] (Figure 2(c)) is one of the most famous combinatorial optimizations, aiming to find the shortest possible path for a salesman to **visit a set of locations**, **traversing each once** and **returning to the origin**. The problem's popularity stems from its applicability to real-world scenarios such as transportation services, goods distribution and delivery, planning, and logistics.

One approach to express an N_{city}-TSP-problem according to the QUBO formulation is by employing a matrix of $N_{\text{city}} \times N_{\text{city}}$ binary variables, where the variable x_{ij} assumes value one if the i^{th} city is visited as j^{th}. For a city-based TSP, the minimization of the overall covered Euclidean distance can be expressed as:

$$f_{\text{distance}} = \sum_{u=0}^{N_{\text{city}}-1} \sum_{v=0}^{N_{\text{city}}-1} \left(\sum_{j=0}^{N_{\text{city}}-2} \left(\sqrt{(c_{x_u} - c_{x_v})^2 + (c_{y_u} - c_{y_v})^2} x_{uj} x_{v(j+1)} \right) + \sqrt{(c_{x_u} - c_{x_v})^2 + (c_{y_u} - c_{y_v})^2} x_{u(N_{\text{city}}-1)} x_{v0} \right), \tag{12}$$

where c_{x_u}, c_{y_u}, c_{x_v}, and c_{y_v} are the x and y spatial coordinate of the u^{th} and v^{th} cities, respectively. To ensure meaningful results, it is necessary to impose that exactly one variable assumes the value one for each row and column. In this way, all the cities are visited once (except for the starting one) and one at a time. These constraints can be described through the following penalties function:

$$f_{\text{row}} = \sum_{i=0}^{N_{\text{city}}-1} \left(\sum_{j=0}^{N_{\text{city}}-1} x_{ij} - 1 \right)^2, \tag{13}$$

$$f_{\text{col}} = \sum_{j=0}^{N_{\text{city}}-1} \left(\sum_{i=0}^{N_{\text{city}}-1} x_{ij} - 1 \right)^2. \tag{14}$$

Just Accepted

Hence, the **final cost function** is the following:

$$f_{\text{TSP}} = \sum_{u=0}^{N_{\text{city}}-1} \sum_{v=0}^{N_{\text{city}}-1} \left(\sum_{j=0}^{N_{\text{city}}-2} \left(\sqrt{(c_{x_u} - c_{x_v})^2 + (c_{y_u} - c_{y_v})^2} x_{uj} x_{v(j+1)} \right) + \sqrt{(c_{x_u} - c_{x_v})^2 + (c_{y_u} - c_{y_v})^2} x_{u(N_{\text{city}}-1)} x_{v0} \right) + \\ + \lambda_{\text{row}} \left(\sum_{i=0}^{N_{\text{city}}-1} \left(\sum_{j=0}^{N_{\text{city}}-1} x_{ij} - 1 \right)^2 \right) + \lambda_{\text{col}} \left(\sum_{j=0}^{N_{\text{city}}-1} \left(\sum_{i=0}^{N_{\text{city}}-1} x_{ij} - 1 \right)^2 \right), \quad (15)$$

where λ_{row} and λ_{col} are the weights of the penalty functions f_{row} and f_{col}, respectively.

This article employs TSPs of different sizes (from 9 to 81 binary variables), randomly generated according to the aforementioned formulation, to test the designed Ising solver. The choice of involving this family of combinatorial optimization problems is driven by their importance in applicative scenarios and to **the presence of single-variable constraints**, that are not available in max-cut problems and can introduce an additional complexity to be handled by the SBa architecture. In contrast to the knapsack case, these problems exhibit a balanced coefficient size between the single-variable (h) and interaction contributions (J). Indeed, both types of coefficients assume values in the same order of magnitude.

TSPs are generated by granularly varying the number of cities and considering the distances involved in a wide range of values. This allows the stimulation of solvers with problems featuring coefficients of diverse magnitudes. As for the previous problems, the maximum size is constrained to ensure the identification of the reference optimal values.

2.2 Simulated Adiabatic Bifurcation

Simulated Adiabatic Bifurcation (**SBa**) is a **heuristic quantum-inspired algorithm** recently proposed in [21] for obtaining **approximate solutions of large-size optimization problems** written according to **Ising** formulation **in a limited amount of iterations**. In the original version, it supports only Ising problems without the external field component, exemplified by the following equation:

$$H(\mathbf{s}) = \frac{1}{2} \sum_{i=0}^{N_{\text{SPIN}}-1} \sum_{j=0, j \neq i}^{N_{\text{SPIN}}-1} J_{ij} s_i s_j \, . \quad (16)$$

In particular, it emulates on classical platforms the **adiabatic evolution** of a **quantum system** involving a network of **non-linear Kerr oscillators** [18] (**KPO**). The main peculiarity of these quantum oscillators is their ability to represent a superposition of two oscillation states, commonly known as a Schrödinger cat state, through quantum adiabatic evolution across its bifurcation point (Figure 3), thereby enabling the representation of a state analogous to that of a qubit [19]. Through this mechanism, a kind of quantum computer called **Quantum Bifurcation Machines** (**QbMs**) is achieved. Analogously to a measurement mechanism in standard quantum computation, at the end of the evolution, the oscillator exhibits a bifurcation phenomenon, where the chosen branch identifies the measured value of the qubit. QbMs are usually employed for quantum adiabatic optimization, but, theoretically, can be employed also for generic quantum computation as well if corresponding operations for oscillation control are defined.

The quantum mechanical KPOs Hamiltonian as follows:

$$H_q(t) = \hbar \sum_{i=1}^{N_{\text{SPIN}}} \left[\frac{K}{2} a_i^{\dagger 2} a_i^2 - \frac{p(t)}{2} (a_i^{\dagger 2} + a_i^2) + \Delta a_i^{\dagger 2} a_i^2 \right] + \hbar \xi_0 \sum_{i=1}^{N_{\text{SPIN}}} \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} a_i^{\dagger 2} a_i^2 \, , \quad (17)$$

where $\hbar$ is the reduced Plank constant, K is the positive Kerr coefficient of oscillators, $a_i^\dagger$ and a_i are the creation and annihilation operators, respectively, $p(t)$ is the photon pumping amplitude — initially at zero, then growing

Just Accepted

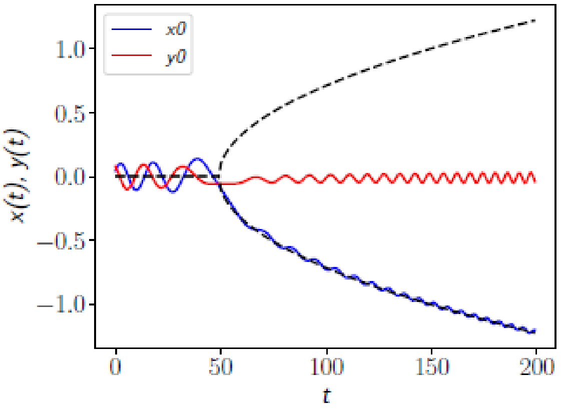
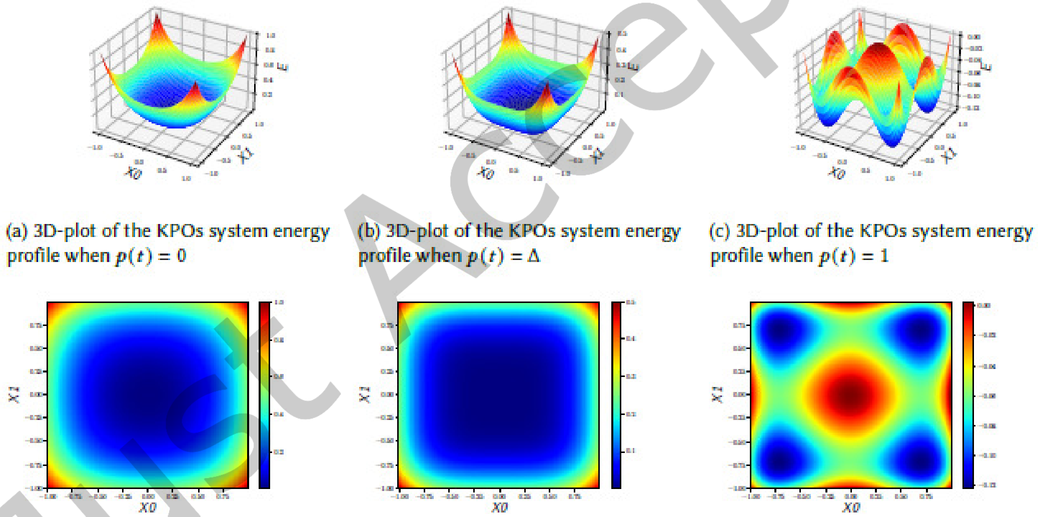


Fig. 3. Time evolution of a **non-linear Kerr parametric oscillator** (**KPO**), where x and y are the **momentum** and **position** variable of the oscillator, respectively.



(a) 3D-plot of the KPOs system energy profile when $p(t) = 0$

(b) 3D-plot of the KPOs system energy profile when $p(t) = \Delta$

(c) 3D-plot of the KPOs system energy profile when $p(t) = 1$

(d) 2D-plot of the KPOs system energy profile when $p(t) = 0$

(e) 2D-plot of the KPOs system energy profile when $p(t) = \Delta$

(f) 2D-plot of the KPOs system energy profile when $p(t) = 1$

Fig. 4. Uncorrelated KPOs system energy profile evolution varying the pumping amplitude (without considering the optimization problem).

to a sufficiently high value —, Δ is the positive detuning frequency and ξ_0 is a positive constant.
The initial state of each oscillator of the quantum system ($p(t) = 0$) is the vacuum state, which is the **superposition of two coherent states** (cat state), as shown in Figures 4(a) and 4(d). It is the ground state of the initial Hamiltonian.

Just Accepted

Just Accepted

Gradually increasing the pumping amplitude $p(t)$ from zero to a sufficiently high value, each KPO becomes a coherent state with positive or negative amplitude of quantum adiabatic bifurcation. If the evolution is sufficiently slow, i.e., adiabatic, the system is in its ground state in each time instant (Figures 4(b) and 4(e)).
Without considering the contribution related to the optimization problem, which creates a correlation among KPOs, every possible combination of the oscillators in one of the two coherent states gives the final ground state. Indeed, each oscillator chooses one of the two states chaotically because both minimize the network's total energy (Figures 4(c) and 4(f)).
The role of the problem contribution $\hbar\xi_0 \sum_{i=1}^{N_{\text{SPIN}}} \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} a_i^{\dagger 2} a_i^2$ is to create an **imbalance in the energy profile** associated with the Hamiltonian such that the **final ground state of the KPOs network is its optimal solution**, as shown in Figure 5(b).
To compute the evolution of the described KPOs system classically, it is necessary to approximate the expected value a_i as a complex amplitude $x_i + jy_i$, where $x_i = \frac{a_i^\dagger + a_i}{2}$ and $y_i = \frac{a_i^\dagger - a_i}{2}$ are the canonical conjugated variables position and momentum of the i^{th} oscillator. The corresponding classical mechanic Hamiltonian is:

$$H_c(\mathbf{x}, \mathbf{y}, t) = \hbar \sum_{i=1}^{N_{\text{SPIN}}} \left[\frac{K}{4}(x_i^2 + y_i^2)^2 - \frac{p(t)}{2}(x_i^2 - y_i^2) + \frac{\Delta}{2}(x_i^2 + y_i^2)^2 \right] + \hbar \frac{\xi_0}{2} \sum_{i=1}^{N_{\text{SPIN}}} \sum_{j=1}^{N_{\text{SPIN}}} J_{ij}(x_i x_j + y_i y_j)\,. \tag{18}$$

Consequently, the equations of motion that can be obtained for this classical system are:

$$\dot{x_i} = \frac{\delta H_c}{\delta y_i} = [K(x_i^2 + y_i^2) + p(t) + \Delta]y_i + \xi_0 \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} y_j\,, \tag{19}$$

$$\dot{y_i} = -\frac{H_c(t)}{x_i} = -[K(x_i^2 + y_i^2) - p(t) + \Delta]x_i - \xi_0 \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} x_j\,, \tag{20}$$

where dots indicate the differentiation with respect to the time t.
It was proven [18] that in the classical equivalent of the KPOs network, a **good approximation of the combinatorial optimization solution** can be obtained by considering the **final sign of motion variable x_i as the value of the spin variable s_i in the final solution**. This substantially permits the definition of a new family of **quantum-inspired classical Ising machines**.
To render Equations 19 and 20 suitable for **fast numerical simulation on classical hardware**, **some terms** proportional to momentum **y**, which varies around zeros, can be **neglected** (in the following, $\cancel{\alpha}$ means that a variable $\alpha \approx 0$), thus simplifying the equation as follows:

$$\begin{aligned} H_{SB}(\mathbf{x}, \mathbf{y}, t) = \hbar \sum_{i=1}^{N_{\text{SPIN}}} \left[\frac{K}{4}(x_i^2 + \cancel{y_i^2})^2 - \frac{p(t)}{2}(x_i^2 - \cancel{y_i^2}) + \frac{\Delta}{2}(x_i^2 + y_i^2)^2 \right] - \hbar \frac{\xi_0}{2} \sum_{i=1}^{N_{\text{SPIN}}} \sum_{j=1}^{N_{\text{SPIN}}} J_{ij}(x_i x_j + \cancel{y_i y_j}) \approx \\ \approx \sum_{i=1}^{N_{\text{SPIN}}} \frac{\Delta}{2} y_i^2 + V(\mathbf{x}, t) = \sum_{i=1}^{N_{\text{SPIN}}} \frac{\Delta}{2} y_i^2 + \sum_{i=1}^{N_{\text{SPIN}}} \left[\frac{K}{4} x_i^4 - \frac{\Delta - p(t)}{2} x_i^2 \right] + \frac{\xi_0}{2} \sum_{i=1}^{N_{\text{SPIN}}} \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} x_i x_j\,. \end{aligned} \tag{21}$$

$$\dot{x_i} = \frac{\delta H_c}{\delta y_i} = \cancel{K(x_i^2 + y_i^2)y_i} + \cancel{p(t)y_i} + \Delta y_i - + \cancel{\xi_0 \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} y_j} \approx \Delta y_i\,, \tag{22}$$

$$\dot{y_i} = -\frac{H_c(t)}{x_i} = -[K(x_i^2 + \cancel{y_i^2}) - p(t) + \Delta]x_i - \xi_0 \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} x_j \approx -[K x_i^2 - p(t) + \Delta]x_i - \xi_0 \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} x_j\,, \tag{23}$$

where $V(\mathbf{x})$ is the potential energy. The same detuning Δ is considered for all. Note that the resulting system is a network of **Duffing oscillators** with mass equal to Δ^{-1} and coupling coefficient equal to $\xi_0 J_{ij}$ [56]. The **fourth power of the position variable** x_i represents a **potential barrier** that initially **constrains the motion** of each oscillator. This ensures the system is initialized around the ground state, i.e., the zero position, and triggers bifurcation when the problem contribution emerges.

Note that the simplified Equation 21 permits the **separation of position and momentum**, differently from the original formula, enabling the resolution of Equations 22 and 23 with the **Euler's method**, that is a stable numerical method for solving Hamiltonian equations of motion. This operation is crucial for obtaining a hardware implementation of the Ising solver. **By discretizing the time** in time-step Δ_t, the following explicit symplectic Euler's method can be obtained for the update of position and momentum variables:

$$x_i(t_{n+1}) = x_i(t_n) + \Delta y_i(t_n)\Delta_t\,, \tag{24}$$

$$y_i(t_{n+1}) = y_i(t_n) - [Kx_i^3(t_{n+1}) + (\Delta - p(t_{n+1}))x_i(t_{n+1}) + \xi_0 \sum_{j=1}^{N_{\text{SPIN}}} J_{ij}x_j(t_{n+1})]\Delta_t\,, \tag{25}$$

where t_n is the n^{th}-time-instant ($t_n = n\Delta_t$).

To achieve the correct evolution of the network, the **position and momentum variables should be initialized close to zero**. However, **at least one must deviate from zero** to emulate the mechanism initiating the oscillation from noise, typically employed to activate oscillators.

In their updated expression, the cubic terms of the position variables come from the derivative of the fourth-power potential barrier included in the Hamiltonian. Its computation could be one of the most crucial aspects during execution. Instead, the pumping amplitude $p(t_n)$ should be initialized at zero and gradually increase each time step. The algorithm pseudocode is provided in 1.

Algorithm 1 SBa algorithm

```
Input: J matrix
Output: Solution vector s and Energy value
Initialize:
  //Initialize position and variables of the $N_{\text{SPIN}}$ oscillators
  for $i = 0$ To $i < N_{\text{SPIN}}$ do
    initialize $x_i$ to a random number close to zero
    initialize $y_i$ to a random number close to zero
  end for
  //Define the value of the parameter
  initialize parameters $K$, $\Delta$, $\Delta_t$, $\xi_0$, $p_{\text{shape}}$ and NumIter
  //Initialize to 0 the pumping amplitude
  $p \leftarrow 0$

SBa steps:
  //In each time step
  for $t = 0$ To $t <$ NumIter do
    //Update the position variables (parallelizable part)
    for $i = 0$ To $i < N_{\text{SPIN}}$ do
      $x_i \leftarrow x_i + \Delta y_i \Delta_t$
    end for
    //Update the momentum variables (parallelizable part)
    for $i = 0$ To $i < N_{\text{SPIN}}$ do
```

Just Accepted

```
        //Computation of the problem contribution
        temp ← 0
        for $j = 0$ To $j < N_{\text{SPIN}}$ do temp ← $temp + J_{ij}x_j$
        end for
        $y_i \leftarrow y_i - (Kx_i^3 + (\Delta - p)x_i + \xi_0\text{temp})\Delta_t$
    end for
    $p \leftarrow p + p_{\text{shape}}$
end for
//Optimal solution identification
for $i = 0$ To $i < N_{\text{SPIN}}$ do
    $s_i \leftarrow \text{sign}(x_i)$
end for
//Computation of final energy
Energy ← 0
for $i = 0$ To $i < N_{\text{SPIN}}$ do
    for $j = 0$ To $j < N_{\text{SPIN}}$ do
        Energy ← Energy + $J_{ij}s_is_j$
    end for
end for
Return: Energy, s
```

Note that each Kerr oscillator's position and momentum update is **independent** of the others, except the optimization problem term $\xi_0 \sum_{j=1}^{N_{\text{SPIN}}} J_{ij}x_j(t_{n+1})$, which represents the **only data dependencies among spins variables**. Therefore, the **oscillator update can be parallelized**. This is the main advantage of choosing this approach for hardware implementation with respect to other quantum-inspired algorithms, such as Simulated Quantum Annealing (SQA) [60]. Another benefit of this approach is the **elimination of random number generation** for each iteration. Indeed, aside from the initially selected random state of the oscillators' system, the SBa algorithm is **deterministic**. Moreover, it is possible to write the algorithm as a set of **matrix operations**, which can facilitate implementations based on **GPUs**, known for their suitability in handling matrix operations. It has been demonstrated that SBa enables the minimization of cost functions associated with a fully-connected 2000-node max-cut problem ten times faster than other Ising machines and it solves effectively large-scale high-density problems [21].

However, the quality of the results strongly depends on the **choice of oscillators' network parameters** and the **initial oscillator system state**. Consequently, it is not always ensured to find a stable solution, requiring **multiple repetitions** of the routine.

Furthermore, the main limitation of the original SBa algorithm proposed is its **inability to directly manage the external field component of the Ising formulation of the problem**. In the state-of-the-art, some efforts have been made to extend the approach to all kinds of problems. The proposed methods range from re-writing the Ising problem with an external field component as a Hamiltonian with higher dimension involving only **J** matrix, which also includes the **h** vector, as in [65], to exploiting an additional evolution variable multiplied by **h** for gradually considering its contribution, as in [7, 46, 51, 66]. In this work, after analyzing the literature and performing some software tests, the methodology proposed by [46] and exploited in [7] has been followed. This method consists of using the evolution variable $A(t)$, which modifies the previous equations as follows:

$$H_{SB}(\mathbf{x}, \mathbf{y}, t) = \sum_{i=1}^{N_{\text{SPIN}}} \frac{\Delta}{2} y_i^2 + V(\mathbf{x}, t) = \sum_{i=1}^{N_{\text{SPIN}}} \frac{\Delta}{2} y_i^2 + \sum_{i=1}^{N_{\text{SPIN}}} \left[\frac{K}{4} x_i^4 - \frac{\Delta - p(t)}{2} x_i^2 \right] + \frac{\xi_0}{2} \sum_{i=1}^{N_{\text{SPIN}}} \sum_{j=1}^{N_{\text{SPIN}}} J_{ij} x_i x_j + 2\xi_0 A(t) \sum_{i=0}^{N_{\text{SPIN}}} h_i x_i , \tag{26}$$

Just Accepted

$$\dot{x_i} = \Delta y_i \,, \tag{27}$$

$$\dot{y_i} = -[Kx_i^2 - p(t) + \Delta]x_i - \xi_0 \sum_{j=1}^{N_{\mathrm{SPIN}}} J_{ij} x_j - 2\xi_0 A(t) h_i \,. \tag{28}$$

Consequently, the position and momentum update expressions become:

$$x_i(t_{n+1}) = x_i(t_n) + \Delta y_i(t_n)\Delta_t \,, \tag{29}$$

$$y_i(t_{n+1}) = y_i(t_n) - [Kx_i^3(t_{n+1}) + (\Delta - p(t_{n+1}))x_i(t_{n+1}) + \xi_0 \sum_{j=1}^{N_{\mathrm{SPIN}}} J_{ij} x_j(t_{n+1}) + 2\xi_0 A(t) h_i]\Delta_t \,. \tag{30}$$

However, finding a suitable evolution for A(t) is a challenging task. In [46], $A(t)$ close to 0 for $p(t) \ll \Delta$ and $A(t) \approx \sqrt{\frac{p(t)-\Delta}{K}}$ for $p(t) \gg \Delta$ is suggested. Unfortunately, to the best of our knowledge, no article explains the employed evolution for $A(t)$ in detail.

2.2.1 Two-oscillator network evolution. To enhance comprehension of the mechanism behind SBa, a simple two-spin Ising model with **anti-ferromagnetic coupling** is analyzed step-by-step in the following. This model implies that ground-state spin configurations are either down-up or up-down, and its Hamiltonian is shown in 31.

$$H = s_0 s_1 \,. \tag{31}$$

When pumping amplitude $p(t)$ equals zero, the Hamiltonian of the simulated Kerr oscillators network is equal to:

$$H(0) = \frac{K}{4}(x_0^4 + x_1^4) + \frac{\Delta}{2}(x_0^2 + x_1^2) + \xi_0(x_0 x_1) + \frac{\Delta}{2}(y_0^2 + y_1^2) \,. \tag{32}$$

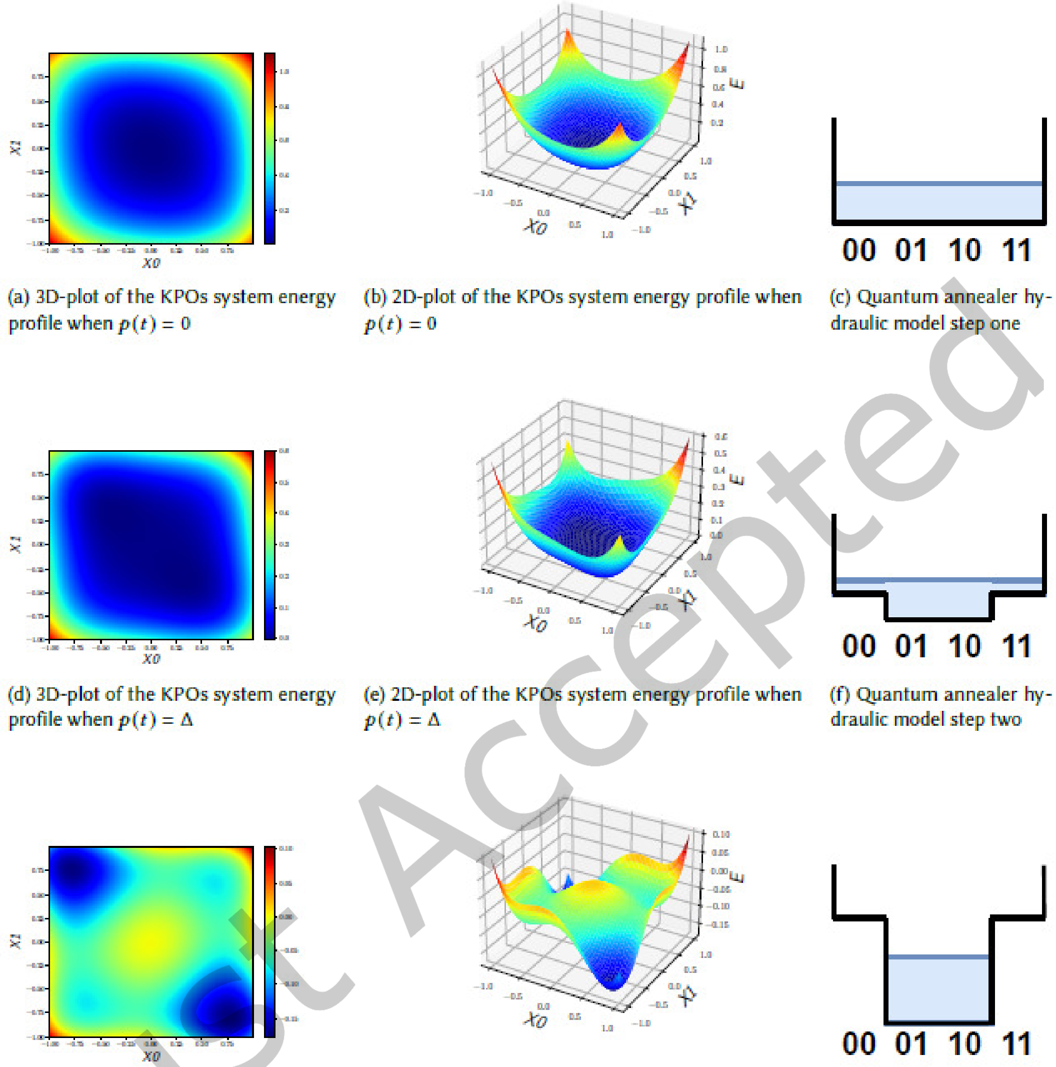
The position and momentum variables of the two involved oscillators are initialized to values close, but slightly different from zero to start the network oscillation. From the Figures 5(a) and 5(b), Note that at this time instant, the energy profile has a single minimum (and stable point) at the origin. Therefore, the trajectory of both oscillators moves around the origin, implying that position and momentum variables oscillate around zero, as shown in Figure 6(b).

By linearly increasing the $p(t)$ up to $\Delta - \xi_0$, the optimal solution of the problem of interest appears as ground states (and stable points) of the oscillators network energy profile. Continuing to increase the pumping amplitude, all possible spin configurations become minima (and stable points) of the Hamiltonian. However, the solutions of the optimization problems are associated with lower energies, i.e., they are global minima of the energy profile, as illustrated in Figures 5(d) and 5(e).

The appearance of optimal solution configurations before and their lower energy convergence ensures convergence to the minimum. Indeed, in the case of adiabatic evolution, the trajectory will stabilize around one of the global minima. Figure 6(b) shows that when $p(t)$ is about equal to $\Delta - \xi_0$ (first bifurcation point, represented as the first dashed black asymptote in Figure 6(b)), the positions of the two oscillators start to follow the branches, assuring a minimum energy configuration. At the second bifurcation point ($p(t) = \Delta + \xi_0$, represented as the second dashed black asymptote in Figure 6(b)), when the other local minima appear, the oscillators should have already chosen the bifurcation branch, thus limiting the possibility of obtaining sub-optimal solutions.

All the previous considerations, related to the achievement of the optimal solutions, are clearly valid assuming that the algorithm parameters have been properly chosen and calibrated.

Just Accepted



(a) 3D-plot of the KPOs system energy profile when $p(t) = 0$

(b) 2D-plot of the KPOs system energy profile when $p(t) = 0$

(c) Quantum annealer hydraulic model step one

(d) 3D-plot of the KPOs system energy profile when $p(t) = \Delta$

(e) 2D-plot of the KPOs system energy profile when $p(t) = \Delta$

(f) Quantum annealer hydraulic model step two

(g) 3D-plot of the KPOs system energy profile when $p(t) = 1$

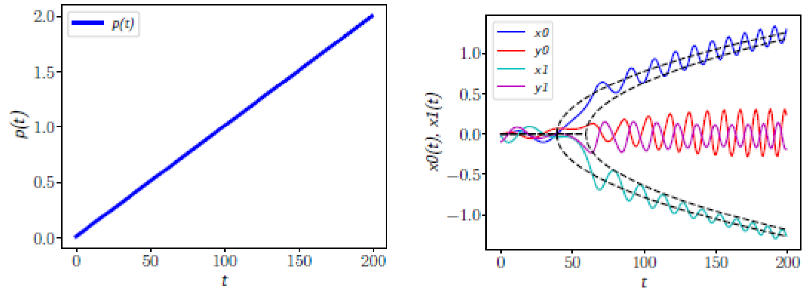
(h) 2D-plot of the KPOs system energy profile when $p(t) = 1$

(i) Quantum annealer hydraulic model final step

Fig. 5. Comparison between KPOs system energy profile evolution varying the pumping amplitude and quantum annealer hydraulic model time evolution, applying the same optimization problem in both cases.

2.2.2 Comparison between Quantum Annealer and Simulated Adiabatic Bifurcation . Quantum annealers [10, 27, 28, 43] are special-purpose quantum computers that allow the minimization of a problem's cost function, formulated in either Ising or QUBO representation, through an adiabatic evolution of their quantum system. Nowadays, they are the best-performing quantum solvers, in terms of complexity of processable optimization

Just Accepted



(a) The $p(t)$ slope required for an adiabatic evolution of the simulated KPOs network

(b) Position and momentum variable evolutions of the two oscillators

Fig. 6. Pumping amplitude, position and momentum variables of two-oscillators network time-evolutions considering an anti-ferromagnetic coupling as an optimization problem to solve.

problems. The Hamiltonian associated with their system is as follows:

$$H(t) = \sum_{ij} J_{ij}\sigma_i^z\sigma_j^z + h\sum_i \sigma_i^z + \Gamma(t)\sum_i \sigma_i^x \; = \; = H_0 + \Gamma(t)\sum_i \sigma_i^x \,, \tag{33}$$

where H_0 is the standard Ising model on which the problem is mapped, $\Gamma(t)$ is a time-dependent transverse field which varies during the evolution for permitting quantum tunneling between system eigenstates, and σ_i^x and σ_i^z are the **Pauli matrices** associated with x and z components of the i^{th} Ising Hamiltonian spin, respectively:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{34}$$

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{35}$$

A hydraulic model is commonly employed to provide a qualitative idea of quantum annealer system evolution. This intuitive representation is exploited in the current work to compare the solutions space exploration done by QA and KPOs (Figure 5).

Initially, when the transverse field is high, the quantum annealer creates the superposition of all possible states with equal weight (spins aligned on the x-axis), corresponding to making the bottom of the tank flat and ensuring a uniform distribution of the water among the solutions (Figure 5(c)). On the other hand, at the beginning of the oscillator's network evolution, the only ground configuration is the origin, so the trajectory of each spin oscillates around this (Figures 5(a) and 5(b)). As mentioned in Section 2.2, this corresponds to a superposition of all possible states.

As the strength of the transverse field decreases, the system energy gradually reproduces the problem energy profile. This corresponds in the hydraulic model to a gradual deformation of the tank's bottom to describe the objective function, and the water begins to flow towards the lowest points (Figure 5(f)). Similarly, the oscillator network's energy profile is gradually deformed so that minimum points are created in correspondence with the lowest energy configurations associated with the Hamiltonian problem (Figures 5(d) and 5(e)). In the end, the quantum annealer converges to the optimal solution, described in the hydraulic model as the water concentrated in the lowest point (Figure 5(i)). At the same time, the trajectory of the oscillators converges to one of the minimum points (Figures 5(g) and 5(h)). Both models obtain the correct behaviour only for sufficiently slow evolution and correct parameter setting.

Just Accepted

2.2.3 Previous works. Some hardware implementations of the simulated bifurcation algorithm, called **Simulated Bifurcation Machines** (**SBMs**), have already been designed and applied to real-world problems based on Ising/QUBO formulations.

To the best of our knowledge, the first was presented by H. Goto et al. in [21, 52]. They propose an SB-based Ising machine with a single **Arria 10 GX 1150 FPGA** described in OpenCL, capable of handling at most 4096 Ising spin variables. Benchmarks involved only max-cut problems with random edge weights in the set $\{-1; 0; +1\}$, which allowed hardware simplifications, especially in terms of required memory and arithmetic components.

Considering that the sum of products operation is the most computationally intensive and that is the only element inserting data dependencies among the position and momentum of the various oscillators, a method for speeding up the algorithm was proposed, i.e., splitting the Hamiltonian as:

$$H_{\mathrm{SB}} = M\frac{H_{\mathrm{SB}} - H_J}{M} + H_J\,, \tag{36}$$

where M is an integer number greater or equal than 2 and $H_J = -\frac{\xi_0}{2}\sum_{i=1}^{N_{\mathrm{SPIN}}}\sum_{j=1}^{N_{\mathrm{SPIN}}} J_{ij}x_i x_j$. This modifies the Euler's method as follows:

$$x_i^{(m+1)} = x_i^m + \Delta y_i^m \delta_t\,, \tag{37}$$

$$y_i^{(m+1)} = y_i^m - [Kx_i^{(m+1)3}(t_{n+1}) + (\Delta - p(t_{n+1}))x_i^{m+1}]\delta_t\,, \tag{38}$$

$$x_i(t_{n+1}) = x_i^M\,, \tag{39}$$

$$y_i(t_{n+1}) = y_i^M - \xi_0 \sum_{j=0}^{N_{\mathrm{SPIN}}} J_{ij}x_j^M \Delta_t\,, \tag{40}$$

where $\delta_t = \frac{\Delta}{M}$, m varies from 0 to $M-1$, $x_i^{(0)} = x_i(t_n)$ and $y_i^{(0)} = y_i(t_n)$. In this way, the most computationally intensive part $\sum_{j=0}^{N_{\mathrm{SPIN}}} J_{ij}x_j$ is considered $\frac{1}{M}$ of the time with respect to the original algorithm, at the cost of reduced quality of the solution. The advantage with respect to the original algorithm is present if the number of position and momentum variables (with and without considering the problem contribution) updates are the same, as shown in Figure 10.

The presented architecture can be subdivided into two parts: one related to the **matrix-vector product** $\sum_{j=0}^{N_{\mathrm{SPIN}}} J_{ij}x_j$ and the other to compute the **position and momentum update** of each oscillator. The first, which is the most computationally expensive, was parallelized by subdividing this into smaller matrix products to be appropriately added and combined later, as shown in [52].

This architecture was scaled by exploiting multiple FPGAs in [50, 55]. Moreover, it was also an important reference for the design of other SBMs, whose oscillators network evolve in ergodic [21], discrete, or ballistic [20] ways. They differ from the original SB approach for an alternative evolution of the oscillators network, due to the usage of a different $p(t)$ (ergodic), the insertion of an inelastic wall (ballistic and discrete), even a discretization of the position variable (discrete). Also in these cases, hardware implementations of the machines have been obtained on the same FPGAs of the first SBM, even for specific applications involving QUBO models different from that of max-cut. For example, [53, 54] describe the hardware architectures of two ballistic SBM solvers of QUBO problems for trading strategies and portfolio optimization, synthesized on the Arria 10 GX 1150 FPGA, employing single-precision floating-point arithmetic components and involving 128 and 256 Ising variables, respectively.

Another architecture was proposed in [70] targeting the Ultra-Scale + XCZU7EV-2FFVC1156 FPGA. This was designed by re-writing the original SBa algorithm in a graph-based edge-centric description. Handling the algorithm as a set of operations on edges, it is possible to avoid the non-operations deriving from the consideration of all the J-matrix coefficients. This approach is effective when the problem of interest is sparse. Again, the optimizations considered as benchmarks are the max-cut problems, belonging to the Gset. The maximum number of Ising variables that can be handled by hardware is not explicitly written. However, it is possible to recognize

Just Accepted

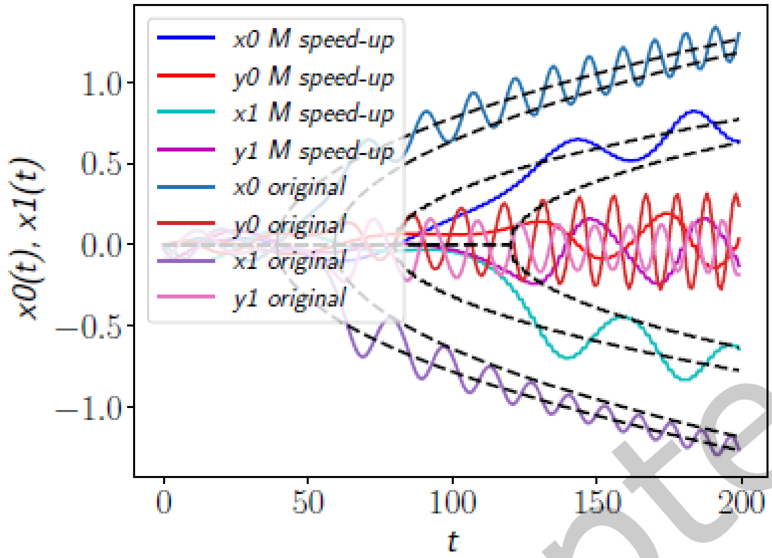


Fig. 7. Comparison of position and momentum evolution obtained with the original algorithm and by applying the splitting of Equation 36 with $M = 2$ in the case of a two-spin problem with an anti-ferromagnetic coupling, while considering the same number of position and momentum variables updated.

from the results that the most complex problem, solved without pre-processing, involves 1000 spins.

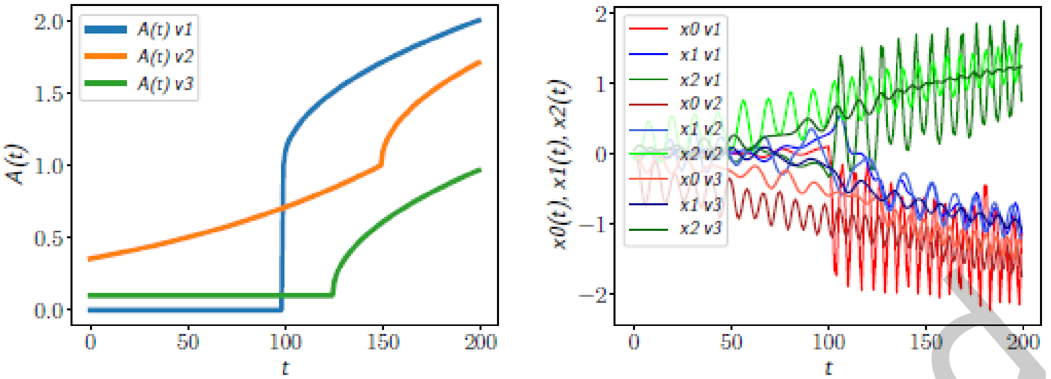
3 PROPOSED SIMULATED ADIABATIC BIFURCATION IMPLEMENTATION

This section presents the SBa digital solver and the associated toolchain, highlighting the unique aspects of this proposal compared to the current literature. In particular, Section 3.1 outlines the algorithmic novelties introduced by this work and the corresponding workflow. Section 3.2 illustrates the policy followed for parameter selection, while Section 3.4 describes the employed software model. Lastly, Section 3.5 showcases the designed architecture.

3.1 Research gap

A significant gap in the investigation of Simulated Adiabatic Bifurcation can be identified in the evaluation of its performance across various types of combinatorial optimization (CO) problems formulated using Ising or QUBO formulations. Indeed, the original algorithm and the Quantum Bifurcation machine were thought to solve only max-cut-like problems, i.e., problems without the external field component of the Ising formulation. As mentioned in Section 2.2, some techniques were proposed in the state-of-the-art, even if without providing many details about them. In this work, the implemented approach is based on introducing the $A(t)$ evolution parameter. In [46], it is suggested to have $A(t)$ close to zero for $p(t) \ll \Delta$ and close to $\sqrt{\frac{p(t)-\Delta}{K}}$ for $p(t) \gg \Delta$. In order to obtain an effective parameter evolution consistent with the provided indication, several trials were done. Initially, a naive piecewise-defined function was considered, fixed to zero until $p(t) = \Delta$ and equal to $\sqrt{\frac{p(t)-\Delta}{K}}$

Just Accepted



(a) The three considered $A(t)$ shapes

(b) Position and momentum variable evolutions of a three-oscillator network with the three considered $A(t)$ shape

Fig. 8. $A(t)$, position and momentum variables of three-oscillator network time evolutions. The three versions v1, v2 and v3 correspond to $A(t)$ of Equations 41, Equation 42 and 43, respectively.

Just Accepted

for $p(t) > \Delta$:

$$A(t) = \left\{ \begin{array}{ll} 0, & \text{if } p(t) < \Delta \\ \sqrt{\frac{p(t)-\Delta}{K}}, & \text{if } p(t) \geq \Delta \end{array} \right\} \tag{41}$$

The obtained results were not satisfactory. In particular, it was noticed that by exploiting the trivial solution, the contribution of the h vector was considered *less* than that of the J matrix and **too abruptly**. Therefore, it was considered to substitute the first part with an exponential, making the function continue, and translate the function along the x-axis by a:

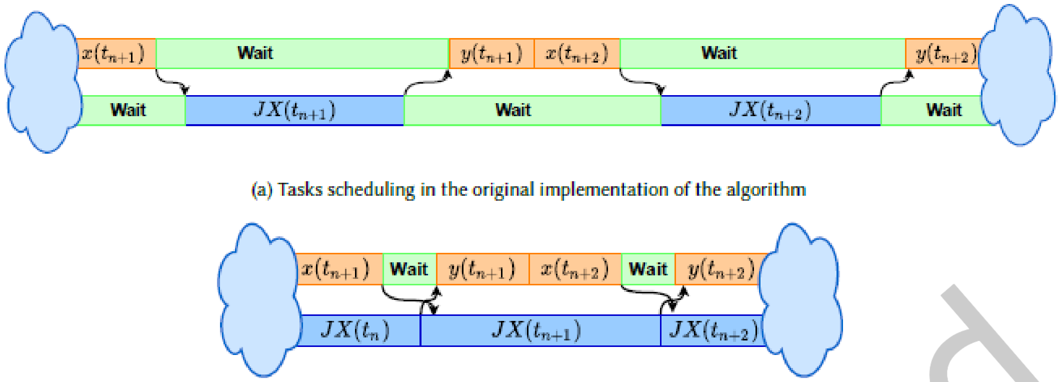
$$A(t) = \left\{ \begin{array}{ll} 2^{\frac{p(t)-\Delta-b}{K}}, & \text{if } p(t) < \Delta + b \\ \sqrt{\frac{p(t)-\Delta-b}{K}} + 1, & \text{if } p(t) \geq \Delta + b \end{array} \right\} \tag{42}$$

This modification resulted in a significant improvement. Moreover, analyzing the results obtained by gradually changing the b value, $b = 4K$ was empirically determined as the best choice. However, since the computation of an exponential function in hardware is costly, it was evaluated to substitute the first part with a constant function, different from zero:

$$A(t) = \left\{ \begin{array}{ll} 0.1, & \text{if } p(t) < \Delta + b \\ \sqrt{\frac{p(t)-\Delta-b}{K}} + 0.1, & \text{if } p(t) \geq \Delta + b \end{array} \right\} \tag{43}$$

This function can provide good results with a limited expense from the computation point of view, so it was chosen for the implementation.

On the other hand, it was noticed that the sum-product component $\sum_{j=1}^{N_{\text{SPIN}}} J_{ij} x_j(t_{n+1})$ of the oscillator's momentum update is the most computation-intensive step, also because data dependencies limit the possibility of parallelizing the computation. Indeed, as shown in Figure 9(a), its computation can start only after the position update of all oscillators. In order to reduce the overall momentum computational time, this work proposes an **approximation**



(a) Tasks scheduling in the original implementation of the algorithm

(b) Tasks scheduling with the proposed approximation

Fig. 9. Scheduling of the sum computation and position-variable update tasks in the original version of the algorithm and when applying the proposed approximation.

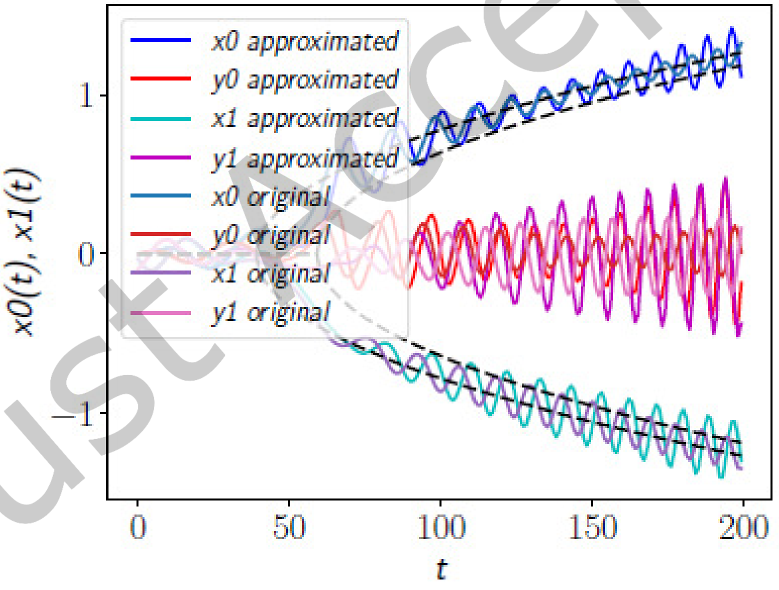


Fig. 10. Comparison of position and momentum evolution obtained with the original algorithm and by applying the proposed approximation in case of a two-spin problem with an anti-ferromagnetic coupling.

Just Accepted

that tries to **exploit the adiabaticity** of the evolution ($x(t_n) \approx x(t_{n+1})$). In particular, the momentum update expression was modified as follows:

$$
\begin{gathered}
y_i(t_{n+1}) = y_i(t_n) - [Kx_i^3(t_{n+1}) + (\Delta - p(t_{n+1}))x_i(t_{n+1}) + \xi_0 \sum_{j=1}^{N_{\mathrm{SPIN}}} J_{ij}x_j(t_{n+1}) + 2\xi_0 A(t)h_i]\Delta_t \\
\Big\downarrow \\
y_i(t_{n+1}) = y_i(t_n) - [Kx_i^3(t_{n+1}) + (\Delta - p(t_{n+1}))x_i(t_{n+1}) + \xi_0 \sum_{j=1}^{N_{\mathrm{SPIN}}} J_{ij}x_j(t_n) + 2\xi_0 A(t)h_i]\Delta_t \,.
\end{gathered}
\tag{44}
$$

As it is possible to notice from Figure 9(b), the proposed approximation allows a more efficient parallelization of product-sum operations and position-momentum update. In particular, the hardware block or the thread (if a GPU implementation was considered) is almost always active, reducing its impact on the overall execution time. In the original algorithm case, the time required for the overall execution grows as:

$$
O((T_{\mathrm{xupdate}} + T_{\mathrm{sum}} + T_{\mathrm{yupdate}}) \times \mathrm{NumIter}) \,, \tag{45}
$$

where T_{xupdate} is the time required for updating the position variables, T_{sum} the time required for product-sum operation, T_{yupdate} the time required for updating momentum variables, and NumIter the number of algorithm iterations. At the same time, with the proposed approximation, the time required grows as:

$$
O(\max(T_{\mathrm{xupdate}} + T_{\mathrm{yupdate}}, T_{\mathrm{sum}}) \times \mathrm{NumIter}) \,. \tag{46}
$$

Consequently, the proposed approximation guarantees, in the most crucial scenario, i.e., when $T_{\mathrm{xupdate}} + T_{\mathrm{yupdate}} < T_{\mathrm{sum}}$, the opportunity of reducing the amount of time required of a factor $(T_{\mathrm{xupdate}} + T_{\mathrm{yupdate}}) \times \mathrm{NumIter}$, independently from the methodology chosen for computing the trajectory variables and the product-sum operations. The advantages in the specific case of the proposed architecture will be discussed in detail in Section 3.5.

Furthermore, this work tries to propose a solution to another algorithm limitation. Considering that SBa provides, even though in a very reduced amount of time with respect to other approaches, an **approximation of the optimal solution**, the final energy is not guaranteed to be the global minima of the objective function. In this article, we show that performing, after the SBa execution, **few steps of a local exploration algorithm**, such as **hill-climbing** or **simulated annealing**, is a good strategy for achieving the optimal solution. In particular, the solution found with SBa is exploited as the exploration starting point. This could significantly improve the solution with a reduced effort because it allows the correction of single variables that could assume wrong values due to a not perfectly good choice of the algorithm parameters or some numerical errors (especially in fixed-point number representation) or the intrinsic nature of the algorithm, which identify the ground state of a Hamiltonian composed by other contributions in addition to those related to the optimization problem itself.

Figure 10 compares the position and momentum evolution obtained with the original SBa and the proposed approximation, for a two-spin problem with anti-ferromagnetic coupling (Equation 31). This problem is sufficiently simple for a visual and intuitive comparison between the canonical and approximated approaches. It is possible to observe that the approximation amplifies and delays the oscillations compared to those of the original algorithm. However, the bifurcations can be considered analogous, since in both SBa versions $x_0 > 0$ and $x_1 < 0$. Similar observations can be done with other problems. It can be concluded that, even though the performed approximation introduces noticeable changes in oscillations, the final result is not significantly affected, as the algorithm typically converges to the same final configuration.

Just Accepted

Just Accepted

3.2 Parameters choice

The **choice of the algorithm parameters** is one the most crucial points for SBa exploitation because a wrong choice can completely negate the possibility of obtaining a good solution. Some indications about this are available in the supplementary information file of [21].

Firstly, regarding the **initialization of position and momentum variables**, the suggested strategy is to set the position and momentum variables to 0 and to a random value in the range $[-0.1, +0.1]$, respectively. This strategy proved to be effective in the tests performed for the current work. However, this introduces an element of **randomness** in the algorithm (which would otherwise be **deterministic**). This implies that multiple executions of the algorithm are required to ensure obtaining a good solution. Therefore, this work also attempts to identify a good initialization of the variables to attempt a single execution of the algorithm. In particular, it has been noticed that **initializing all the position and momentum variables to 0, except** $y_{N_{\text{SPIN}}-1} = 0.1$, where N_{SPIN} is the number of oscillators, yields a final energy close to the average energy that can be obtained with the random starting condition. For this reason, this could be a **good starting point** if there are limitations on the number of possible runs.

Regarding the positive detuning frequency Δ and the Kerr oscillators parameter K, the suggestion is to set them to 1; it was verified that this assignment is reasonable in all the considered types of optimization problems. On the other hand, the time step Δ_t has to assume, in any case, a value lower than 1 to avoid algorithm divergence. However, a higher value of Δ_t assures, together with a faster evolution of the pumping amplitude, a faster convergence of the algorithm to a solution at the cost of a possible reduction of its quality. Therefore, the best choice depends on users' needs.

The choice of the value of ξ_0 is the most complex. Indeed, it is the only parameter strictly problem-dependent and plays a crucial role in weighting the contribution of the target optimization problem in the Hamiltonian with respect to the others. In this work, the same strategy for the choice of ξ_0 reported in the supplementary information file of [21] was chosen, on one hand for simplicity, on the other for its empirical validity. In particular, it is proposed to set ξ_0 for smaller problems to:

$$\xi_0 = \frac{0.7\Delta}{\sigma\sqrt{N_{\text{SPIN}}}}\,, \tag{47}$$

where σ is the standard deviation of the off-diagonal elements of J and N the number of spins, while for larger problems:

$$\xi_0 = \frac{0.5\Delta}{\sigma\sqrt{N_{\text{SPIN}}}}\,. \tag{48}$$

These equations were derived by considering that the first bifurcation point is given by the maximum eigenvalue $\lambda_{\max}$ of the J matrix. To reach this point, and consequently the approximate optimal solution, as fast as possible, the first bifurcation point is set to 0 by setting $\xi_0 = \frac{\Delta}{\lambda_{\max}}$. Assuming, for sake of simplicity, that J is a random symmetric matrix (clearly not true in a real scenario), $\lambda_{\max} \approx 2\sigma\sqrt{N_{\text{SPIN}}}$ for the Wigner's semicircle law [9]. Consequently, the optimal setting for the variable is given by Equation 48, but for obtaining a faster convergence, it is also possible to use a larger ξ_0 (Equation 47). The obtained ξ_0 is not always the best possible, because the relation is obtained by making some assumptions about the J matrix, but, on average, it provides good results.

The main challenge in this context concerns the extension of this rule to the case in which the **external field** of the Ising computation is considered. Firstly, it was decided to provide the highest possible flexibility through separate ξ_0 parameters for the J and h contributions:

$$y_i(t_{n+1}) = y_i(t_n) - [Kx_i^3(t_{n+1}) + (\Delta - p(t_{n+1}))x_i(t_{n+1}) + \boldsymbol{\xi_0}\sum_{j=1}^{N_{\text{SPIN}}} J_{ij}x_j(t_n) + 2\boldsymbol{\xi_{0_h}}A(t)h_i]\Delta_t\,. \tag{49}$$

In the previous equation, ξ_0 and ξ_{0_h} are associated with J and h, respectively. This strategy allows the compensation of, eventually, a strong imbalance in the values (e.g. a different order of magnitude) of the coefficients of the J matrix and the h vector. In any case, by default, they are considered equal, and to take into account the effect of the external field, the standard deviation is also computed considering its coefficients as if they were the diagonal elements of the J matrix.

Another important parameter that needs to be managed is the slope (s_p) with which the variable $p(t)$ grows and, consequently, the number of algorithm interactions required (NumIter). The latter clearly depends not only on the dimension of the problem under analysis but also on its characteristics. For example, max-cut problems can obtain a good solution with a faster convergence than knapsack problems. This can be explained by remembering that the second type of problem also has the h component to handle, and the coefficients involved usually differ much more from each other than in the max-cut case. In general, in this work, NumIter and s_p are chosen to achieve a final p-value in the range $[2, 10]$, and the best combination is found by performing small variations in their values until a good quality of the solution is reached for one problem of the group under analysis.

3.3 Toolchain structure

This article presents the toolchain shown in Figure 1. In particular, it consists of software models written in the **Cython language**, featuring a C++ core with an external Python interface. These models were utilized for studying the mechanism of managing degrees of freedom, evaluating the effectiveness of the proposed improvement, and identifying the **optimal number representation** for architecture, using a set of well-known problems as benchmarks. The software models can also be used independently as software solvers and come with a user-friendly interface. Leveraging the insights gained from software tests, an architecture was designed and described in **VHDL**, **generic for the number of bits in the number representation and the number of emulable oscillators**. To ensure its correctness, each block was initially tested in isolation using Python scripts simulating the roles of other components. Subsequently, a Python class was developed to automatically generate suitable testbenches and **Tool Command Language** (**TCL**) scripts for solving each problem and launching the simulation in **ModelSim 17.0**.

The architecture was synthesized by changing its degrees of freedom, such as the number of bits for number representation and the number of emulated oscillators, to analyze their impact on **area requirements**, **maximum frequency**, and **power consumption**. To streamline this process, a Python script was employed to generate TCL scripts and initiate synthesis using **Intel Quartus Lite 17.0**, targeting the **5CSEMA5F31C6 Cyclone V FPGA** on the **DE1 SoC board**.

For **onboard functional testing**, the **Nios II lite processor** was exploited for sending to the architecture parameters and problem coefficients at the beginning of the execution and for receiving the final optimal solution from the FPGA.

Each component of the toolchain is elaborated upon in the following sections.

3.4 Software models

Proper software models were developed to study the algorithm characteristics and evaluate the proposed approximation, considering the M-speedup, and to verify the possibility of computing the oscillators' evolution with fixed-point number representation.

In particular, the following implementations were obtained:

- **SBa**: floating-point implementation of the original algorithm;
- **SBaFixed**: fixed-point implementation of the original algorithm;
- **SBaVLSILab**: floating-point implementation of the algorithm by applying the proposed approximation;
- **SBaVLSILabFixed**: fixed-point implementation of the algorithm by applying the proposed approximation;

Just Accepted

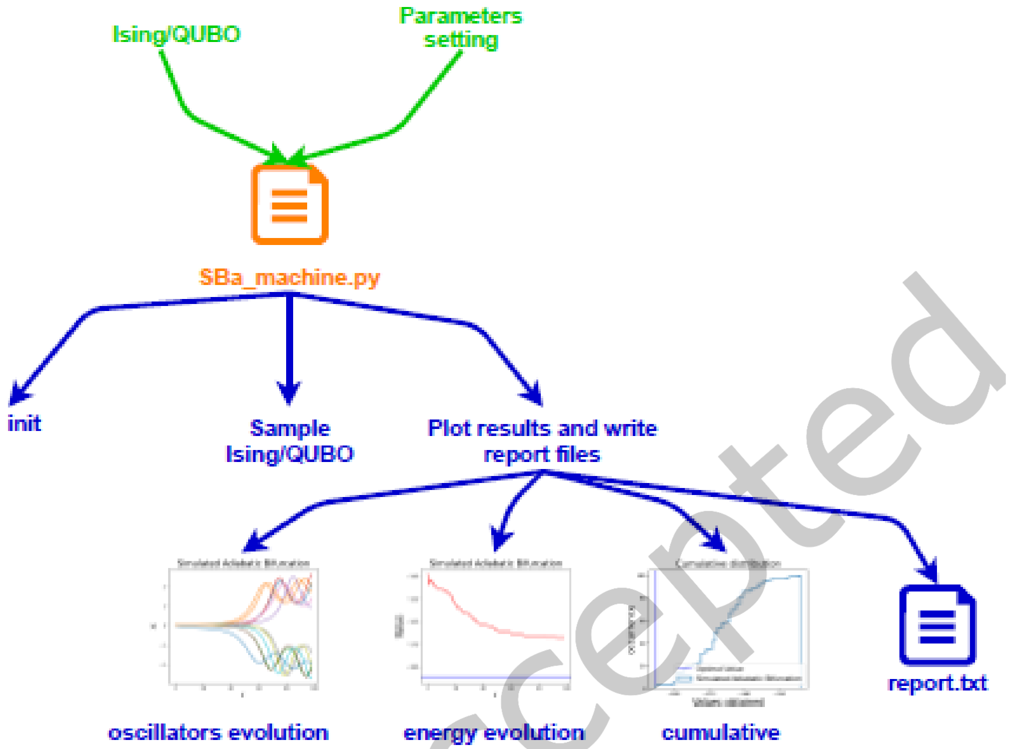


Fig. 11. Graphical description of the class exploited to execute the solver. Green arrows indicate the required input information, i.e. the Ising or QUBO description of the target optimization problem and the parameter values. If the second input is not provided, default parameters are assumed. The blue arrows represent the available methods within the class: the `init` create the software object, the `Sample_Ising` or `Sample_QUBO` executes the algorithm and the others provide a report on the algorithm's execution along with a set of plots detailing aspects of the solver execution, such as the evolution of oscillators and energy or the cumulative distributions of the final output.

- **SBaTo**: floating-point implementation of the algorithm by applying the M-speedup;
- **SBaToFixed**:fixed-point implementation of the algorithm by applying the M-speedup;
- **SBaTM**: floating-point implementation of the algorithm by applying both the proposed approximation and the M-speedup;
- **SBaTMFixed**: fixed-point implementation of the algorithm by applying both the proposed approximation and the M-speedup.

The floating-point number representation was considered to study the algorithm and behavior of all its possible versions without stringent limitations on the range of values assumable by the algorithm parameters, to examine the problem coefficients, and to evaluate intermediate results for debugging purposes. The fixed-point implementations were obtained because this number representation usually permits the design of a less expensive hardware design from the area's point of view and with lower latency. Therefore, their functional verification is crucial for achieving the best possible hardware design. In this implementation, numbers are represented as standard integers, where the Least Significant Bit (LSB) has virtual weight $2^{-\text{NumFrac}}$, where NumFrac is the

Just Accepted

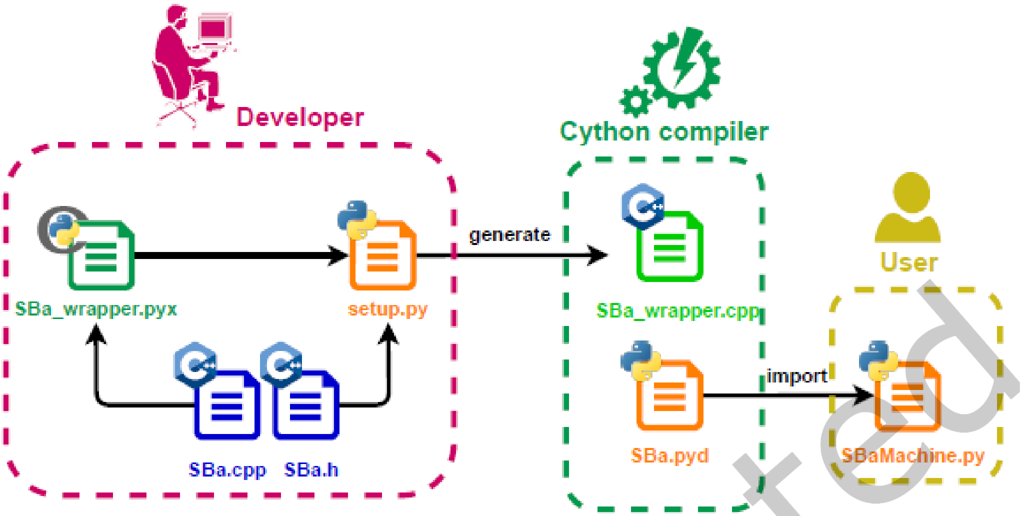


Fig. 12. Graphical representation of the procedure for obtaining a Python module with a C/C++ core by exploiting the Cython language. First, the developer writes the C/C++ function of interest (.cpp and .h) and the Pyrex file with the header file containing the declaration of the function of interest (.h). In the Pyrex file, the target function has to be translated into Cython syntax, and the C++ function must be called within a wrapper function that converts its inputs from Python data structures to C++ types and converts the returned from C++ types to Python data structures. Then, the setup.py file, which specifies the name of the Python extension to obtain and the C++ and Pyrex files to consider, is exploited for compiling the C++ source with Cython. All the files produced by the developer are shown in the magenta box. The compilation produces an optimized translation of the wrapper (_wrapper.cpp), and the Python module to import in other scripts (.pyd). These files are shown in the green box. The obtained module can be imported by the user as a standard Python module, as illustrated in the orange box.

Just Accepted

number of bits considered for the fractional part representation. A right shift is performed to keep constant the number of bits of the representation after complex numerical operations such as multiplication and square root. Each solver is defined inside a Python class, which permits the definition of the algorithm parameters, running the solver for a specific problem written in Ising or QUBO formulation with a certain number of iterations for a certain number of times, and analyzing the obtained results. This analysis is possible through a written report and some plots, such as the graph of the position and momentum variables' evolutions, the plot with the cumulative distribution of the final energies, and another showing the energy evolution as a function of the iteration (Figure 11). The choice of Python language for the solver class came from the need to interface the solver with the main libraries for Ising/QUBO problems, including qubovert [26], PyQUBO [25, 63] and dimod [2].
However, the algorithm core is written in C++ language to have an efficient execution of the most complex computational part. The interface between the Python part and the C++ part is realized by exploiting Cython, a **programming language** and an **optimizing static compiler** that allows the writing of C/C++ extensions for Python and the declarations of static type declarations in Python code. It is based on Pyrex, a programming language for writing Python extension modules obscuring the Python/C API to the user. The following procedure, shown in Figure 12, must be followed to exploit this mechanism:

- Wrap the C++ functions (.cpp, .h files) in a Pyrex file (_wrapper.pyx file), which includes the header file containing the declaration of the function of interest (.h). In this file, translate the function into Cython

Algorithm cycles	
Multiplication	$(11 \times N_{\mathrm{SPIN}} + N_{\mathrm{SPIN}} \times (N_{\mathrm{SPIN}} - 1)) \times \mathrm{NumIter}$
Addition/subtraction	$(8 \times N_{\mathrm{SPIN}} + N_{\mathrm{SPIN}} \times (N_{\mathrm{SPIN}} - 1)) \times \mathrm{NumIter}$
sign determination	N_{SPIN}
Square-root	NumIter
Comparisons	NumIter

Table 1. Operations required for executing NumIter number of iterations of the SBa of N_{SPIN} variables problem.

syntax, and call the `C++` function within a wrapper function that converts its inputs from `Python` data structures to `C++` types and converts data returned from `C++` to `Python` types.
- Use the `setup.py` file, which specifies the name of the `Python` extension to obtain and the `C++` and `Pyrex` files to consider, for compiling the `C++` source with `Cython`. Its execution generates an optimized translation of the wrapper (`_wrapper.cpp`), and the `Python` module to import in other scripts (`.pyd`), in this case, in the solver class.

The software implementations are not parallelized using multi-thread or multiprocessing programming, as required by the algorithm's nature due to the limited resources of the devices on which they were executed. However, the `C++` description of the algorithm core would permit the partial parallelization with a limited amount of changes in the code, e.g. by adding, for example, the `#pragma omp for` option before the variables update loop.

3.5 Architecture

This section describes the **digital design** obtained for a Simulated Bifurcation machine, with detailed comments on the architectural choices. Further details useful for improving or modifying the **open source architecture**, such as Datapaths (DPs), Data Flow Graphs (DFGs) and Control Units (CUs) of single blocks, are provided in the Supplementary Materials files.

Before delving into the architecture description, an analysis of **algorithm complexity** is presented and discussed. Table 1 outlines the number of operations required to execute NumIter iterations of the algorithm for an N_{SPIN} variables problem. It can be observed that the algorithm complexity grows as $O(\boldsymbol{N^2_{\mathrm{SPIN}}} \cdot \textbf{NumIter})$. This indicates that, without parallelization, the time required for a single algorithm iteration **increases quadratically with the problem dimension** and the number of required iterations rises rapidly with the number of oscillators to mimic. Particularly, drawing parallels with quantum annealers, the expected growth of the required number of iterations (NumIter) is proportional to $10^{\sqrt{N_{\mathrm{SPIN}}}}$ [37]. This underscores the importance of obtaining implementations that allow **parallelization**, such as FPGA, GPU or ASIC designs, leveraging the **reduced amount of data dependencies** present in the algorithm. Indeed, the update of position and momentum variables can be entirely parallelized, with the $\sum_{j=1}^{N_{\mathrm{SPIN}}} J_{ij} x_j(t_n)$ contribution being the unique dependency among them.

This work proposes the architecture reported in Figure 13, aiming to **parallelize the algorithm as much as possible with an area growth slightly more than linear with the number of binary variables**. Its main blocks are:

- **Processing elements** (**PEs**): one for each oscillator to be emulated, computing its position and variable update.
- **Sum** blocks: one for each oscillator to be emulated, with the purpose of computing $\sum_{j=1}^{N_{\mathrm{SPIN}}} J_{ij} x_j(t_n)$.
- **P A updater** block: evaluates the $p(t)$ and the $A(t)$ evolution parameters in each iteration;
- **Control Unit** (**CU**): synchronizes the overall system.

Just Accepted

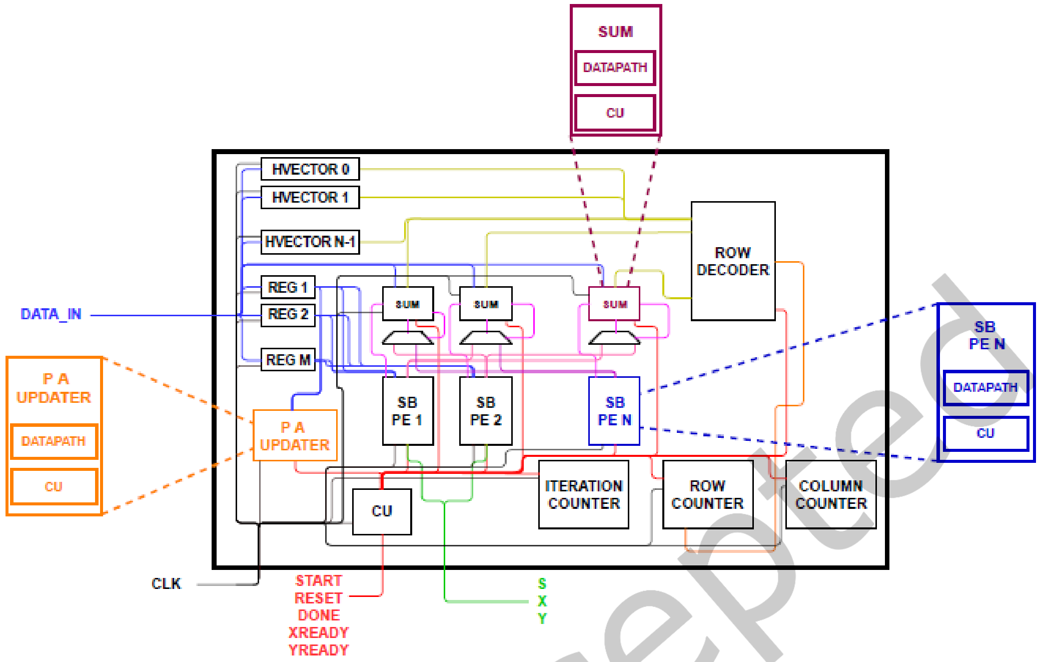


Fig. 13. Architecture proposed for digital implementation of SBa.

Each PE computes the position and momentum variables of one spin following Equations 29 and 44. The summation $\sum_{j=1}^{N_{\text{SPIN}}} J_{ij} x_j(t_n)$ is computed for each PE by the Sum block on top, which can be considered as a **hardware accelerator** for the PEs.

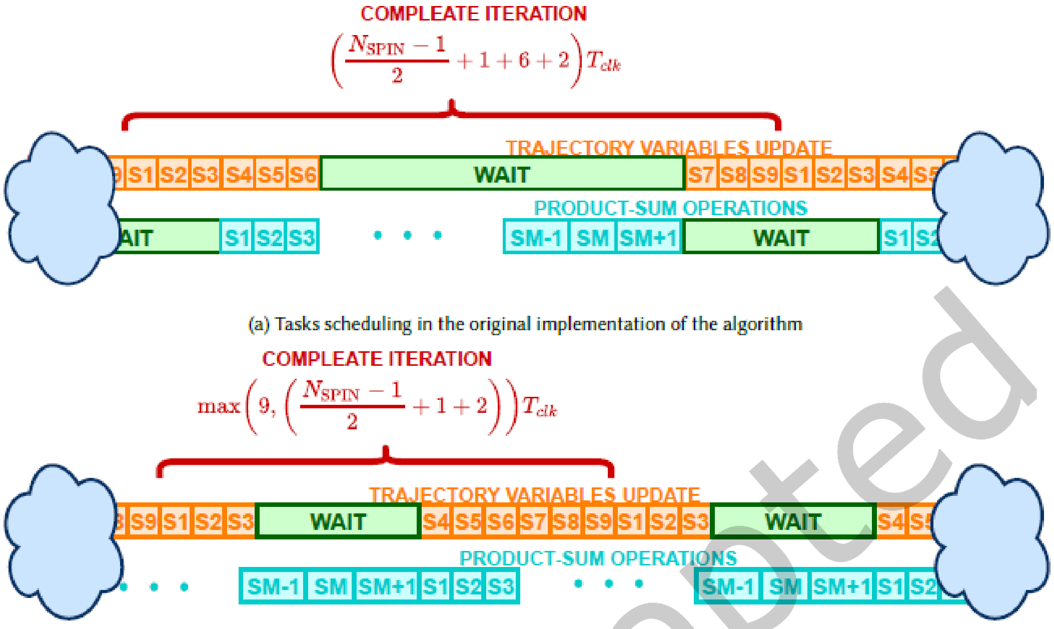
The P A updater block evaluates the evolution of the p and A parameters during the algorithm's execution. For p, a simple addition suffices, while for A, the **square root operation** is required. Although this operation may seem problematic at first glance, as it requires several clock periods if pipelining is applied, it is performed in parallel with the operations of the PEs and Sum blocks, thus **not limiting the total latency of the system**.

The **Control Unit** (**CU**) block synchronizes the overall system by initiating the start and new iteration signals to the CUs of the other blocks and collecting the done signals. The finite-state machine of the top-level CU evolves after the assertion of an **external signal**.

As evident from Figure 13, besides the presented blocks, column and row counters with a decoder are required. These serve two purposes: ensuring that each sum block acquires the required coefficients at the outset, and correctly sampling the single-spin constraint and the initialization values of the momentum variables. Additionally, an iteration counter is necessary to track the number of iterations performed.

It is apparent that the Sum block is **critical** in the design, as it can introduce significant **latency** if the sums are computed in series with a **multiply-and-accumulate** (MAC) [67] policy or substantial **area** overhead if parallel computation is implemented with structures like **trees of adders** [47, 68]. In both cases, this component can be considered the **bottleneck** in terms of scalability or latency. To make a design choice, the architecture's complexity was evaluated for both scenarios, as shown in Table 2. Choosing the first option provides a **linear area growth** with the number of oscillators to emulate ($O(N_{\text{SPIN}} \times \text{NumIter})$), without considering the problem's data storage elements, whose size does not depend on the implementation. Conversely, the second option increases

Just Accepted



(a) Tasks scheduling in the original implementation of the algorithm

(b) Tasks scheduling with the proposed approximation

Fig. 14. Scheduling of the sum computation and position-variable update tasks in the original version of the algorithm and by applying the proposed approximation, considering the proposed architecture.

Just Accepted

the area quadratically with the number of spins ($O(N^2_{\text{SPIN}} \times \text{NumIter})$). Moreover, this option is also critical for what concerns memory access. For these reasons, this work implements the Sum block by exploiting a MAC approach.

The presented architecture was described in VHDL (IEEE 1164) and, for **maximum flexibility**, employs generic descriptions for the integer (INT) and fractional (FRAC) part number of bits, the stopping value of the iteration counter (N_ITERATION), the number of spins (N_SPIN), and the number of bits required for iterations N and row/column counter M = $\log_2$(N_SPIN), respectively.

The Simulated Bifurcation Machine receives the algorithm parameters, the variables' initial state, and the problem coefficients at the beginning of the algorithm execution, one by one. Providing the initial states as an input and generating them in software — for example, with a connected processor — is chosen due to the difficulty of generating high-quality random numbers in hardware. Moreover, allocating dedicated hardware for an operation required only at the beginning of the algorithm is not considered efficient from a hardware resources standpoint. To simplify the computation of an algorithm iteration, recurring and constant products such as $\xi_0 J$ and $\xi_{0_h} h$ products are precomputed in software to avoid numerical cancellations. This allows for significant savings in operations. The number of clock cycles required for the completion of a single iteration is determined by the maximum value between:

- 9, which is the number of cycles required by a single PE if it does not have to wait for the results from Sum and P A updater blocks;

Processing elements	
Control units	N_{SPIN} or 1
Multipliers	$2 \times N_{\text{SPIN}}$
Adder/subtractors	N_{SPIN}
$p(t)$ and $A(t)$ updater	
Control units	1
Multipliers	1
Adder/subtractors	1
Square-root	1
Sums Spins - multiply and accumulator	
Control units	N_{SPIN} or 1
Adder/subtractors	N_{SPIN}
Carry Save Adder	$2 \times N_{\text{SPIN}}$
Multipliers	$2 \times N_{\text{SPIN}}$
Counters	N_{SPIN} or 1
Register file	N_{SPIN}
Sums Spins - tree of adder	
Control units	N_{SPIN} or 1
Adder/subtractors	N_{SPIN}
Carry Save Adder	$N_{\text{SPIN}} \times (N_{\text{SPIN}} - 2)$
Multipliers	$N_{\text{SPIN}} \times N_{\text{SPIN}}$
Register file	N_{SPIN}
Outside of blocks	
Control units	1
Counters	3
Decoder	1

Table 2. Hardware required for executing the Simulated Adiabatic Bifurcation of N_{SPIN} variables problem considering both multiply-and-accumulate and tree of adder implementations. As for the Control Units of each block, it is reported as either 1 or N_{SPIN}, as it is possible to share the control unit among blocks of the same type or consider one for each. The second choice is better from a routing perspective, while the first is better for reducing the number of required logic elements.

Just Accepted

- $\left\lceil \frac{N_{\text{SPIN}}-1}{2} \right\rceil + 1 + 2$, with N_{SPIN} number of variables, $\left\lceil \frac{N_{\text{SPIN}}-1}{2} \right\rceil$ number of MAC operations, 1 associated with the last sum, and 2 for waiting for the x variable sampling and the results sampling;
- $\left\lceil \frac{N_{\text{bit}}}{2} \right\rceil + 5$, where $\left\lceil \frac{N_{\text{bit}}}{2} \right\rceil$ is the number of clock cycles required by the pipeline square root module, 4 is the number of clock cycles in which the P A updater block performs the other operations for p and A update, and 1 p and A sampling.

Figure 14 shows the advantage of the proposed approximation in the designed architecture. The analysis assumes that the number of spins N_{SPIN} is sufficiently high that the execution time of the block computing matrix-vector products is higher than those for $p(t)$ and $A(t)$ updates.

In the original version of the algorithm, due to data dependencies, trajectory variable updates can be done in parallel to product sum operations for at most three steps of the data flow graph, which must be reorganized to increase the distance between the computation of positions and the evaluation of the sum blocks' outcome.

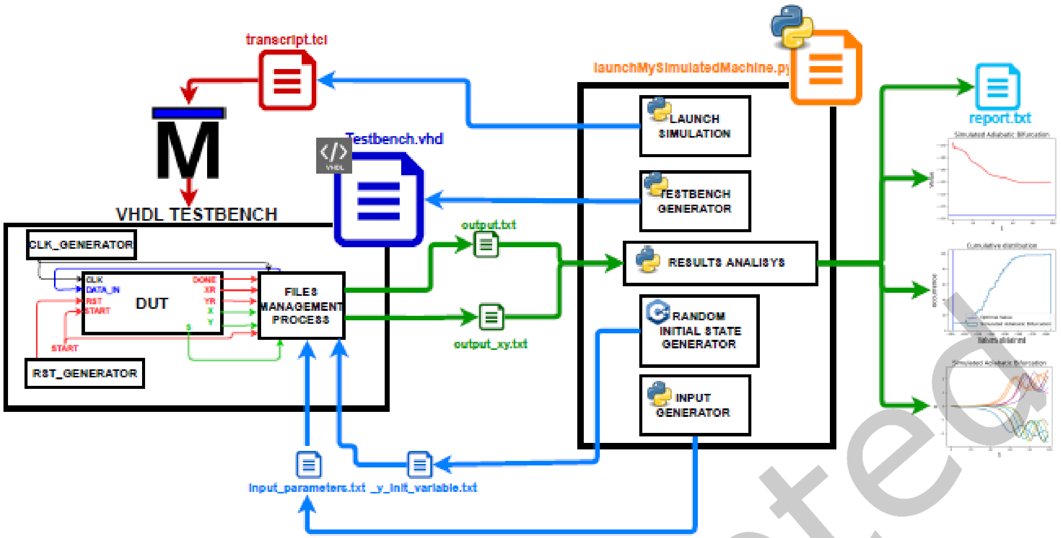


Fig. 15. Scheme of the simulation automatization.

The current version of the graph is shown in the Supplementary Materials document. For these reasons, the execution time grows as:

$$T_{\text{exec}} = \left(\frac{N_{\text{SPIN}} - 1}{2} + 9 \right) \times \text{NumIter} \,. \tag{50}$$

On the other hand, Figure 14(b) demonstrates that the proposed approximation reduces the execution time to:

$$T_{\text{exec}} = \max \left(9, \left(\frac{N_{\text{SPIN}} - 1}{2} + 3 \right) \right) \times \text{NumIter} \,. \tag{51}$$

While this may seem like a small savings, its cumulative impact is significant.
Finally, the architecture produces a spin configuration in output. The corresponding energy state can then be computed in software, similar to the generation of random numbers mentioned earlier. Additionally, as previously mentioned, the solution can be improved by exploiting local search mechanisms like hill climbing and simulated annealing.

Each component of the architecture was initially simulated alone to verify its functionality. Python scripts were used to emulate the other blocks, along with all their outputs and stimuli. Subsequently, the overall architecture underwent functional testing through wave simulations with **Modelsim 17.1** and on-board testing by interfacing with the **Nios II processor** for acquiring algorithm parameters and reading the final solution obtained. Both types of tests are elaborated below.
For wave simulation automation, a Python class associated with the test was implemented, as depicted in Figure 15. This class includes methods for generating:

- the testbench file (`testbench.vhd`), in which the **Device Under Test** (**DUT**) is stimulated by reading a parameter and a variable initialization file;
- the TCL file (`transcript.tcl`), required for running the simulation via command line;

Just Accepted

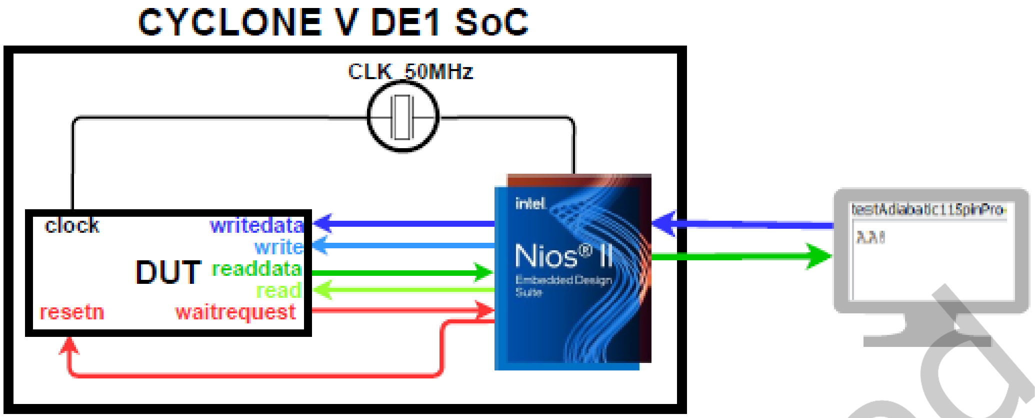


Fig. 16. Architecture interface with Nios II processor.

- the stimuli files, with the algorithm parameters and problem coefficients, and the other containing variable initialization values.

Moreover, the same class is responsible for launching the test and analyzing the obtained results. A Cython module writes the variables initialization file to generate the random numbers in C++ (that is a more efficient programming language for this task), providing higher randomness for the inputs.
The top entity employed for the generated testbench has three input signals:

- `Data_in`, represented on INT+FRAC number of bits, used to acquire parameters and problem coefficients at the beginning;
- `start`, asserted to initiate computation;
- `reset`, for clearing registers at the beginning.

The output signals are:

- S, representing the final spin configuration with a number of bits equal to N_{SPIN} (the problem's size);
- done, asserted at the end to indicate the completion of computation;
- the `X` and `Y` probes report the position and momentum variables assumed by oscillators in each iteration, facilitating observation of evolution, when `XReady` and `YReady` are asserted.

The results are compared with those of the software model to ensure correctness of the description.
The FPGA considered for synthesis was **5CSEMA5F31C6** of the **Cyclone V System on Chip** (SoC) **Altera family**, integrated on the **DE1-SoC** board. The device includes **85000 programmable logic elements** (LE), making it a small but practical board for prototype development. The DE1-SoC board can be programmed using the **Intel Quartus** software associated with the **Modelsim simulator**. In this work, **Intel Quartus lite 17.1** was employed. For functional tests, the architecture was interfaced with **Nios II/e soft-processor**, a Reduced Instruction Set Computer (RISC) 32-bit general-purpose soft processor synthesizable on Altera FPGAs. The main advantage of employing this processor is that it is possible to implement extra logic in addition to its basic architecture and some features not necessary for the application can be removed, thus guaranteeing a high degree of flexibility. In this case, the processor role is minimal: it sends inputs to the architecture and receives outputs; consequently, a simple implementation is sufficient. Moreover, the Nios processor can be synthesized in **many Altera FPGAs**, also those not SoC, enhancing portability.

Just Accepted

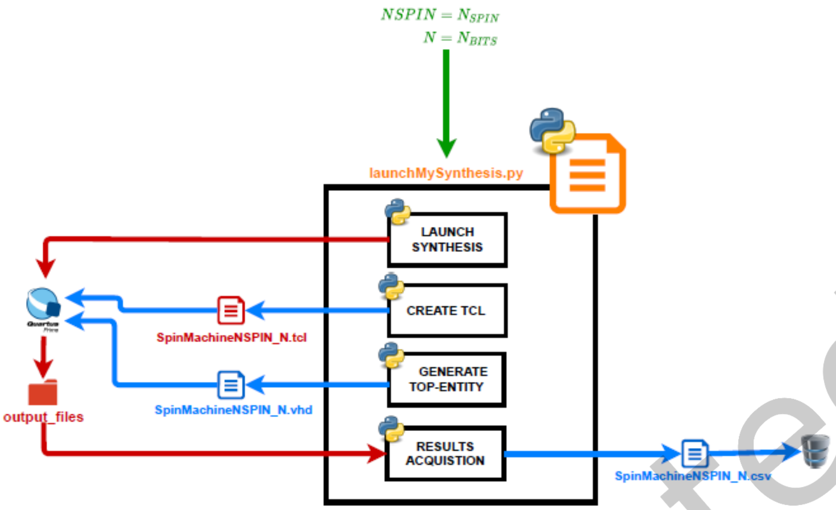


Fig. 17. Scheme of the synthesis automatization.

In this work, the architecture was synthesized as a **hardware accelerator** of the Nios processor, exploiting the possibility of defining custom instructions. The **Avalone interface** was employed for connecting the two elements in the FPGA, as shown in Figure 16. In order to implement the required interface, the architecture top entity has the following inputs:

- `writedata`, with size 32 bits, employed for receiving from the processor the algorithm parameters and the problem coefficients;
- `write` is a 1-bit signal, which indicates that the processor is sending new data;
- `read` is a 1-bit signal, which indicates that the processor can read the new data;
- `resetn`;
- `clock`;

and the following outputs:

- `readdata`, with size 32 bits, employed for sending to the processor the final spins configuration;
- `waitrequest` is a 1-bit signal exploited by architecture for informing the processor about the availability of new data to send.

Correctness was established by comparing the results obtained by running the algorithm onboard and using the software model with the same parameters and initial spin configuration.
After architecture validation, it was synthesized by varying the number of bits employed for representation (N) and the number of considered oscillators (N_{SPIN}) to observe the architecture scaling. For efficient execution, a Python class was developed to automate this process, as illustrated in Figure 17. The interface of the top-entity considered with this synthesis is the same considered for interfacing the custom architecture with a processor.
Before concluding, it is important to note that synthesis is only performed when the architecture needs to be changed in terms of the number of oscillators to be emulated and data width (e.g., the number of bits for fixed-point representation). Once the hardware is consolidated, compiling the Nios II firmware is the only required step before solving a given problem.

Just Accepted

4 RESULTS

This section reports some relevant software results (others are included in the Supplementary Materials files), simulations results and synthesis results. The benchmark problems considered are those presented in Section 2.1.2. All the other results not reported in the following are available in the supplementary information file. The codes employed for obtaining the reported results are available in the GitHub repository associated with this project.

4.1 Software results

This section reports the results obtained with software models presented in Section 3.4 by considering as benchmark problems those of Section 2.1.2.

4.1.1 Setup overview. All the tests were performed by exploiting the `Cython` implementations of software models (Section 3.4). The different versions of the algorithm were tested under equivalent conditions, meaning they were provided with the same algorithm parameters and number of iterations for each optimization problem considered. All the benchmark problems have been written with the qubovert `Python` library. Tests were executed on a single-process Intel(R) Xeon(R) Gold 6134 CPU @ 3.20 GHz opta-core, Model 85, with a memory of about 103 GB [1]. Each solver was executed on the same optimization problem one hundred times to extract statistics on its effectiveness in solving it.
The benchmark problems considered are those described in 2.1.2. For the problems automatically generated from a generic QUBO description, which involved defining the size and randomly generating some elements (e.g. the weights of edges in the max-cut problem), the reference optimal values were obtained by solving each of them with the **qubovert simulated annealer** on an extremely long annealing duration, allowing reasonable trust in the obtained solution as the optimum. On the other hand, for the G-set and the 0/1 Knapsack set, the best-known solutions available in the state-of-the-art were considered as reference optimal values.
The number of iterations SB solvers allows reasonable trust in the obtained solution as the optimum.

4.1.2 Figures of merit. The quality of a solver can be evaluated in terms of **optimal value** (**opt**) reached, **average value** (**avg**), **probability** (p_{range}) of obtaining a value which is the optimum or lower than it by a certain percentage (p_c) and the **time-to-solution** (**TTS**), where:

- The **optimal value** is the best final value obtained by a solver in the one hundred runs.
- The **average value** is obtained by averaging the hundred obtained final values.
- The **optimal value single** is the value obtained by executing a single time the algorithm with the suggested initialization of the variables.
- The **optimal value hill-climbing** is the value obtained after the execution of some hill-climbing iterations on the **optimal value**.
- p_{range} is the **probability of obtaining final energy lower than**:

$$\text{val} = \text{opt} + \left|\text{opt} \cdot p_c\right|, \tag{52}$$

  where opt is the expected optimal value. It is possible to say that one solver is better than another if its p_{range} is higher. This metric is defined as:

$$p_{\text{range}} \triangleq \frac{n_{\text{in\_range}}}{n_{\text{tot}}} 100, \tag{53}$$

  where $n_{\text{in\_range}}$ is the number of times in which the solver achieved final energy lower than val, and n_{tot} is the number of runs. This metric can be appreciated by reminding us that its value is expected to increase with the number of iterations. For this reason, it is more useful to consider a *normalized* $\frac{p_{\text{range}}}{\text{NumIter}}$ for comparing problems of different sizes.

Just Accepted

- The **time-to-solution** (TTS) is usually employed in the state-of-the-art for identify the quantum speed up in algorithms like SBa [4, 23, 32, 37, 69] and is defined as the *time required to find a target solution*, e.g. the optimum or a sub-optimum, *with a certain percentage of confidence* p_{conf}, usually set to 99%. In particular, it can be computed as:

$$\text{TTS} = t_f \frac{\log\left(1 - p_{\text{conf}}\right)}{\log\left(1 - p_{\text{range}}(t_f)\right)}, \tag{54}$$

  where t_f is the algorithm execution time, $p_{\text{range}}(t_f)$ is the probability of finding energy lower than a certain value, executing the algorithm for a time t_f. In this work, for simplicity and to reduce the dependence of the result from the computational resources considered, the time t_f is measured in terms of the number of iterations. Moreover, some special cases were managed:
  - if p_{range} is equal to 0, TTS was computed by considering a p_{range} equal to 0.1%;
  - if p_{range} is equal to 100%, TTS was computed by considering a p_{range} equal to 99%.

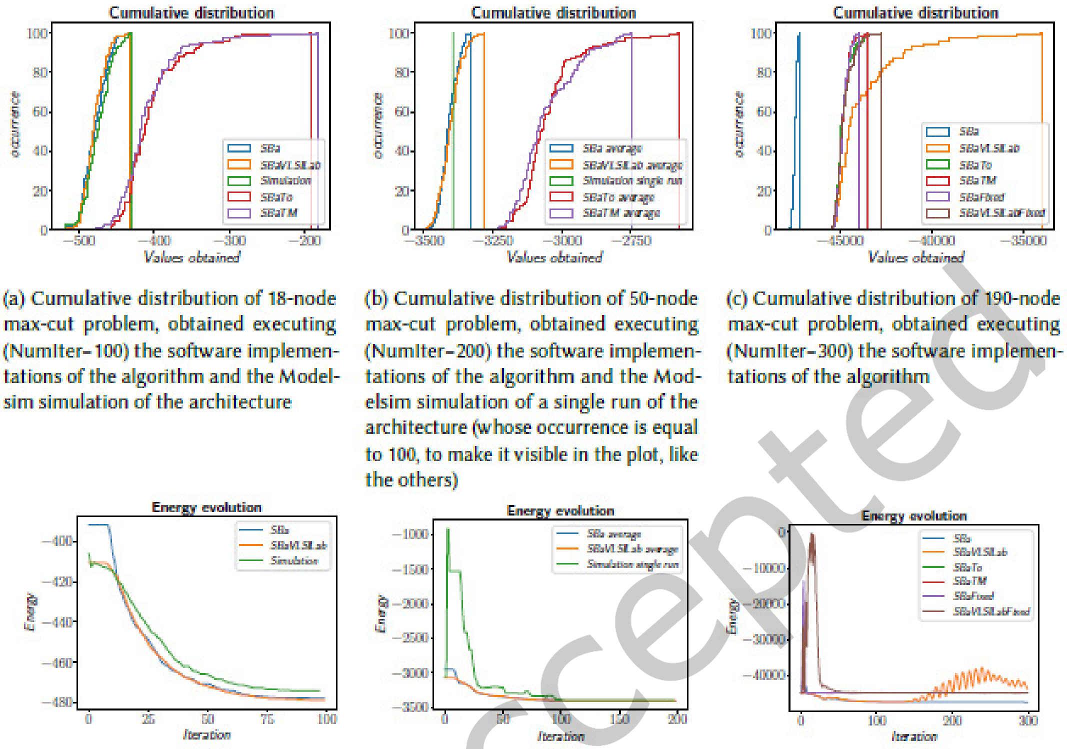
To better appreciate the variation of TTS and $p_{\text{range}}(t_f)$ metrics with the problem size, the moving average (MA) is applied to smooth the variation and to identify the trend easier:

$$\text{MA} = \frac{1}{w} \sum_{i=0}^{w-1} x[k-i], \tag{55}$$

where x is the array of data to be moving-averaged and $k \geq 1$ is the index of the last sample of the window. For $k < w$, the first element of x is replicated $w - k$ times.
In addition to these explicit figures of merit, it could also be interesting to observe the evolution of the x variables concerning SBa. In particular, this observation aims to verify the presence of bifurcation and ensure that variables do not diverge. Furthermore, it is essential to monitor the evolution of the average energy and the cumulative distribution obtained through multiple repetitions of the algorithm on a target optimization problem. These observations help in confirming that the algorithm parameters are correctly chosen.

4.1.3 Performed tests. This section presents and discusses some significant results obtained with the software implementations presented in Section 3.4 for each **benchmark** considered (Section 2.1.2). Additionally, a comparison between hardware and software results for the max-cut case is exemplified. Further results are provided in tables in the Supplementary Materials file.
Figure 18 reports examples of cumulative distributions and energy evolutions of max-cut problems generated with qubovert of sizes 18, 50 and 190-node, respectively. In particular, for the 18-node case, cumulative distribution and average energy evolution obtained by simulating one hundred times the proposed architectures are shown to demonstrate the consistency between software and hardware results. For the 50-node problem, results obtained from a single run — i.e., with the suggested variables initialization — of the simulated architecture are presented. It is possible to notice the behaviour of the original version of the algorithm (SBa), and the proposed approximated implementation (SBaVLSILab) are very similar. Moreover, the results obtained by simulating the proposed architecture, using fixed-point number representation, are comparable to the floating-point software results. Finally, it is shown that the implementations with the M-speedup is applied (SBaTo and SBaTM) perform worse than the original (SBa) and the approximated algorithm (SBaVLSILab).
Figure 19 shows the optimal value obtained with a single run (**opt single run**), one hundred runs (**opt multiple runs**), the average value (**avg multiple runs**) obtained in the multiple execution case and the energy obtained after the application of some hill-climbing iterations on the multiple run optimal solution (**opt SBa + HC**) for each software implementation considering G1, G2, G48, G55, G66, and G81 problems belonging to the G-set benchmark. Moreover, results obtained by running SBa for a larger number of iterations (SBalonge) are included. It demonstrates that the proposed approximation does not significantly affect the performance with this kind of

Just Accepted



(a) Cumulative distribution of 18-node max-cut problem, obtained executing (NumIter=100) the software implementations of the algorithm and the Modelsim simulation of the architecture

(b) Cumulative distribution of 50-node max-cut problem, obtained executing (NumIter=200) the software implementations of the algorithm and the Modelsim simulation of a single run of the architecture (whose occurrence is equal to 100, to make it visible in the plot, like the others)

(c) Cumulative distribution of 190-node max-cut problem, obtained executing (NumIter=300) the software implementations of the algorithm

(d) Energy evolution of 18-node max-cut problem, obtained executing (NumIter=100) the software implementations of the algorithm and the Modelsim simulation of the architecture

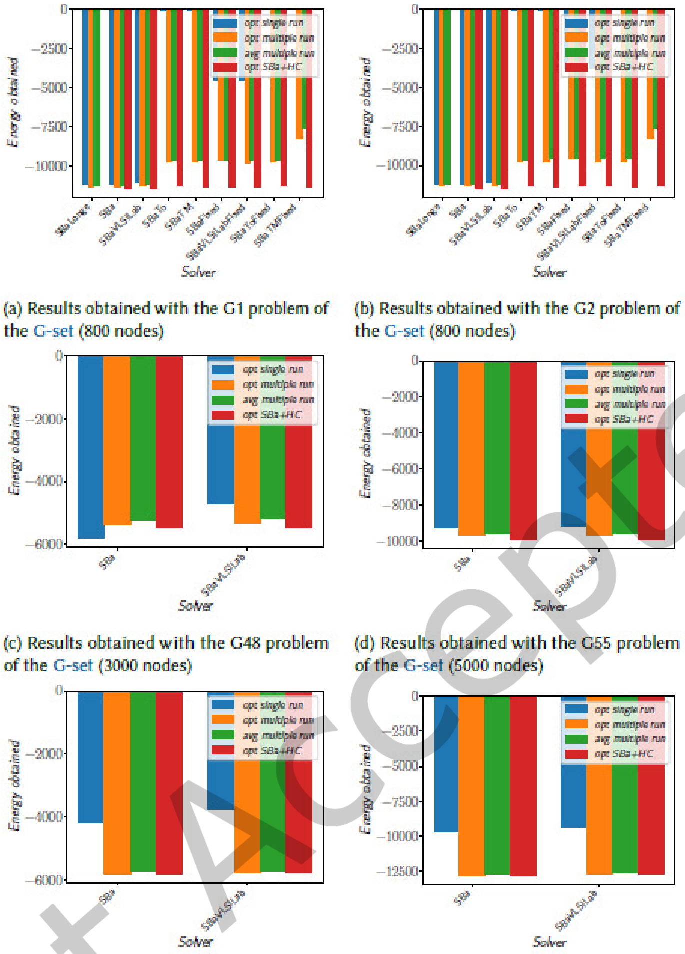
(e) Energy evolution of 50-node max-cut problem, obtained executing (NumIter=200) the software implementations of the algorithm and the Modelsim simulation of a single run of the architecture

(f) Energy evolution of 190-node max-cut problem, obtained executing (NumIter=300) the software implementations of the algorithm

Fig. 18. Cumulative distributions and average energy evolutions obtained by running one hundred times each software implementation on max-cut problems of different sizes. **SBa** is the floating-point implementation of the original algorithm. **SBaFixed** is the fixed-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation. **SBaVLSILabFixed** is the fixed-point implementation of the algorithm by applying the proposed approximation. **SBaTo** is the floating-point implementation of the algorithm by applying the M-speedup. **SBaTM** is the floating-point implementation of the algorithm by applying both the proposed approximation and the M-speedup.

Just Accepted

problem, as the energies obtained with SBa and SBaVLSILab solvers are very close. Moreover, it also validates the proposal of applying hill-climbing iterations on the final solution to improve its quality and indicates that the proposed initialization of the variables in the case of a single run is effective, except for the solver in which the M-speedup is applied. Instead, Figure 20 reports the cumulative distributions, energy and position variable evolution of G2 (800-node), G55 (5000-node), and G81 (20000-node) max-cut problem obtained by running one hundred times the software implementations. Again, Note that the results obtained with the original algorithm (SBa) and the proposed approximation (SBaVLSILab) are similar and better than the solutions obtained with



(a) Results obtained with the G1 problem of the G-set (800 nodes)

(b) Results obtained with the G2 problem of the G-set (800 nodes)

(c) Results obtained with the G48 problem of the G-set (3000 nodes)

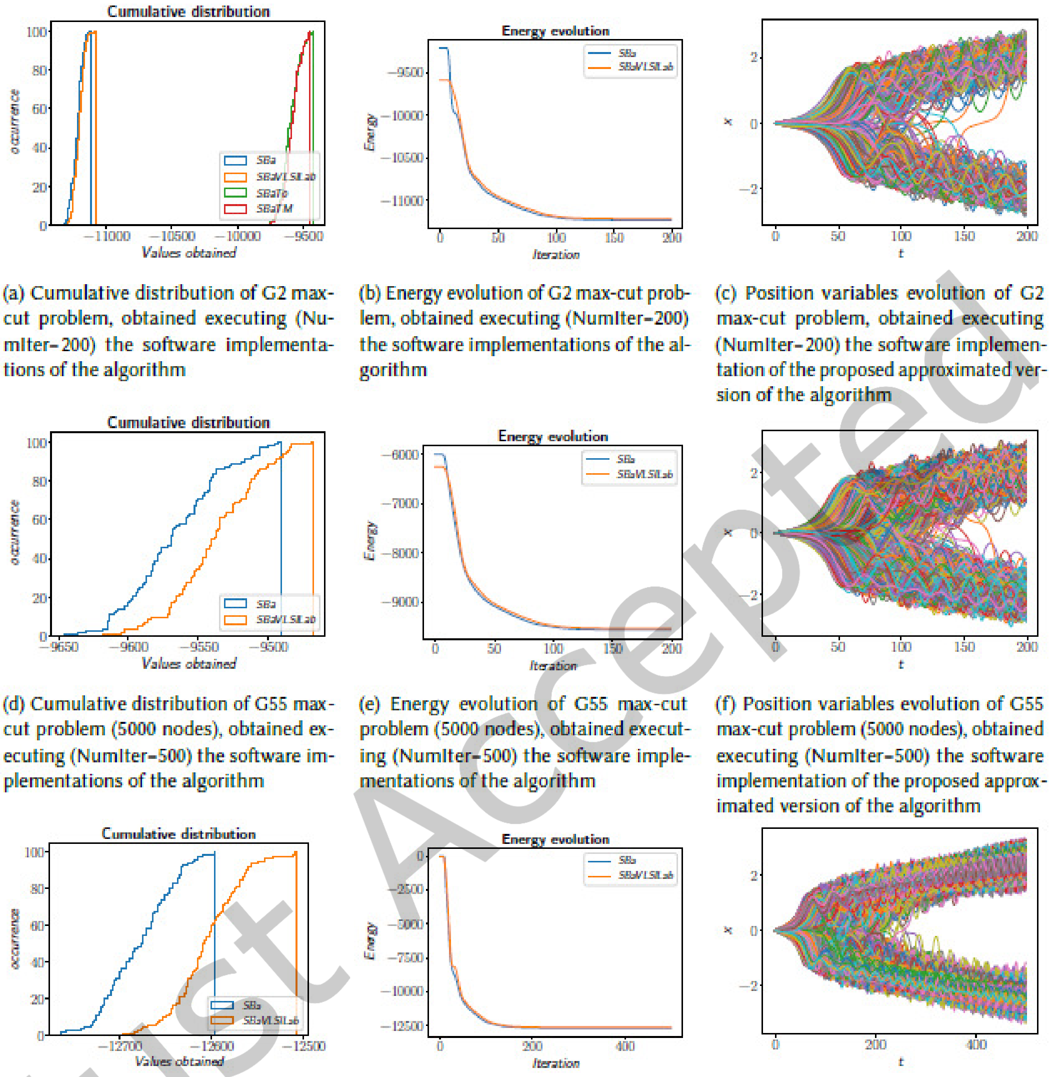
(d) Results obtained with the G55 problem of the G-set (5000 nodes)

(e) Results obtained with the G66 problem of the G-set (9000 nodes)

(f) Results obtained with the G81 problem of the G-set (20000 nodes)

Just Accepted

Fig. 19. Optimal value obtained with a single run (**opt single run**), one hundred runs (**opt multiple run**), the average value (**avg multiple run**) obtained in the multiple execution case and the energy obtained after the application of some hill-climbing iterations on the multiple run optimal solution (**opt SBa + HC**) are reported for each software implementation considering G1 and G2 problems of the G-set. **SBa** is the floating-point implementation of the original algorithm. **SBaLonge** is an SBa execution on a longer evolution time. **SBaFixed** is the fixed-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation. **SBaVLSILabFixed** is the fixed-point implementation of the algorithm by applying the proposed approximation. **SBaTo** is the floating-point implementation of the algorithm by applying the M-speedup. **SBaToFixed** is the fixed-point implementation of the algorithm by applying the M-speedup. **SBaTM** is the floating-point implementation of the algorithm by applying both the proposed approximation and the M-speedup. **SBaTMFixed** is the fixed-point implementation of the algorithm by applying both the proposed approximation and the M-speedup.



(a) Cumulative distribution of G2 max-cut problem, obtained executing (NumIter=200) the software implementations of the algorithm

(b) Energy evolution of G2 max-cut problem, obtained executing (NumIter=200) the software implementations of the algorithm

(c) Position variables evolution of G2 max-cut problem, obtained executing (NumIter=200) the software implementation of the proposed approximated version of the algorithm

(d) Cumulative distribution of G55 max-cut problem (5000 nodes), obtained executing (NumIter=500) the software implementations of the algorithm

(e) Energy evolution of G55 max-cut problem (5000 nodes), obtained executing (NumIter=500) the software implementations of the algorithm

(f) Position variables evolution of G55 max-cut problem (5000 nodes), obtained executing (NumIter=500) the software implementation of the proposed approximated version of the algorithm

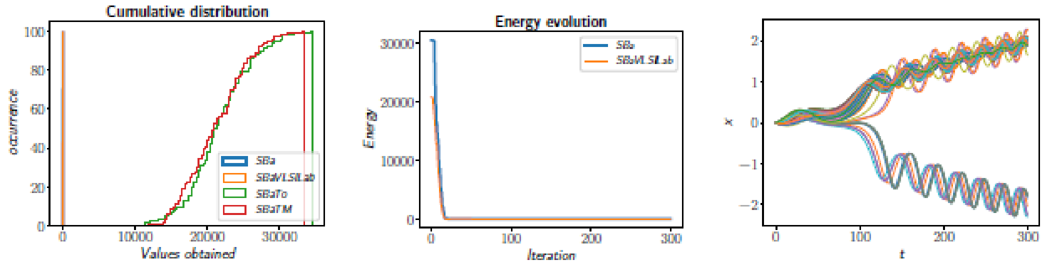
(g) Cumulative distribution of G81 max-cut problem (20000 nodes), obtained executing (NumIter=500) the software implementations of the algorithm

(h) Energy evolution of G81 max-cut problem (20000 nodes), obtained executing (NumIter=500) the software implementations of the algorithm

(i) Position variables evolution of G81 max-cut problem (20000 nodes), obtained executing (NumIter=500) the software implementation of the proposed approximated version of the algorithm

Just Accepted

Fig. 20. Cumulative distributions, average energy and position variables evolutions obtained by running one hundred times each software implementation on G2, G55, and G81 max-cut problem of the G-set. **SBa** is the floating-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation. **SBaTo** is the floating-point implementation of the algorithm by applying the M-speedup. **SBaTM** is the floating-point implementation of the algorithm by applying both the proposed approximation and the M-speedup.

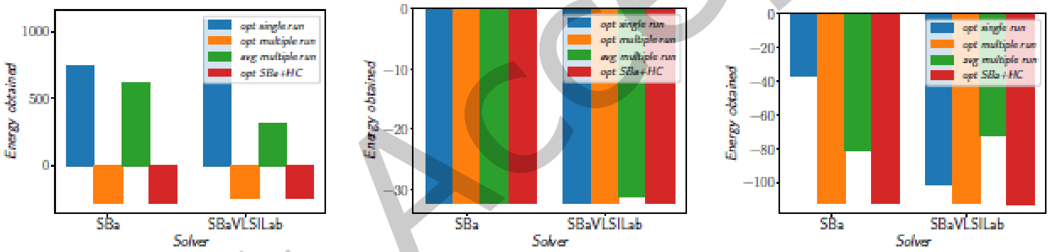


(a) Cumulative distribution of 103-variable knapsack problem, obtained executing (NumIter=300) the software implementations of the algorithm

(b) Energy evolution of 103-variable knapsack problem, obtained executing (NumIter=300) the software implementations of the algorithm

(c) Position variables evolution of 103-variable knapsack problem, obtained executing (NumIter=300) the software implementation of the proposed approximated version of the algorithm

Fig. 21. Cumulative distributions, average energy and position variables evolutions obtained by running one hundred times each software implementation on a 103-variable knapsack problem. **SBa** is the floating-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation. **SBaTo** is the floating-point implementation of the algorithm by applying the M-speedup. **SBaTM** is the floating-point implementation of the algorithm by applying both the proposed approximation and the M-speedup.



(a) Results obtained with the f1_l-d_kp_10_269 problem of the 0/1 Knapsack set

(b) Results obtained with the f4_l-d_kp_4_11 problem of the 0/1 Knapsack set

(c) Results obtained with the f7_l-d_kp_7_50 problem of the 0/1 Knapsack set

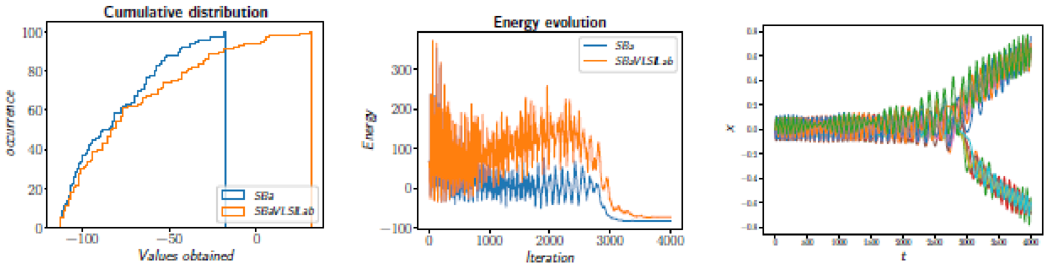
Fig. 22. Optimal value obtained with a single run (**opt single run**), one hundred runs ((**opt multiple run**), the average value (**avg multiple run**) obtained in the multiple execution case and the energy obtained after the application of some hill-climbing iterations on the multiple run optimal solution (**opt SBa + HC**) are reported for each software implementation considering f1_l-d_kp_10_269, f4_l-d_kp_4_11 and f7_l-d_kp_7_50 problems of the 0/1 Knapsack set. **SBa** is the floating-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation.

implementations involving the M-speedup (SBaTo and SBaTM).

Figure 2(b) presents cumulative distribution, energy and position variables evolutions of a 103-variable knapsack problem generated with qubovert. It proves that the performance of SBa and SBaVLSILab are comparable and better than that of SBaTo and SBaTM for this kind of problem.

Figure 22 reports the optimal values obtained with a single run (**opt single run**), one hundred runs (**opt multiple runs**), the average value (**avg multiple runs**) obtained in the multiple execution case and the energy obtained

Just Accepted

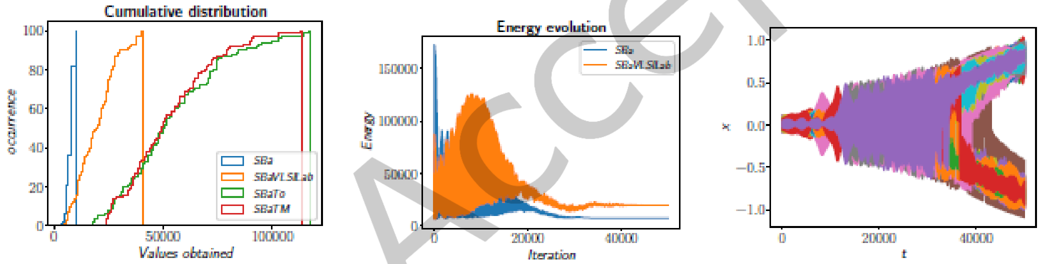


(a) Cumulative distribution of f7_l-d_kp_7_50 knapsack problem, obtained executing (NumIter=4000) the software implementations of the algorithm

(b) Energy evolution of f7_l-d_kp_7_50 knapsack problem, obtained executing (NumIter=4000) the software implementations of the algorithm

(c) Position variables evolution of f7_l-d_kp_7_50 knapsack problem, obtained executing (NumIter=4000) the software implementation of the proposed approximated version of the algorithm

Fig. 23. Cumulative distributions, average energy and position variables evolutions obtained by running one hundred times each software implementation on the f7_l-d_kp_7_50 knapsack problem of the 0/1 Knapsack set. **SBa** is the floating-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation.



(a) Cumulative distribution of a 25-variable traveling salesman problem, obtained executing (NumIter=50000) the software implementations of the algorithm

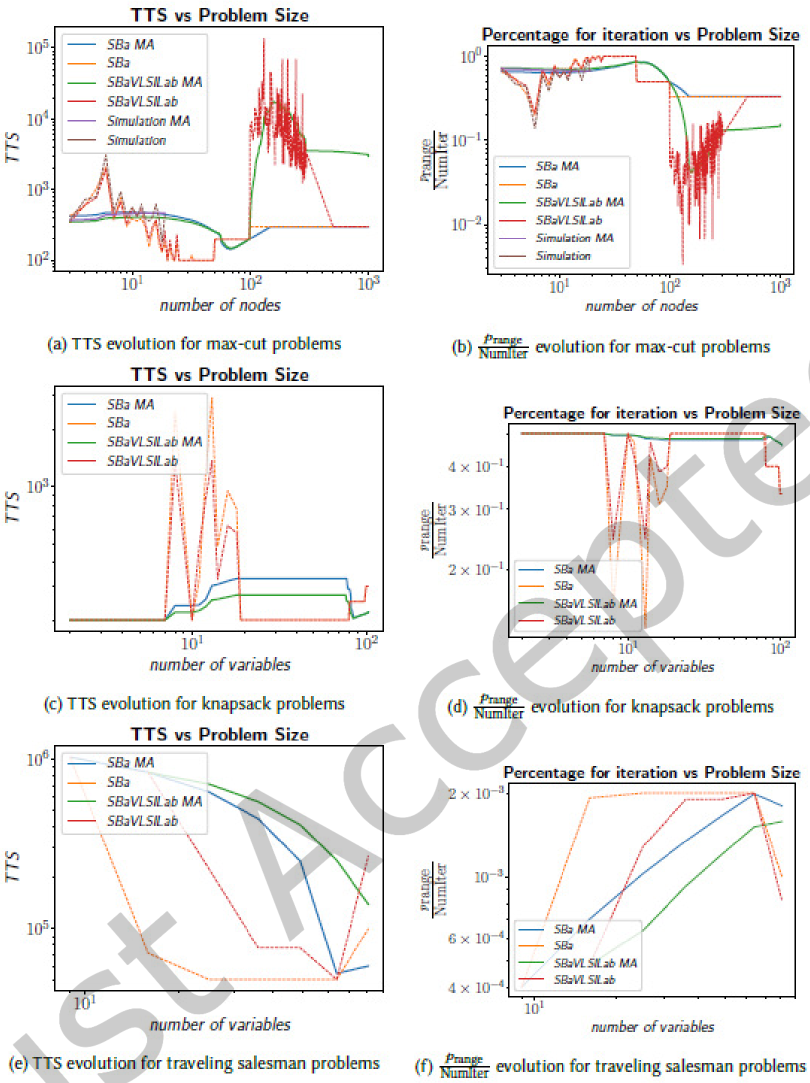
(b) Energy evolution of a 25-variable traveling salesman problem, obtained executing (NumIter=50000) the software implementations of the algorithm

(c) Position variables evolution of a 25-variable traveling salesman problem, obtained executing (NumIter=50000) the software implementation of the proposed approximated version of the algorithm

Fig. 24. Cumulative distributions, average energy and position variables evolutions obtained by running one hundred times each software implementation on a 25-variable traveling salesman problem. **SBa** is the floating-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation. **SBaTo** is the floating-point implementation of the algorithm by applying the M-speedup. **SBaTM** is the floating-point implementation of the algorithm by applying both the proposed approximation and the M-speedup.

after the application of some hill-climbing iterations on the multiple run optimal solution (**opt SBa + HC**) for SBa and SBaVLSILab, considering f1_l-d_kp_10_269, f4_l-d_kp_4_11, and f7_l-d_kp_7_50 of the 0/1 Knapsack set. The results of SBa and SBaVLSILab are consistent also with these benchmarks, showing that the suggested initialization in the case of a single run is effective and the application of hill-climbing on the optimal solution can help improve it. Figure 23 shows, as an example, the cumulative distribution, energy and position variables

Just Accepted



(a) TTS evolution for max-cut problems

(b) $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution for max-cut problems

(c) TTS evolution for knapsack problems

(d) $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution for knapsack problems

(e) TTS evolution for traveling salesman problems

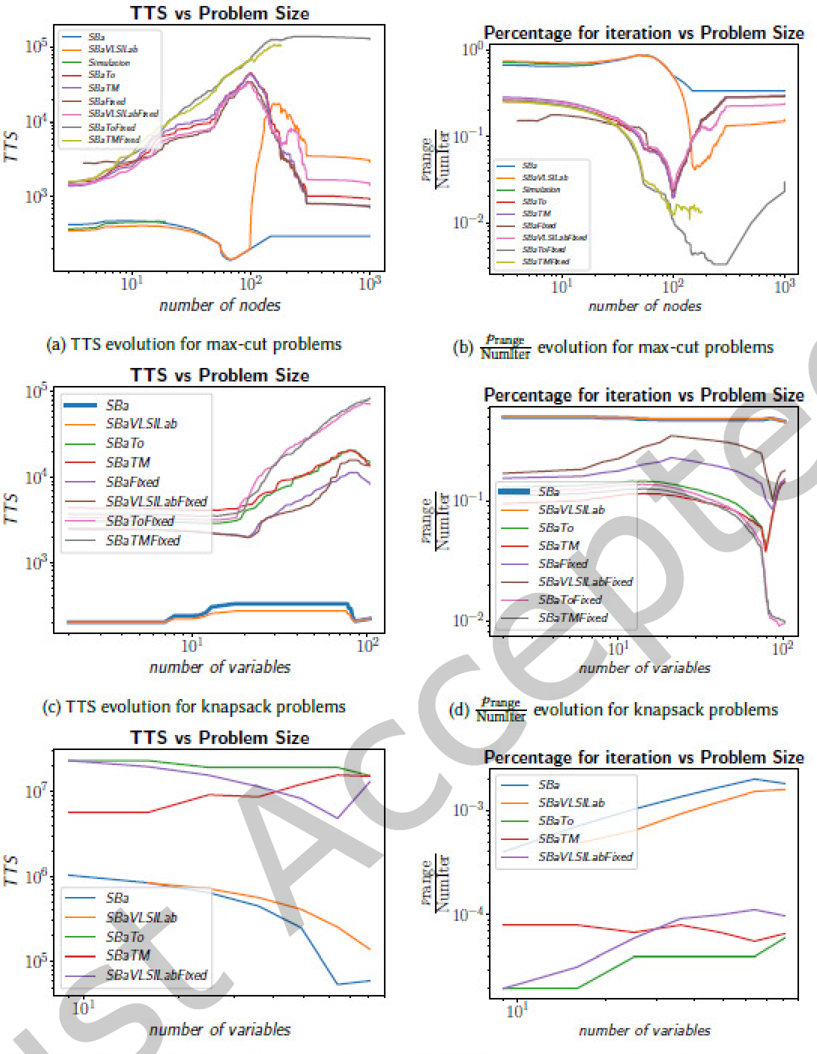
(f) $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution for traveling salesman problems

Fig. 25. Evolution of $\frac{p_{\text{range}}}{\text{NumIter}}$ and Time-To-solution (TTS), both not averaged and averaged (MA), varying the number of involved binary variables in the optimization problems. **SBa** is the floating-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation. **Simulation** corresponds to the hardware simulation.

Just Accepted

evolutions of the f7_l-d_kp_7_50 knapsack problem.

Moreover, the cumulative distribution, energy and position variables evolutions of a 25-variable traveling salesman problem generated with qubovert are reported in Figure 24, indicating that SBa and SBaVLSILab final results are better than those of SBaTO and SBaTM.



(a) TTS evolution for max-cut problems

(b) $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution for max-cut problems

(c) TTS evolution for knapsack problems

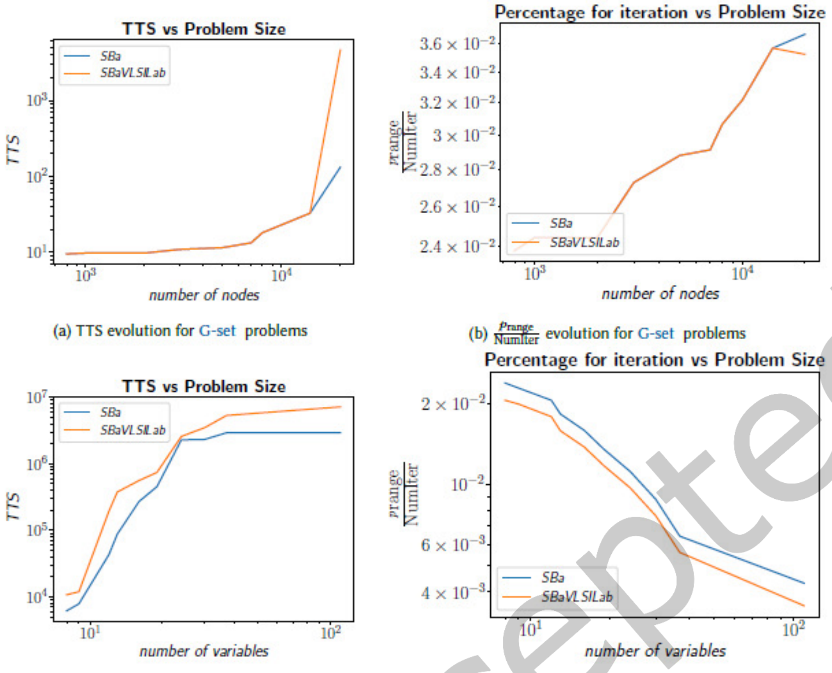
(d) $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution for knapsack problems

(e) TTS evolution for traveling salesman problems

(f) $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution for traveling salesman problems

Fig. 26. Evolution of $\frac{p_{\text{range}}}{\text{NumIter}}$ and Time-To-solution (TTS) varying the number of involved binary variables in the optimization problems. In these plots, both metrics are averaged. **SBa** is the floating-point implementation of the original algorithm. **SBaFixed** is the fixed-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation. **SBaVLSILabFixed** is the fixed-point implementation of the algorithm by applying the proposed approximation. **SBaTo** is the floating-point implementation of the algorithm by applying the M-speedup. **SBaToFixed** is the fixed-point implementation of the algorithm by applying the M-speedup. **SBaTM** is the floating-point implementation of the algorithm by applying both the proposed approximation and the M-speedup. **SBaTMFixed** is the fixed-point implementation of the algorithm by applying both the proposed approximation and the M-speedup. **Simulation** is the Modelsim simulation.

Just Accepted



(a) TTS evolution for G-set problems

(b) $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution for G-set problems

(c) TTS evolution for 0/1 Knapsack set problems

(d) $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution for 0/1 Knapsack set problems

Fig. 27. Evolution of $\frac{p_{\text{range}}}{\text{NumIter}}$ and Time-To-solution (TTS) varying the number of involved binary variables in the optimization problems. In these plots, both metrics are averaged. **SBa** is the floating-point implementation of the original algorithm. **SBaVLSILab** is the floating-point implementation of the algorithm by applying the proposed approximation.

Just Accepted

Finally, Time-To-Solution (TTS) and $\frac{p_{\text{range}}}{\text{NumIter}}$ evolution varying the number of involved binary variables are reported. Before commenting on any result, it is important to justify the preference of averaged results for TTS and $\frac{p_{\text{range}}}{\text{NumIter}}$. Figure 25(d) reports both the non-averaged and averaged results for max-cut, knapsack and traveling salesman problems, depending on the number of nodes. It is possible to observe on one hand many fluctuations in the dotted plots, which are associated with results not averaged, on the other the chosen moving average length smooths the trend, not significantly altering the overall trend of the metric, thus allowing reasonable performance comparisons between the different solvers. Plots associated with knapsack (Figures 25(c) and 25(d)) and traveling salesman problems (Figures 25(e) and 25(f)) present fewer fluctuations than the counterpart for max-cut problems (Figures 25(a) and 25(b)). The reason can be simply traced back to the fact that fewer nodes were involved.

Figures 26 and 27 show the averaged TTS and $\frac{p_{\text{range}}}{\text{NumIter}}$ results, for each class of problem considered (max-cut, knapsack and traveling salesman) for each solver. Note that evolutions obtained with SBa and SBaVLSILab are enough similar, proving that the proposed approximation does not affect significantly the performance. TTS parameter is expected to grow exponentially with the problem dimension [37] for quantum annealers:

$$\text{TTS} \approx 10^{a+b\sqrt{n}+c\log(\sqrt{n})}, \tag{56}$$

where n is the number of involved binary variables, a, b and c are interpolation coefficients. Therefore, to compare the evolution trend of SBa with that of QA, the evolution of TTS is reported in a logarithmic scale for

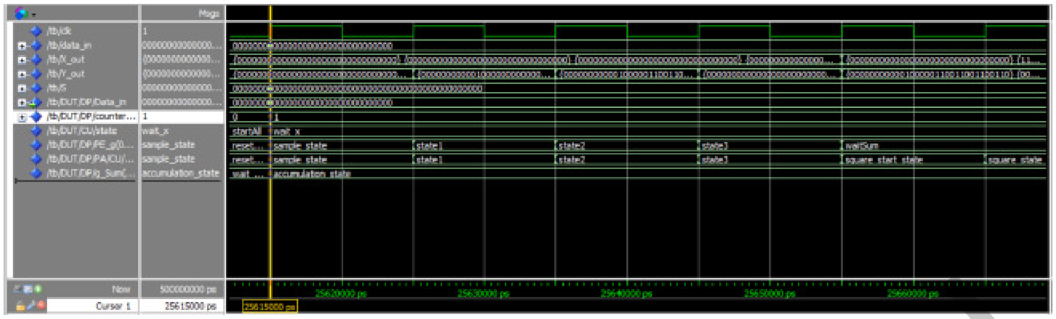


Fig. 28. Waves of the simulation of the first iteration of a 50-node max-cut.

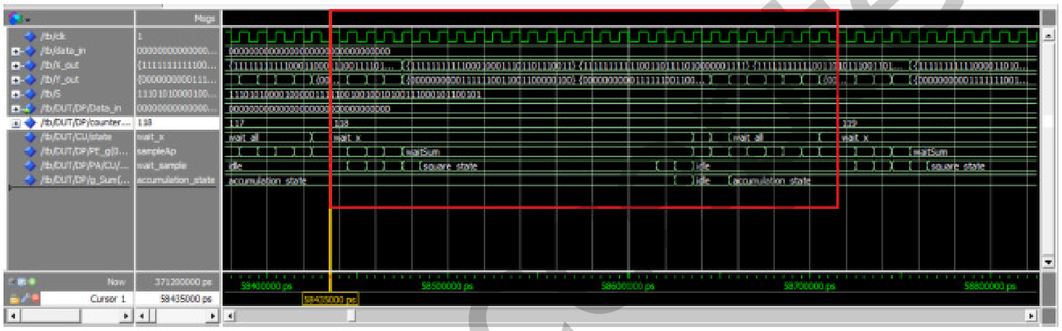


Fig. 29. Waves of the simulation of an iteration of a 50-node max-cut. In particular, it shows the state of the top-level control unit as well as those of the PE0, Sum0, and the P A0 updater, for allowing the observation of the high level of parallelization in the architecture of Figure 13.

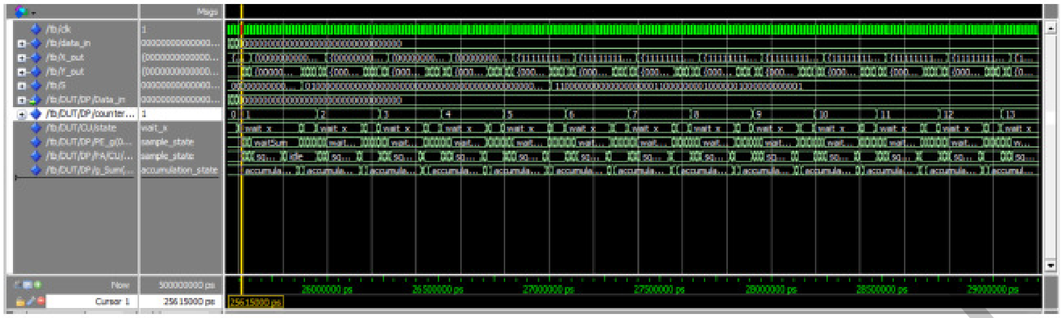
Just Accepted

each considered optimization and for each solver. In the max-cut case, where a higher number of problems are considered, and thus the trend is identifiable, it is possible to observe an evolution similar to that of a quantum annealer.

4.2 Functional verification

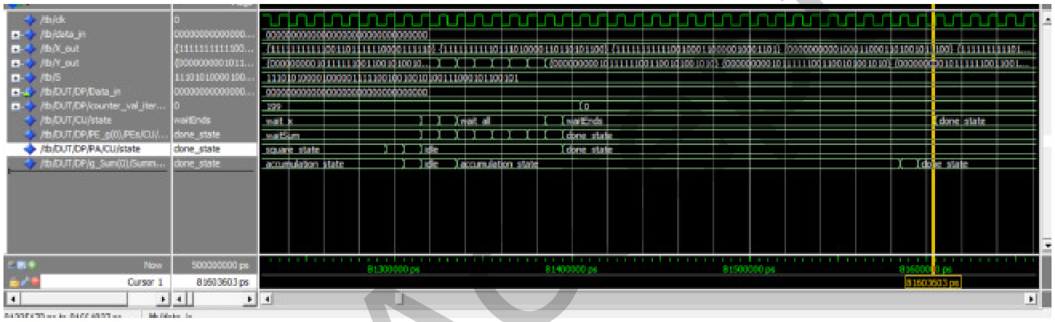
As mentioned, functional verification of the architecture was done in two ways.

Firstly, each block (`PE`, `Sums`, and `P A0 Update`) was tested individually. Subsequently, the overall architecture was simulated with Modelsim 17.1, employing the automation script represented in Figure 15, considering many problem sizes. The results obtained are compared with the outcomes of the software models to verify the correctness of the architecture description. Figure 29 shows waves of the simulation of a single iteration of a 50-node max-cut (red square). In particular, the state top-level control unit and those of the PE0, Sum0, and the P A0 updater are reported for emphasizing the advantage in terms of parallelization of the proposed approximation. Indeed, Note that the sum block remains active throughout (except for two clock cycles in which it waits for the correct sampling of its output and of the x variables) and that the number of clock cycles required for an iteration is, coherently with the theory reported in Section 3, equal to 28, i.e. $\left\lceil \frac{49}{2} \right\rceil + 3$. Additionally, Figures 28, 31



```
/tb/clk                     1
/tb/data_in                 000000000000000...
/tb/X_out                   {000000000000000...
/tb/Y_out                   {000000000000000...
/tb/S                       000000000000000...
/tb/DUT/DP/Data_in          000000000000000...
/tb/DUT/DP/counter...       1
/tb/DUT/CU/state            wait_x
/tb/DUT/DP/PE_g(0...        sample_state
/tb/DUT/DP/PA/CU/...        sample_state
/tb/DUT/DP/g_Sum(...        accumulation_state
Now                         500000000 ps
Cursor 1                    25615000 ps
```

Fig. 30. Waves of the simulation of some iterations of a 50-node max-cut. In particular, it shows the state of the top-level control unit as well as those of the PE0, Sum0, and the P A0 updater, for allowing the observation of the high level of parallelization in the architecture of Figure 13.



```
/tb/clk                          0
/tb/data_in                      000000000000000...
/tb/X_out                        {11111111111100...
/tb/Y_out                        {0000000001011...
/tb/S                            1110101000010 0...
/tb/DUT/DP/Data_in               000000000000000...
/tb/DUT/DP/counter_val_iter...   0
/tb/DUT/CU/state                 waitEnds
/tb/DUT/DP/PE_g(0)/PEs/CU/...    done_state
/tb/DUT/DP/PA/CU/state           done_state
/tb/DUT/DP/g_Sum(0)/Summ...      done_state
Now                              500000000 ps
Cursor 1                         81603603 ps
```

Fig. 31. Waves of the simulation at the end of an algorithm run.

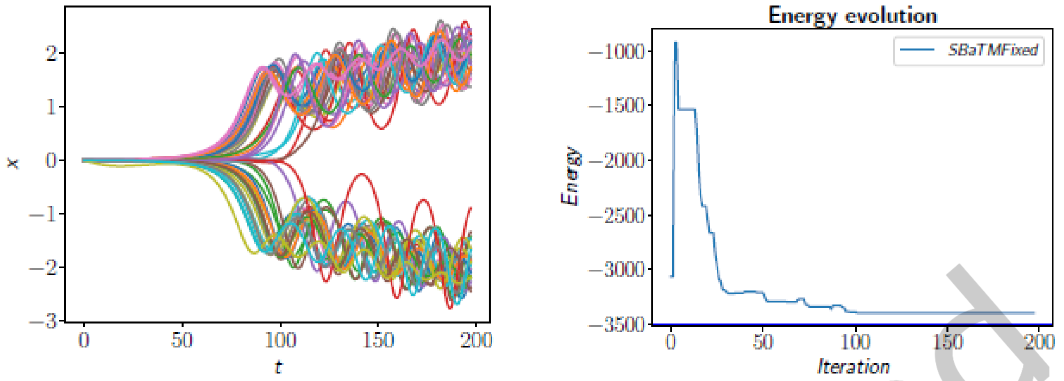
Just Accepted

and 30 show the first, last, and sequences of several iterations, respectively. The position variables x and energy evolution obtained by simulating a single run are reported in Figure 32.
Afterwards, the correctness of the post-synthesis description of the architecture was verified by testing the solver on-board connected to the Nios II processor, whose role is to send the algorithm parameters and to receive the final configuration. Also, in this case, the obtained results are compared with those of the software model, and the coherence among them assures us of the correctness of the synthesized architecture.

4.3 Synthesis results

This section reports the results obtained by synthesizing the architecture employing the automation script of Figure 17 while varying the bits for number representation and the size of the solvable problems.

4.3.1 Setup overview. The synthesis was performed by executing the tcl files generated by the automation script using Quartus Lite 17.1. In order to explore the tool potential, each combination, number of bits/number of spins was synthesized by applying speed optimization (`OPTIMIZATION_MODE "AGGRESSIVE PERFORMANCE"`), area optimization (`OPTIMIZATION_MODE "AGGRESSIVE AREA"`) and balanced options (`OPTIMIZATION_MODE BALANCED`).



(a) Position variables x evolution obtained by simulating the architecture for solving the 50-node max-cut problem

(b) Energy evolution obtained by simulating a single run of the architecture for solving a 50-node max-cut problem

Fig. 32. Simulation of a single run of the architecture for solving a 50-node max-cut problem.

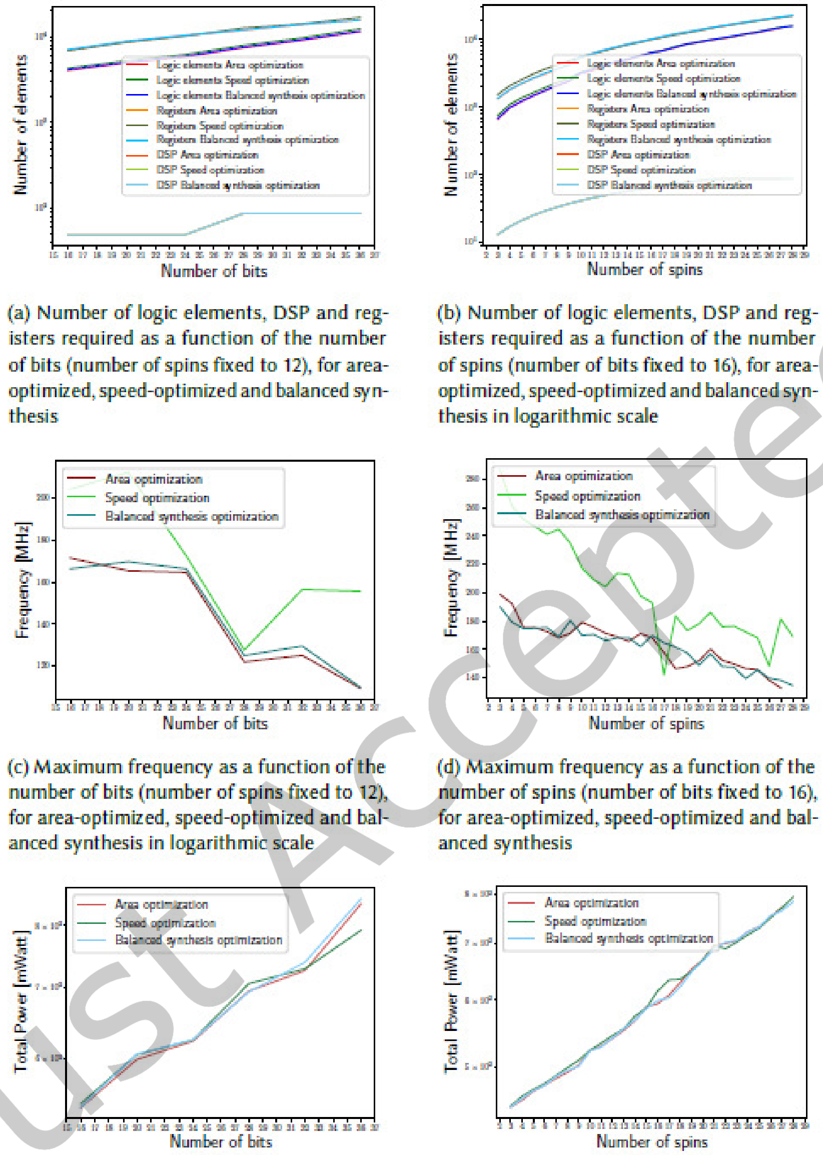
Just Accepted

Moreover, the power and the time multi-corner analyzer were activated.
Before reporting all the results, it is important to clarify that synthesis took a few minutes for all the performed tests and was only needed when changing the architecture in terms of oscillators and data width (e.g., number of bits for fixed-point representation). Once done, the hardware was stable, and further synthesis was unnecessary. Compiling the Nios II processor firmware was the only additional step, typically taking less than a minute.

4.3.2 Figures of merit. The synthesis results were evaluated in terms of occupied area, speed, and power consumption. In particular, the following figures of merit were considered:

- **Logic utilization**, measured in terms of **Adaptive Logic Modules** (**ALM**), which are the basic building block of the device family (**LE**);
- **Digital Signal Processors** (**DSPs**) employed;
- Required **registers** (**REG**);
- Maximum operating **frequency**, expressed in MHz;
- **Dynamic Power** consumption, expressed in mW (**DP**);
- **Static Power** consumption, expressed in mW (**SP**);
- **Total Power** consumption, expressed in mW (**Tot P**).

All these values can be read from the report files provided by the Quartus synthesis.
The most important figures of merit for evaluating the architecture are the frequency (associated with speed) and those related to area occupation, i.e., number of logic elements, DSP, and registers. Indeed, the occupied area allows the estimation of the FPGA size required for solving a problem of a certain dimension, while the maximum operating frequency provides an idea of the time required to solve a problem. The considered power consumption is obtained without applying back-annotation — i.e., without providing a transactions file coming from simulation — procedure but exploiting the Quartus power analyzer since the power consumption is not the most critical point in this design and n estimation is sufficient, even if not perfectly accurate.



(a) Number of logic elements, DSP and registers required as a function of the number of bits (number of spins fixed to 12), for area-optimized, speed-optimized and balanced synthesis

(b) Number of logic elements, DSP and registers required as a function of the number of spins (number of bits fixed to 16), for area-optimized, speed-optimized and balanced synthesis in logarithmic scale

(c) Maximum frequency as a function of the number of bits (number of spins fixed to 12), for area-optimized, speed-optimized and balanced synthesis in logarithmic scale

(d) Maximum frequency as a function of the number of spins (number of bits fixed to 16), for area-optimized, speed-optimized and balanced synthesis

(e) Total power consumed as a function of the number of bits (number of spins fixed to 12), for area-optimized, speed-optimized and balanced synthesis

(f) Total power consumed as a function of the number of spins (number of bits fixed to 16, for area-optimized, speed-optimized and balanced synthesis

Fig. 33. Synthesis results obtained by considering area-optimized, speed-optimized and balanced directives.

4.3.3 Performed tests. The architecture was synthesized by gradually increasing the number of spins (step 1) and bits for representing numbers (step 4) until the available hardware resources were sufficient. All the obtained

Just Accepted

results are reported in tables of the Supplementary Materials file.

The relation between figures of merit, number of spins, and bits employed for number representation is emphasized in Figure 33. Plots of figures merit as a function of the number of bits are obtained considering a 12-spin machine. In contrast, graphs showing the relation between synthesis results and the number of spins are acquired with a 16-bit for the number representation.

In particular, Figure 33(a) shows the linear relation between the number of registers and logic utilization and the bits employed for numerical representation, which aligns with expectations. DSPs are employed in the synthesis as arithmetic components (when they are described behaviourally), so linear growth alternating with constant stretches can be explained by the availability of these blocks with a non-fine granularity for what concerns the number of bits.

Figure 33(b) reports the relation of DSP, number of registers, and logic utilization with the number of considered spins. Note that, coherently with expectations, the number of registers and logic utilization grow little more than linearly with the number of considered oscillators (the contribution out of linearity comes from registers files for J matrix memorization, which have a quadratic growth). The DSP grows linearly.

It is possible to observe that, except for the number of DSPs required (which is constant), the optimization area directives are generally effective since the results obtained by exploiting these produce an area lower than those obtained with high-speed options and balanced synthesis.

Figure 33(c) reports the relation between the maximum operating frequency and the number of bits. As expected, a decreasing trend, even if not continuous, is probably related to the DSP allocation trend's discontinuity and the variable number of square root pipeline levels with the number of bits.

Figure 33(d) illustrates instead the relation between the maximum operating frequency and the problem size. Again, it is possible to notice a decreasing trend, even if not particularly smooth. In both figures, it is possible to notice how the speed optimization directives are very effective in that the maximum operating frequency is perceptibly higher than those obtained with a balanced synthesis or with area optimization options.

Figures 33(e) and 33(f) present the total power consumption as a function of the number of bits and the number of spins, respectively. It is possible to observe that in both cases, the power increases. This is reasonable because by increasing the number of bits or the number of spins the overall size of the Simulated Bifurcation Machine increases, implying a higher power consumption.

5 CONCLUSIONS

This work proposed a quantum-inspired digital solver for Ising models based on the Simulated Adiabatic Bifurcation (SBa) algorithm, emulating the evolution of a quantum system composed of interacting non-linear Kerr oscillators. The architecture — characterized by a flexible description in terms of bits employed for number representation and of the size of the problem to be solved — can manage the external field contribution of the Ising model and implements a proposed approximation of the original algorithm able to guarantee a speed-up thanks to a higher degree of parallelism in computation. Furthermore, the Ising machine has been integrated in a toolchain context, supporting future developers and users. First of all, software models of the algorithm in `Cython` language were provided, compliant with the qubovert library for QUBO problems description and formulation. Users can use these directly, and developers can exploit these as a reference for validating hardware. Then, `Python` classes for automatizing synthesis and architecture simulations were provided. Finally, the architecture was interfaced with the Nios II processor for an on-board test.

The results show that the proposed approximation does not significantly affect the quality of the solution but allows a higher degree of parallelization. Moreover, the implemented method for supporting the external field contribution of the Ising formulation was proven to be satisfactory, even if there is room for improvement. Finally, the proposal of exploiting local search to improve the final solution obtained gives encouraging results. Therefore,

Just Accepted

it could be interesting to investigate it more in-depth. As expected, the architecture scales linearly, increasing the bits for numerical representation and a little more than linearly, increasing the number of oscillators to mimic.
A Julia ecosystem for QUBO modelling, rich in variables encoding, constraints and solvers, has been recently proposed in [62]. The development of an interface or a wrapper between this and both the software model and the hardware presented in this article could undoubtedly be an interesting future work, not only for the developer in terms of accessibility to additional backends but also for the backend developers since more benchmark problems can be tested on their platforms. Moreover, the Window Sticker framework, aiming to support the study of the quantum-inspired optimization approach available in [39], could be exploited in the future for identifying further strategies for tuning the algorithm parameters, thus improving the effectiveness of the exploration approach.
The architecture can be further improved by working on its bottleneck: the sum block. A possibility for overcoming limitations of the current design could be investigating the most recent architectures for computing matrix-vector multiplication — i.e., observing together the sum blocks, Note that they compute the product between the J matrix and the x vector —, such as systolic arrays [61], or exploit emerging technology, such as logic-in-memory [34].
The work could also be expanded to support different evolutions of the oscillator network beyond adiabatic, such as the aforementioned ergodic, discrete and ballistic, and the higher-order cost functions version of the algorithm [29].
Even though the proposed work is a preliminary prototype of the Ising machine implementing SBa, it could represent, also for the accessibility of codes, a starting point for spreading the knowledge and the use of SBa, improving its exploitability.
We aspire that the creation of this open-access toolchain will empower both researchers and industries to explore this captivating optimization method, thus enabling them to tackle challenges that have the potential to enhance the quality of people's lives.

REFERENCES

[1] [n.d.]. Intel Xeon Gold 6134 Processor - Product Specification. [Online] https://ark.intel.com/content/www/us/en/ark/products/120493/intel-xeon-gold-6134-processor-24-75m-cache-3-20-ghz.html, accessed 25-October-2021.

[2] [n.d.]. *Python package Dimod.* https://github.com/dwavesystems/dimod.

[3] Amira Abbas, Andris Ambainis, Brandon Augustino, Andreas Bärtschi, Harry Buhrman, Carleton Coffrin, Giorgio Cortiana, Vedran Dunjko, Daniel J. Egger, Bruce G. Elmegreen, Nicola Franco, Filippo Fratini, Bryce Fuller, Julien Gacon, Constantin Gonciulea, Sander Gribling, Swati Gupta, Stuart Hadfield, Raoul Heese, Gerhard Kircher, Thomas Kleinert, Thorsten Koch, Georgios Korpas, Steve Lenk, Jakub Marecek, Vanio Markov, Guglielmo Mazzola, Stefano Mensa, Naeimeh Mohseni, Giacomo Nannicini, Corey O'Meara, Elena Peña Tapia, Sebastian Pokutta, Manuel Proissl, Patrick Rebentrost, Emre Sahin, Benjamin C. B. Symons, Sabine Tornow, Victor Valls, Stefan Woerner, Mira L. Wolf-Bauwens, Jon Yard, Sheir Yarkoni, Dirk Zechiel, Sergiy Zhuk, and Christa Zoufal. 2023. Quantum Optimization: Potential, Challenges, and the Path Forward. arXiv:2312.02279 [quant-ph]

[4] Tameem Albash and Daniel A Lidar. 2018. Demonstration of a scaling advantage for a quantum annealer over simulated annealing. *Physical Review X* 8, 3 (2018), 031016. https://doi.org/10.1103/PhysRevX.8.031016.

[5] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G. Katzgraber. 2019. Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer. *Frontiers in Physics* 7 (2019), 48. https://doi.org/10.3389/fphy.2019.00048.

[6] Mandell Bellmore and George L Nemhauser. 1968. The traveling salesman problem: a survey. *Operations Research* 16, 3 (1968), 538–558. https://doi.org/10.1287/opre.16.3.538.

[7] Thomas Bouquet, Mehdi Hmyene, François Porcher, Lorenzo Pugliese, and Jad Zeroual. 2021. Approximating Optimal Asset Allocations using Simulated Bifurcation. arXiv:2108.03092 [q-fin.PM] https://doi.org/10.48550/arXiv.2108.03092.

[8] Howard E. Brandt. 1999. Qubit devices and the issue of quantum decoherence. , 257–370 pages. http://dx.doi.org/10.1016/S0079-6727(99)00003-8.

[9] Giulio Casati and Vyacheslav Girko. 1993. Wigner's semicircle law for band random matrices. (1993). https://doi.org/10.1515/rose.1993.1.1.15.

[10] Bikas K Chakrabarti, Hajo Leschke, Purusattam Ray, Tatsuhiko Shirai, and Shu Tanaka. 2023. Quantum annealing and computation: challenges and perspectives. *Philosophical Transactions of the Royal Society A* 381, 2241 (2023), 20210419. https://doi.org/10.1098/rsta.

Just Accepted

2021.0419.

[11] Mark W. Coffey. 2017. Adiabatic quantum computing solution of the knapsack problem. *arXiv* (2017). https://doi.org/10.48550/arxiv.1701.05584.

[12] Chase Cook, Hengyang Zhao, Takashi Sato, Masayuki Hiromoto, and Sheldon X-D Tan. 2019. GPU-based Ising computing for solving max-cut combinatorial optimization problems. *Integration* 69 (2019), 335–344. https://doi.org/10.1016/j.vlsi.2019.07.003.

[13] Davide Costa, Mario Simoni, Gianluca Piccinini, and Mariagrazia Graziano. 2023. Advances in Modeling of Noisy Quantum Computers: Spin Qubits in Semiconductor Quantum Dots. *IEEE Access* 11 (2023), 98875–98913. https://doi.org/10.1109/ACCESS.2023.3312559

[14] Alexander M. Dalzell, Sam McArdle, Mario Berta, Przemyslaw Bienias, Chi-Fang Chen, András Gilyén, Connor T. Hann, Michael J. Kastoryano, Emil T. Khabiboulline, Aleksander Kubica, Grant Salton, Samson Wang, and Fernando G. S. L. Brandão. 2023. Quantum algorithms: A survey of applications and end-to-end complexities. arXiv:2310.03011 [quant-ph]

[15] Bryan Dury and Olivia Di Matteo. 2020. A QUBO Formulation for Qubit Allocation. https://doi.org/10.48550/ARXIV.2009.00140.

[16] Elias Fernandez-Combarro, CASTILLO ALVAREZ, and SAMUEL GONZALEZ. 2023. *PRACTICAL GUIDE TO QUANTUM MACHINE LEARNING AND QUANTUM OPTIMISATION: Hands-on Primer To... Quantum Computing*. PACKT PUBLISHING LIMITED.

[17] Fred Glover, Gary Kochenberger, and Yu Du. 2018. A tutorial on formulating and using QUBO models. (2018). https://doi.org/10.48550/arXiv.1811.11538.

[18] Hayato Goto. 2016. Bifurcation-based adiabatic quantum computation with a nonlinear oscillator network. *Scientific reports* 6, 1 (2016), 21686. https://doi.org/10.1038/srep21686.

[19] Hayato Goto. 2019. Quantum computation based on quantum adiabatic bifurcations of Kerr-nonlinear parametric oscillators. *Journal of the Physical Society of Japan* 88, 6 (2019), 061015. https://doi.org/10.7566/JPSJ.88.061015.

[20] Hayato Goto, Kotaro Endo, Masaru Suzuki, Yoshisato Sakai, Taro Kanao, Yohei Hamakawa, Ryo Hidaka, Masaya Yamasaki, and Kosuke Tatsumura. 2021. High-performance combinatorial optimization based on classical mechanics. *Science Advances* 7, 6 (2021), eabe7953. https://doi.org/10.1126/sciadv.abe7953 arXiv:https://www.science.org/doi/pdf/10.1126/sciadv.abe7953

[21] Hayato Goto, Kosuke Tatsumura, and Alexander R Dixon. 2019. Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems. *Science advances* 5, 4 (2019), eaav2372. https://doi.org/10.1126/sciadv.aav2372.

[22] Yoshitaka Haribara, Shoko Utsunomiya, Ken-ichi Kawarabayashi, and Yoshihisa Yamamoto. 2015. A coherent Ising machine for MAX-CUT problems: Performance evaluation against semidefinite programming relaxation and simulated annealing. *arXiv preprint arXiv:1501.07030* (2015). https://doi.org/10.1007/978-4-431-55756-2_12.

[23] Itay Hen, Joshua Job, Tameem Albash, Troels F Rønnow, Matthias Troyer, and Daniel A Lidar. 2015. Probing for quantum speedup in spin-glass problems with planted solutions. *Physical Review A* 92, 4 (2015), 042325. https://doi.org/10.1103/PhysRevA.92.042325.

[24] Kazuki Ikeda, Yuma Nakamura, and Travis S Humble. 2019. Application of quantum annealing to nurse scheduling problem. *Scientific reports* 9, 1 (2019), 1–10. https://doi.org/10.48550/arXiv.1904.12139.

[25] Joseph T. Iosue. [n.d.]. *pyqubo*. https://github.com/recruit-communication https://github.com/recruit-communication.

[26] Joseph T. Iosue. 2019. qubovert Documentation. [Online at https://qubovert.readthedocs.io/en/stable/,; accessed 10-May-2022].

[27] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. 2011. Quantum annealing with manufactured spins. *Nature* 473, 7346 (2011), 194–198. https://doi.org/10.1038/nature10012.

[28] Tadashi Kadowaki and Hidetoshi Nishimori. 1998. Quantum annealing in the transverse Ising model. *Physical Review E* 58, 5 (1998), 5355. https://doi.org/10.1103/PhysRevE.58.5355.

[29] Taro Kanao and Hayato Goto. 2022. Simulated bifurcation for higher-order cost functions. *Applied Physics Express* 16, 1 (2022), 014501. https://doi.org/10.35848/1882-0786/acaba9

[30] A.U. Khalid, Z. Zilic, and K. Radecka. 2004. FPGA emulation of quantum circuits. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.* 310–315. https://doi.org/10.1109/ICCD.2004.1347938.

[31] Ines Khoufi, Anis Laouiti, and Cedric Adjih. 2019. A survey of recent extended variants of the traveling salesman and vehicle routing problems for unmanned aerial vehicles. *Drones* 3, 3 (2019), 66. https://doi.org/10.3390/drones3030066.

[32] Matthew Kowalsky, Tameem Albash, Itay Hen, and Daniel A Lidar. 2022. 3-regular three-XORSAT planted solutions benchmark of classical and quantum heuristic optimizers. *Quantum Science and Technology* 7, 2 (2022), 025008. https://doi.org/10.1088/2058-9565/ac4d1b.

[33] Janusz Kusyk, Samah M. Saeed, and Muharrem Umit Uyar. 2021. Survey on Quantum Circuit Compilation for Noisy Intermediate-Scale Quantum Computers: Artificial Intelligence to Heuristics. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–16. https://doi.org/10.1109/TQE.2021.3068355.

[34] Orian Leitersdorf, Ronny Ronen, and Shahar Kvatinsky. 2022. MatPIM: Accelerating Matrix Operations with Memristive Stateful Logic. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. 215–219. https://doi.org/10.1109/ISCAS48785.2022.9937557

[35] Yangyang Li, Mengzhuo Tian, Guangyuan Liu, Cheng Peng, and Licheng Jiao. 2020. Quantum Optimization and Quantum Learning: A Survey. *IEEE Access* 8 (2020), 23568–23593. https://doi.org/10.1109/ACCESS.2020.2970105.

[36] Frauke Liers, Tim Nieberg, and Gregor Pardella. 2011. Via Minimization in VLSI Chip Design Application of a Planar Max-Cut Algorithm. http://e-archive.informatik.uni-koeln.de/630/.

Just Accepted

[37] Salvatore Mandra, Zheng Zhu, Wenlong Wang, Alejandro Perdomo-Ortiz, and Helmut G Katzgraber. 2016. Strengths and weaknesses of weak-strong cluster problems: A detailed overview of state-of-the-art classical heuristics versus quantum approaches. *Physical Review A* 94, 2 (2016), 022337. https://doi.org/10.1103/PhysRevA.94.022337.

[38] Yuki NAITO and Kunihiro FUJIYOSHI. [n.d.]. A Study on Updating Spins in Ising Model to Solve Combinatorial Optimization Problems. *city* 4 ([n.d.]), 3. https://sasimi.jp/new/sasimi2019/files/archive/pdf/p310_R4-13.pdf.

[39] Author Name. Year. *Stochastic Benchmark Repository.* https://github.com/usra-riacs/stochastic-benchmark Accessed: May 29, 2024.

[40] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press. https://doi.org/10.1017/CBO9780511976667.

[41] Keshab K Parhi. 1999. *VLSI digital signal processing systems: design and implementation.* John Wiley & Sons.

[42] Rodolfo A Quintero and Luis F Zuluaga. 2021. *Characterizing and benchmarking QUBO reformulations of the knapsack problem.* Technical Report. Technical Report. Department of Industrial and Systems Engineering, Lehigh University.

[43] Atanu Rajak, Sei Suzuki, Amit Dutta, and Bikas K Chakrabarti. 2023. Quantum annealing: an overview. *Philosophical Transactions of the Royal Society A* 381, 2241 (2023), 20210417. https://doi.org/10.1098/rsta.2021.0417.

[44] Mario Simoni, Giovanni Amedeo Cirillo, Giovanna Turvani, Mariagrazia Graziano, and Maurizio Zamboni. 2021. Towards compact modelling of noisy quantum computers: A molecular-spin-qubit case of study. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 18, 1 (2021), 1–26. https://doi.org/10.1145/3474223.

[45] Stefano Speziali, Federico Bianchi, Andrea Marini, Lorenzo Menculini, Massimiliano Proietti, Loris F. Termite, Alberto Garinei, Marcello Marconi, and Andrea Delogu. 2021. Solving Sensor Placement Problems In Real Water Distribution Networks Using Adiabatic Quantum Computation. https://doi.org/10.48550/arxiv.2108.04075.

[46] Kyle Steinhauer, Takahisa Fukadai, and Sho Yoshida. 2020. Solving the Optimal Trading Trajectory Problem Using Simulated Bifurcation. *arXiv preprint arXiv:2009.08412* (2020). https://doi.org/10.48550/arXiv.2009.08412.

[47] Junqing Sun, Gregory Peterson, and Olaf Storaasli. 2007. Sparse Matrix-Vector Multiplication Design on FPGAs. In *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2007).* 349–352. https://doi.org/10.1109/FCCM.2007.56

[48] Toufan D. Tambunan, Andriyan B. Suksmono, Ian J. M. Edward, and Rahmat Mulyawan. 2022. Quantum Annealing for Vehicle Routing Problem with weighted Segment. https://doi.org/10.48550/ARXIV.2203.13469.

[49] Kotaro Tanahashi, Shinichi Takayanagi, Tomomitsu Motohashi, and Shu Tanaka. 2019. Application of Ising machines and a software development for Ising machines. *Journal of the Physical Society of Japan* 88, 6 (2019), 061010. https://doi.org/10.7566/JPSJ.88.061010.

[50] Kosuke Tatsumura. 2021. Large-scale combinatorial optimization in real-time systems by FPGA-based accelerators for simulated bifurcation. In *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies.* 1–6. https://doi.org/10.1145/3468044.3468045.

[51] Kosuke Tatsumura, Alexander R. Dixon, and Hayato Goto. 2019. FPGA-Based Simulated Bifurcation Machine. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL).* 59–66. https://doi.org/10.1109/FPL.2019.00019.

[52] Kosuke Tatsumura, Alexander R. Dixon, and Hayato Goto. 2019. FPGA-Based Simulated Bifurcation Machine. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL).* 59–66. https://doi.org/10.1109/FPL.2019.00019.

[53] Kosuke Tatsumura, Ryo Hidaka, Jun Nakayama, Tomoya Kashimata, and Masaya Yamasaki. 2023. Pairs-Trading System Using Quantum-Inspired Combinatorial Optimization Accelerator for Optimal Path Search in Market Graphs. *IEEE Access* 11 (2023), 104406–104416. https://doi.org/10.1109/ACCESS.2023.3316727

[54] Kosuke Tatsumura, Ryo Hidaka, Jun Nakayama, Tomoya Kashimata, and Masaya Yamasaki. 2023. Real-Time Trading System Based on Selections of Potentially Profitable, Uncorrelated, and Balanced Stocks by NP-Hard Combinatorial Optimization. *IEEE Access* 11 (2023), 120023–120033. https://doi.org/10.1109/ACCESS.2023.3326816

[55] Kosuke Tatsumura, Masaya Yamasaki, and Hayato Goto. 2021. Scaling out Ising machines using a multi-chip architecture for simulated bifurcation. *Nature Electronics* 4, 3 (2021), 208–217. https://doi.org/10.1038/s41928-021-00546-4.

[56] John Robert Taylor and John R Taylor. 2005. *Classical mechanics.* Vol. 1. Springer.

[57] Lieven MK Vandersypen and Isaac L Chuang. 2005. NMR techniques for quantum control and computation. *Reviews of modern physics* 76, 4 (2005), 1037. https://doi.org/10.1103/RevModPhys.76.1037

[58] Davide Venturelli, Salvatore Mandrà, Sergey Knysh, Bryan O'Gorman, Rupak Biswas, and Vadim Smelyanskiy. 2015. Quantum optimization of fully connected spin glasses. *Physical Review X* 5, 3 (2015), 031040. https://doi.org/10.1103/PhysRevX.5.031040.

[59] Amit Verma and Mark Lewis. 2021. Variable Reduction For Quadratic Unconstrained Binary Optimization. https://doi.org/10.48550/arxiv.2105.07032.

[60] Deborah Volpe, Giovanni Amedeo Cirillo, Maurizio Zamboni, and Giovanna Turvani. 2023. Integration of Simulated Quantum Annealing in Parallel Tempering and Population Annealing for Heterogeneous-Profile QUBO Exploration. *IEEE Access* 11 (2023), 30390–30441. https://doi.org/10.1109/ACCESS.2023.3260765.

[61] Mahendra Vucha and Arvind Rajawat. 2011. Design and FPGA implementation of systolic array architecture for matrix multiplication. *International Journal of Computer Applications* 26, 3 (2011), 18–22. https://tinyurl.com/ymmmapf9

Just Accepted

[62] Pedro Maciel Xavier, Pedro Ripper, Tiago Andrade, Joaquim Dias Garcia, Nelson Maculan, and David E. Bernal Neira. 2023. QUBO.jl: A Julia Ecosystem for Quadratic Unconstrained Binary Optimization. arXiv:2307.02577 [math.OC]

[63] Mashiyat Zaman, Kotaro Tanahashi, and Shu Tanaka. 2022. PyQUBO: Python Library for Mapping Combinatorial Optimization Problems to QUBO Form. *IEEE Trans. Comput.* 71, 4 (2022), 838–850. https://doi.org/10.1109/TC.2021.3063618.

[64] Stefanie Zbinden, Andreas Bärtschi, Hristo Djidjev, and Stephan Eidenbenz. 2020. Embedding algorithms for quantum annealers with chimera and pegasus connection topologies. In *International Conference on High Performance Computing*. Springer, 187–206. https://doi.org/10.1007/978-3-030-50743-5_10.

[65] Tingting Zhang and Jie Han. 2022. Efficient Traveling Salesman Problem Solvers using the Ising Model with Simulated Bifurcation. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 548–551. https://doi.org/10.23919/DATE54114.2022.9774576.

[66] Tingting Zhang, Qichao Tao, and Jie Han. 2021. Solving Traveling Salesman Problems Using Ising Models with Simulated Bifurcation. In *2021 18th International SoC Design Conference (ISOCC)*. 288–289. https://doi.org/10.1109/ISOCC53507.2021.9613918.

[67] Yan Zhang, Yasser H. Shalabi, Rishabh Jain, Krishna K. Nagar, and Jason D. Bakos. 2009. FPGA vs. GPU for sparse matrix vector multiply. In *2009 International Conference on Field-Programmable Technology*. 255–262. https://doi.org/10.1109/FPT.2009.5377620

[68] Ling Zhuo and Viktor K Prasanna. 2005. Sparse matrix-vector multiplication on FPGAs. In *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. 63–74. https://doi.org/10.1145/1046192.1046202

[69] Michael Ryan Zielewski and Hiroyuki Takizawa. 2022. A method for reducing time-to-solution in quantum annealing through pausing. In *International Conference on High Performance Computing in Asia-Pacific Region*. 137–145. https://doi.org/10.1145/3492805.3492815.

[70] Yu Zou and Mingjie Lin. 2020. Massively simulating adiabatic bifurcations with FPGA to solve combinatorial optimization. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 65–75. https://doi.org/10.1145/3373087.3375298.

Received 27 September 2023; revised 23 February 2024; accepted 26 April 2024

Just Accepted