QxSQA: GPGPU-Accelerated Simulated Quantum Annealer within a Non-Linear Optimization and Boltzmann Sampling Framework

Dan Padilha, Serge Weinstock, and Mark Hodson
Rigetti Computing, Berkeley, CA 94710
Email: {dan.padilha, serge.weinstock, mark.hodson}@rigetti.com

***Abstract*—We introduce QxSQA, a GPGPU-Accelerated Simulated Quantum Annealer based on Path-Integral Monte Carlo (PIMC). QxSQA is tuned for finding low-energy solutions to integer, non-linear optimization problems of up to 2^{14} (16,384) binary variables with quadratic interactions on a single GPU instance. Experimental results demonstrate QxSQA can solve *Maximum Clique* test problems of $8{,}100$ binary variables with planted solutions in under one minute, with linear scaling against key optimization parameters on other large-scale problems. Through the PIMC formulation, QxSQA also functions as an accurate sampler of Boltzmann distributions for machine learning applications. Experimental characterization of Boltzmann sampling results for a reinforcement learning problem showed good convergence performance at useful scales. Our implementation integrates as a solver within our QxBranch developer platform, positioning developers to efficiently develop applications using QxSQA, and then test the same application code on a quantum annealer or universal quantum computer hardware platform such as those from D-Wave Systems, IBM, or Rigetti Computing.**

I. Introduction

In this paper we introduce QxSQA, a GPGPU-Accelerated Simulated Quantum Annealer based on the Path-Integral Monte Carlo (PIMC) method. Our motivations for this work are to improve the effectiveness of quantum application developers, and to provide enterprises with new candidate techniques for solving industry problems.

A. Simulated Quantum Annealing for Developers

Quantum annealing simulation enables developers to explore the applicability of a range of quantum computing techniques to problems of interest. Today's Noisy Intermediate-Scale Quantum (NISQ) computers are an expensive resource, limited in scale and by noise. Simulation provides a surrogate limited only by the capabilities of the simulation algorithm and the required classical compute resources.

The effectiveness of developers is impacted by their develop-test-debug cycle time. Simulation speed is a factor in this. In this paper we consider 1 minute to be a reasonable time for a developer to wait for simulation results. Motivated by improving developer effectiveness, we demonstrate in this paper that QxSQA can solve some classes of NP-hard problems of $8{,}100$ binary variables within this time constraint.

B. Simulated Quantum Annealing for Enterprise Problems

Quantum annealing simulation is also a candidate for "quantum-inspired" approaches to enterprise problems. This technique provides a new classical meta-heuristic for combinatorial optimization problems that may out-perform simulated annealing and genetic algorithms. Simulation of quantum dynamics using this technique supports an alternative approach to training deep learning networks for discrete variable problems that may lead to faster or more accurate training.

The effectiveness of a new technique for an enterprise is determined by its performance at industrial scale. In our experience, industrial problems involve many thousands or tens of thousands of binary variables. Quadratic optimization benchmark results such as QPLIB[1] demonstrate that optimization problems of these scales can be computationally intractable for many meta-heuristic solvers. Motivated by the potential for QxSQA to provide a new candidate technology for solving such problems, we demonstrate in this paper that QxSQA can scale to solve some classes of NP-hard problems of $16{,}002$ binary variables, finding an optimal solution (from known planted solutions) within 7 minutes.

C. Related Work

Our previous work in symbolic programming of optimization problems[2] minimized the time required for developers to build applications targeting quantum annealers and compatible simulators.

Other simulation tools have been used to conduct quantum annealing research. These include fast simulated annealing codes from [3], meta-heuristics exploiting limited-connectivity architectures from [4], and open source packages such as [5]. Most recently, [6] demonstrated a hardware-based $2{,}000$-spin Ising machine implemented on an FPGA, and a $100{,}000$-spin Ising model running on a GPU cluster, solved using a novel method known as *simulated bifurcation*.

II. Path-Integral Simulation of Quantum Annealing

We describe a simulation method based on the Path-Integral formulation of quantum mechanics, equivalent to Schrödinger's canonical formalism for the time-evolution of a quantum system[7]. This method is applicable to the simulation of *quantum annealing*, a quantum computation technique

based on a time evolution. Quantum annealing is of particular interest due to its applicability across a broad range of classical NP-Hard optimization problems, and is studied within the contexts of NISQ computing ([8], [9]), digital annealing hardware ([6], [10], [11]), and universal quantum algorithms[12].

A. Quantum Annealing and the Ising Model

As a method of computation, quantum annealing seeks the lowest-energy solution of a minimization problem, formulated via the computational *Ising model*:[13]

$$H^z(\boldsymbol{s}) = \boldsymbol{s}^T J \boldsymbol{s} + \boldsymbol{h}^T \boldsymbol{s} + c \tag{1}$$

where $H^z(\boldsymbol{s})$ is the Hamiltonian[1] defining the energy value of any given state $\boldsymbol{s}$, a vector of N two-state quantum-mechanical systems (quantum bits, or *qubits*) represented classically as binary variables $\{s_i = \pm 1 \mid i \in 1, \ldots, N\}$[2]. The *bias* vector $\boldsymbol{h}$, *coupling* matrix J, and constant c contain real-valued coefficients that represent a desired problem as interaction strengths on or between the problem variables of $\boldsymbol{s}$.

In an *adiabatic transformation*[7], it is possible to prepare a quantum-mechanical system H^x in a known eigenstate, and to slowly transform to the corresponding eigenstate of H^z:

$$H(t) = A(t)H^x + B(t)H^z \tag{2}$$

where $H^x = -\sum_i^N \sigma_i^x$ represents an interaction field directed transversely along the x-axis, and $A(t)$ and $B(t)$ control the *annealing schedule* at a time t. This adiabatic transformation results in quantum annealing, in which H^x is initially prepared in its known minimum-energy or *ground state*, and at the end of the annealing schedule, the process arrives at the corresponding ground state of H^z.

Mappings between Eq. 1 and all of Karp's 21 NP-Complete problems[14] imply that this formulation is general enough to represent any NP-Hard optimization problem. Quantum annealing can therefore be used to search for low-energy solutions to unconstrained binary optimization problems over their combinatorial space of 2^N possible solutions.

B. Path-Integral Monte Carlo (PIMC)

Path-Integral Monte Carlo (PIMC) is a numerical method that makes use of the Path-Integral formulation of quantum mechanics and the Monte Carlo method of statistical simulation. The Path-Integral formulation can be applied to simulate a quantum annealing process[15] by considering the transverse field H^x to be a non-commuting *kinetic term* of the Hamiltonian $H(t)$ of Eq. 2. This simulates the quantum dynamics induced by the transverse field, helping reduce the probability of becoming trapped in local minima. By introducing an imaginary-time dimension via the Trotter product formula [16], the kinetic term can be approximated via Monte Carlo, where P Trotter *slices* are simulated together with increasing accuracy as $P \to \infty$. For the annealing Hamiltonian $H(t)$, we introduce slices along the imaginary-time k-axis:

$$H = \sum_k^P \left(H^z(\boldsymbol{s}^k) + J_\perp (\boldsymbol{s}^k)^T \boldsymbol{s}^{k+1} \right) \tag{3}$$

where $H^z(\boldsymbol{s}^k)$ is the Ising energy (Eq. 1) of the kth Trotter slice. Here, $J_\perp$ is a coupling between each variable s_i and its counterpart in neighboring[3] slice $k+1$, defined as:

$$J_\perp = -\frac{PT}{2} \ln \tanh \frac{\Gamma}{PT} \tag{4}$$

where T represents a classical temperature and Γ represents a *transverse-field strength*.

PIMC can be implemented with the Metropolis algorithm[13] to stochastically explore a problem's solution space and search for the lowest energy states. Here, each of the P slices represents a proposal state, each of which affects the total energy of the system H (Eq. 3). The iterative Monte Carlo process performs S randomization *sweeps* across the P Trotter slices for each of τ time *steps* in a given annealing schedule. In each sweep, a random qubit s_i in each slice k is flipped with some probability α based on the Metropolis acceptance rate:

$$\alpha = \min \left(1, \exp \left(\frac{-\Delta_i^k H}{PT} \right) \right) \tag{5}$$

where we define $\Delta_i^k H$ as the difference in energy due to flipping the sign of qubit s_i on slice k:

$$\Delta_i^k H = -2 s_i^k \left(\sum_{j \neq i}^N (J_{ij} + J_{ji})\, s_j^k + h_i + J_\perp (s_i^{k-1} + s_i^{k+1}) \right) \tag{6}$$

Note that when $\Delta_i^k H < 0$, flipping qubit s_i^k results in a lower energy, thus the acceptance probability for the flip is $\alpha = 1$.

Given a constant temperature T and a linearly decaying transverse field Γ, this algorithm is known as *simulated quantum annealing* (SQA), and has been shown to perform well as an optimization technique in cases compared to classical simulated annealing[13], [17].

C. Techniques for GPGPU Acceleration

From Eq. 3, a PIMC parallelization opportunity arises by noting that the total energy of the system H is defined by a sum across Trotter slices. This allows each slice's contribution to the system energy to be computed in parallel[18], assuming some additional considerations for interactions between neighboring slices. Here, we describe techniques used in QxSQA to achieve accelerated parallelism on General Purpose Graphical Processing Units (GPGPU) using NVIDIA's `CUDA` framework.

Parallelization of the algorithm involves the allocation of the P Trotter slices onto GPU threads, such that, as the number of GPU threads is high, many Trotter slices are computed

[1] Canonically, the Ising Hamiltonian is defined over Pauli operators in the z-computational basis. For simplicity, we use classical binary variables.

[2] Alternatively, the equivalent Quadratic Unconstrained Binary Optimization (QUBO) model defines problem variables as $x_i = \frac{1}{2}(s_i + 1) \mid x_i \in \{0, 1\}$.

[3] Boundary conditions when $k+1 > P$ are computed as *open* (this coupling is ignored), or *periodic* (coupled to the first slice, $k + 1 \equiv 1$).

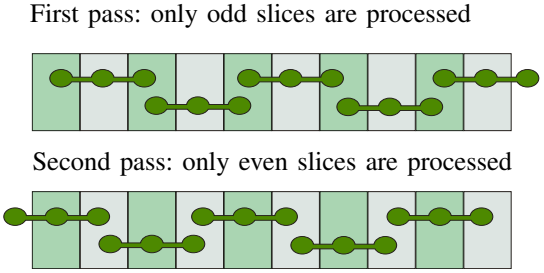

Fig. 1. A contiguous array of Trotter slices can be computed simultaneously in two passes of different "colors", where every odd slice is the same color, and every even slice is another.

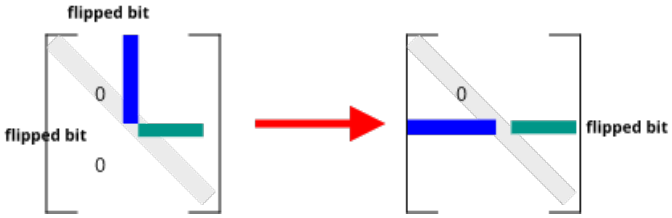

Fig. 2. A column (blue rectangle) of the upper-triangular J coupling matrix can be transposed along the lower-triangular part of the matrix, allowing for cache-efficient access when computing linear sums and scalar products.

in parallel. Avoiding synchronization between these threads is the first step towards realizing significant performance improvements[5]. As the energy contribution of each slice k (Eq. 6) depends on its neighboring slices $k \pm 1$, it is possible to compute independent simultaenous sweeps of all odd slices, then all even slices, as in Fig. 1.

Our definition of Eq. 6 additionally reduces the normally quadratic energy computation into a linear sum and multiplication of a single bit flip and constant factor. The linear sum is precomputed and stored in global memory before then running a separate compute kernel for computing the remaining terms.

Issues arising due to memory and communication overheads are mitigated through various data structure optimizations; for example, vectorized loads in `CUDA` allow loading up to four elements with only a single instruction[19]. To avoid expensive cache misses, which have significant impacts on performance, we note that *warps* of 32 threads share the same `L1`-cache, so the J coupling matrix can be structured in rows of multiples of 32, providing the threads with strided access to the same contiguous memory. The linear sum of Eq. 6 is made more cache-friendly by utilising the unused lower-triangular part of the J coupling matrix to store a transposed copy of the couplings, as in Fig. 2. In addition, *blocks* of warps process segments of this matrix row sequentially, allowing QxSQA to scale to problems with arbitrary numbers of binary variables, limited only by the total host and GPGPU memory available.

Other best-practice GPGPU optimizations also lead to significant performance improvements. Among these are parallel reductions[20], loop unrolling[21], pre-generating random numbers in global memory (using the built-in `curand` library for fast bulk random number generation), and avoiding the deallocation of GPGPU memory between similar runs.

III. Applications

Developing real-world applications for NISQ computers can be complex, inefficient, or otherwise unintuitive[22]. It becomes necessary for a prospective developer to have access to an environment in which potential solutions may be prototyped, iteratively improved, evaluated, and productionized. The utility of QxSQA for exploring quantum annealing applications or solving enterprise problems is delivered to data scientists, software engineers, and quantitative analysts through Integer Non-Linear Programming and Boltzmann sampling frameworks more familiar to these enterprise practitioners.

A. Integer Non-Linear Programming (INLP)

Integer Non-Linear Programming (INLP) is a technique for describing non-linear optimization problems on integer variables that may be subject to constraints, which may also be non-linear. The combinatorially large search space of solutions to such problems means that finding optimal solutions is exceedingly computationally demanding. Although relatively efficient heuristics sometimes exist for certain sub-classes of problems, solving the general case of an INLP problem requires the use of *meta-heuristic* or *approximation* methods[23], such as SQA.

A generic mapping exists from INLP to an Ising formulation, providing a means to easily reproduce constrained problems within the inherently unconstrained formulation of Eq. 1. Therefore, any Ising-based quantum annealer can be used to solve INLP problems[4], including QxSQA, digital annealers such as those from Hitachi[11] and Fujitsu[10], and adiabatic quantum optimizers such as those from D-Wave [28]. In addition, in conjunction with relevant gate-model quantum algorithms such as the Quantum Approximate Optimization Algorithm (QAOA)[12], this allows for the same problems to be run on current- and next-generation universal NISQ devices, such as those from IBM[29], Rigetti[30], and Google[31].

Expanding on our previous work in [2], we have developed the QxBranch developer platform, providing a suite of software development abstractions, including the INLP toolkit, for using quantum computers and simulators (as seen in Fig. 3). While this allows for a direct translation from INLP and Ising-formulated problems to some of the aforementioned devices, in practice there are often unique limitations inherent in each. For example: quantum optimizers are subject to the problems of minor-embedding[32] and additional penalty tuning for constraints[33]; gate-model devices require additional compilation to their supported gate sets and architectures, a trade-off of error arising due to circuit depth versus number of qubits[34], and may not support hard constraints[35]; cloud-based devices may require connectivity and queue management; and so on.

[4]A detailed description of INLP, its mapping to Ising, and translation to different devices or simulators is out of scope of the present paper. For an introduction, refer to our previous works of [2], [24], and related works of [25] in mapping of higher-order terms, [26] for boolean logic constructions, and most recently and comprehensively, [27] showing a constraints toolkit including a real-world application mapped on to quantum annealing hardware.

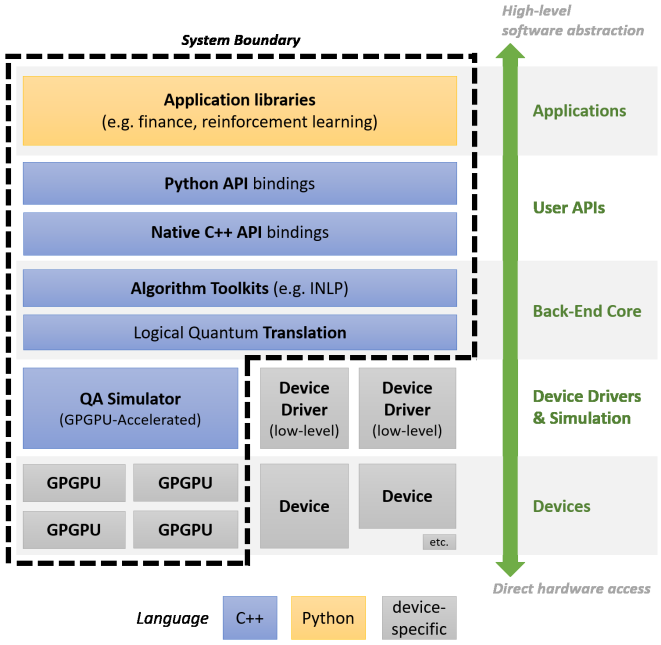

Fig. 3. An architectural slice from the QxBranch developer platform showing the software and hardware components in a System composed with the QxSQA ("QA Simulator"). The layered approach allows for problems written at higher layers of abstraction (e.g. via the "Application libraries") to run portably on any available computing device, whether they be simulators, annealers, or NISQ-era quantum computers.

B. Example problem: Maximum Clique

We present here a simple representative example: the Maximum Clique problem, which is NP-Hard in its optimization version[36]. A *clique* is a subset V' of fully-connected nodes in an arbitrary undirected graph defined by a set of nodes V and edges E. The maximum clique is the subset $V' \subseteq V$ with the most number of fully-connected nodes $\max |V'|$. Finding the maximum clique is an important problem with some real-world applications[37], and we use it here due to its simplicity and benchmark availability.

A possible formulation associates a binary variable x_i to each node in V such that $x_i = 1$ if the ith node is selected in V' (that is, $V' = \{i \mid x_i = 1\}$). We seek the solution vector $\boldsymbol{x}$ corresponding to the largest set V', as follows:

$$\text{maximize } |V'| \equiv \text{argmax}_{\boldsymbol{x}} \sum_{i \in V} x_i \tag{7a}$$

$$\text{subject to } \{i,j\} \in E \qquad \forall\ (i,j) \in V' \tag{7b}$$

where the constraint of Eq. 7b states that the set of selected nodes V' must form a clique (all edges in a fully-connected graph formed from the nodes V' must exist in E). Note that if the selected nodes V' did *not* form a clique, then there must be at least one selected edge that does not exist in the graph, or $\{i,j\} \notin E$. Conversely, this would imply that at least one edge was selected from the complementary edge set of the graph, $\{i,j\} \in \overline{E}$. Therefore, we can impose the constraints

Code Example 1. Example using our Python INLP framework to generate and run a Max-Clique optimization problem on the QxSQA backend.

```
from qxb.backends import QxSQA
from qxb.inlp import OptimizationProblem
import networkx as nx

# Define a max-clique problem (on random graph).
problem = OptimizationProblem(name="max-clique")
graph   = nx.gnp_random_graph(n=100, p=0.5)

# Associate x_i ∈ 0,1 to each node i.
variables = problem.binary_variables(
  prefix="x", size=graph.order())

# Maximize clique = ∑x_i (selected nodes).
problem.add_maximization_function(sum(variables))

# Loop through all edges not present in graph.
for (i, j) in nx.complement(graph).edges:
  # Edge does not exist; cannot be in a clique.
  problem.add_hard_constraint(
    (variables[i] * variables[j]) == 0
  )

# Solve using QxSQA backend GPU solver.
solver = QxSQA(samples=100, slices=1000)
results = solver.optimize(problem)

# Plot the results.
results.plot_histogram()
```

that no such edges may be selected:

$$x_i x_j = 0 \qquad \forall\ \{i,j\} \in \overline{E} \tag{8}$$

Thus, the problem defined by Eq. 7a subject to the constraints of Eq. 8 conforms to the structure of an INLP formulation, and can be easily implemented directly using our INLP framework. Code Example 1 shows this implementation. Internally, the INLP framework generates a formulation that exactly matches the direct Ising formulation of [38], and translates it to the desired backend (in this case, QxSQA). Note that this implementation requires only $|V|$ binary variables.

C. Simulation of Quantum Dynamics

A useful feature of PIMC-based quantum annealing simulation arises as the number of Trotter slices in Eq. 3 increases, where the approximation of the quantum effects induced by the transverse field during the quantum annealing process improves[39]. As the quantum Ising model is a discrete energy-based statistical mechanics model, the probability distribution of all of its possible states is defined by a Boltzmann distribution when in some thermodynamic equilibrium, as at the end of the annealing process. This is simulated directly by the acceptance rule (Eq. 5); in essence, a PIMC-based quantum simulation converges towards a good approximation of the Boltzmann distribution of a quantum annealing Hamiltonian[40]. Sampling from this Boltzmann distribution has previously been used to study the behavior of NISQ devices such as those by D-Wave Systems[8], [9], [41].

Boltzmann sampling also has value in Quantum Boltzmann Machines (QBMs)[42], a type of stochastic neural network function appproximator used to compress very high-dimensional function spaces, and learned by approximating values that are Boltzmann-distributed[43]. QBMs have many applications as predictive models[44], [45], as well as in deep reinforcement learning[46]. In general, any improvement in

the efficiency and quality of sampling from an appropriate Boltzmann distribution is beneficial to these applications[47].

D. Commercial Applications

Many real-world applications can be formulated within INLP and Ising formulations. Abundant examples range from canonical problems such as MaxCUT, SAT, and graph-based problems, through to scheduling[48] and planning[49], portfolio optimization[50], fault diagnosis[33], protein folding[51], election forecasting[45], and problems within machine learning including non-convex regularization[52] and deep neural networks[44].

In evaluating QxSQA, we ran a successful research project to develop two novel quantum-based applications in collaboration with a finance industry partner. This sought to demonstrate the applications on a cluster of classical computing resources, while providing a generic implementation capable of translating directly to NISQ hardware. In both cases, QxSQA showed competitive performance against classical solvers we tested. For these reasons, the combined capability of QxSQA and the QxBranch developer platform in easing development and production on a high-performance solver, and supporting evaluation on complementary quantum computing hardware is, we believe, novel in the quantum computing ecosystem.

IV. Experimental Results

We present a brief experimental analysis of QxSQA, demonstrating its performance and solution characteristics. These showcase the viability of the simulator as a development and analysis tool of novel algorithms, and its potential use as a production-ready solver back-end within our developer platform. We ran the experiments on Google Cloud-hosted, standard-sized Intel Haswell-based machines, each paired with an NVIDIA Tesla P100 GPGPU containing 16GB of processing memory.

A. Computation Time as Function of Parameters

QxSQA exhibits a linear increase in computation time when increasing either the number of sweeps or annealing steps. However, as Fig. 4 shows, the number of slices has a seemingly sub-linear impact in the execution time, notably exhibiting a reduced linear scaling beyond 512 slices. This is likely due to the way that QxSQA handles a single slice over multiple GPU warps, as discussed in Section II-C.

B. Canonical NP-Hard Problems

We evaluated QxSQA against random instances with planted solutions as well as benchmark instances of the Maximum Clique problem (Section III-B). For instances smaller than ∼30 qubits, a brute-force solver was used to verify that QxSQA could find optimal solutions reliably. Comparable tests on MaxCUT and Subset Sum problems — not reported here — showed similar results.

For the Max-Clique problem, we used a selection of different-sized problems from the DIMACS benchmark set[37]. Here, solution quality was measured simply as the ratio of the largest clique found divided by the known maximum clique for a given benchmark.[5] The results of these benchmarks in Fig. 5 showed that, without any other parameter tuning, QxSQA obtained cliques at least ∼85% as large as the maximum known size in under ∼30 s of wall-clock time (5×10^4 iterations) across all runs of all tested benchmarks (Fig. 5b). Computation time increased approximately linearly with iterations and problem size, with wall-clock times tested through to ∼120 s (2.5×10^5 iterations) for the largest instance of 4,000 nodes. Higher numbers of Trotter slices resulted in a less significant improvement of the solution quality (Fig. 5d).

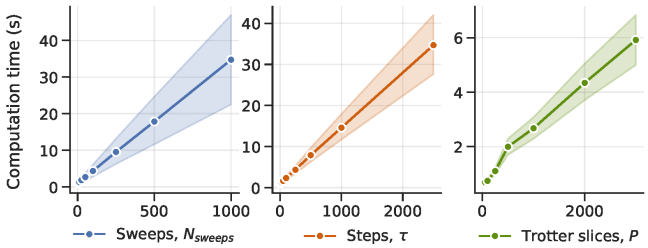

Fig. 4. Representative effect of QxSQA algorithm parameters on computation times. Each figure shows a different set at least 100 random problem instances averaged per point. The widening confidence bands reflect the additional scaling factors due to different problem sizes tested (N).

For such a meta-heuristic, more significant improvements arise from encouraging the algorithm to stochastically explore the space of possible solutions to avoid becoming stuck in local minima. This can be by annealing slowly over more iterations ($\tau * N_{\text{sweeps}}$, as in Fig. 5a); by adjusting the annealing schedule temperature T, transverse-field strength Γ, and annealing schedule functions; or even by softening or hardening constraints (for example, multiplying Eq. 8 by some constant), modifying the overall energy landscape. Each problem instance has a unique energy landscape that may be optimally explored by tuning these parameters; although in principle, an optimal setting may not exist for all, or even any, classes of problems, it can nonetheless be empirically useful to observe and report the performance effects of parameters on subsets of problems[53]. In Fig. 5, note that independently increasing the annealing steps and number of Trotter slices resulted in distinct, qualitative improvements in solution quality of Max-Clique instances known to be computationally hard.

On random instances of Max-Clique with planted solutions[6], QxSQA was able to find at least one optimal clique in all cases on all runs using the default parameter settings. This included on graphs of 8,100 nodes, finding the optimal clique in under 1 min (averaged across 10 runs), and of 16,002 nodes in under 7 min (averaged across 5 runs).

[5]For these tests, invalid (non-clique) solutions were discarded; in practice, imperfect but low-energy solutions returned by QxSQA are often valuable.

[6]Instances generated semi-randomly with planted local and global minima designed to trap solvers from finding the optimal solutions.

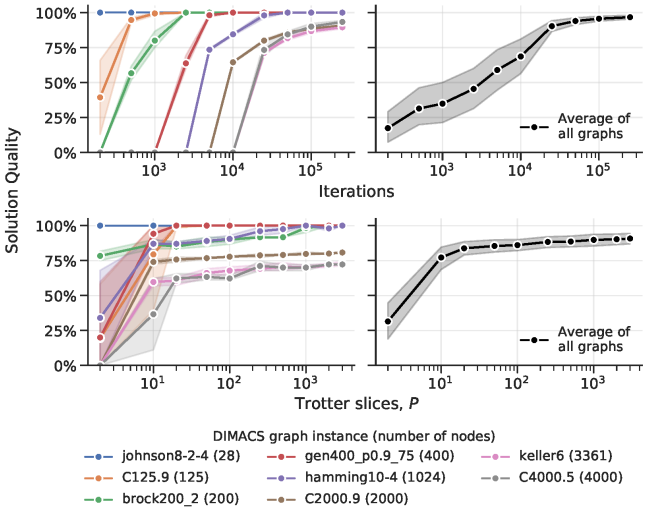

Fig. 5. Performance of QxSQA on a selection of DIMACS benchmark Max-Clique problems, averaged over 5 runs. (a, top-left) Parameters set to $T = 1 \times 10^{-20}$, $\Gamma_0 = 3$, $\Gamma_\tau = 1 \times 10^{-30}$, $N_{\text{sweeps}} = 100$, $P = 2000$, and iterations increasing with τ, (b, top-right) resulting in an improvement in quality with increased iterations. (c, bottom-left) Parameters set as in (a) with $\tau = 250$ and varying Trotter slices P, (d, lower-right) resulting in a modest improvement in quality with more slices.

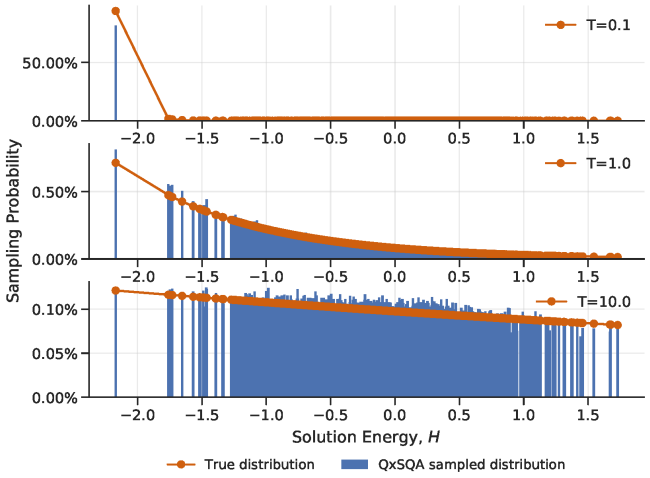

Fig. 6. Example distributions of 200,000 samples (for a random Hamiltonian of size $N = 10$) as returned by QxSQA, compared to the expected discrete Boltzmann distribution of the system for the given sampling temperature T.

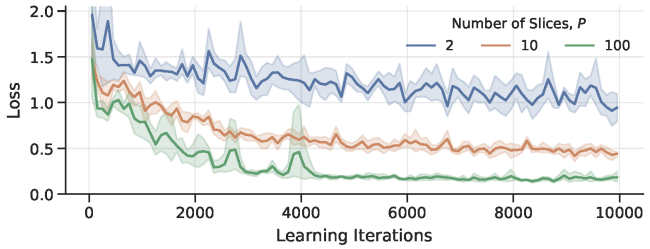

Fig. 7. Representative comparison of the quality of a QBM-based reinforcement learning policy learned using different levels of Boltzmann distribution sampling as provided by QxSQA. Each line is averaged across 15 runs. A loss of 0 implies a perfect function approximation of the true Q-value function.

C. Boltzmann Sampling

We tested two approaches to determine QxSQA's Boltzmann sampling accuracy. The first approach attempted a realistic simulation of a physical Boltzmann-distributed quantum annealer by running R simulations and sampling a random slice from each, representing a random sample from the Trotter-approximated system. The second approach reduced the sampling time substantially, with R samples collected from $\lceil \frac{R}{P} \rceil$ simulations, where each returned all P slices as samples – this is a factor of P faster, but relies on a computational artifact (Trotter slices) with no true physical analogue.

Both approaches resulted in similar sampling distributions, which closely matched their corresponding Boltzmann distributions for all tested Hamiltonians and classical temperatures T. Fig. 6 shows an example of the second approach where small random Hamiltonians were sampled and compared to the true Boltzmann distribution of solution energies (as computed using a brute-force solver). For larger Hamiltonians, tests were possible only for specifically constructed problems where the full set of solution energies was known; these also resulted in accurate Boltzmann-distributed samples.

The example of Fig. 7 demonstrates the effect of Boltzmann sampling on a novel QBM-based reinforcement learning algorithm developed as part of a commercial collaboration (Section III-D). Here, more Boltzmann-distributed samples from QxSQA resulted in a smoother convergence to a more accurate QBM-based function approximation. In this case, we showed that more samples led to a faster convergence to an optimal *policy* (including when compared against a classical simulated annealer), which was the primary metric of quality considered for our algorithm.

V. Summary & Future Work

Our GPGPU-Accelerated Simulated Quantum Annealer (QxSQA) was created to support software developers in developing, testing, and ultimately deploying novel quantum computing solutions to real-world problems. QxSQA finds low-energy solutions to integer, non-linear optimization problems of up to 2^{14} (16,384) unconstrained binary variables with quadratic interactions on a GPGPU, and functions also as an accurate sampler of Boltzmann distributions for machine learning applications. During a commercial collaboration we were able to evaluate QxSQA's use as a research and development tool for novel quantum-based optimization and machine learning algorithms. Coupled with the performance results observed, we suggest that QxSQA may be useful to solve industry problems on current, commodity hardware, and in support of analysis of future NISQ devices.

Further work is needed to understand performance on more applications, and to tailor the configuration and parameter tuning of the solver algorithm. We are currently investigating support for multiple GPGPU instances, and additional `CUDA`, CPU, and memory optimizations for larger ($\geq$ 32,768 variables) problems on available GPGPU hardware.

Acknowledgments

We would like to thank Duncan Fletcher, Marco Paini, and Sakthi Jayabalan for their contributions to the development of the simulator and to the preparation of this paper.

References

[1] F. Furini, E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann, N. Sahinidis, S. Vigerske, and A. Wiegele, "QPLIB: A library of quadratic programming instances," *Mathematical Programming Computation*, 2018. DOI: 10.1007/s12532-018-0147-4.

[2] M. Hodson, D. Fletcher, D. Padilha, and T. Cook, "Rapid prototyping with symbolic computation: Fast development of quantum annealing solutions," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2016. DOI: 10.1109/HPEC.2016.7761632.

[3] S. V. Isakov, I. N. Zintchenko, T. F. Rønnow, and M. Troyer, "Optimised simulated annealing for Ising spin glasses," *Computer Physics Communications*, vol. 192, pp. 265–271, 2015.

[4] A. Selby, "Efficient subgraph-based sampling of Ising-type models with frustration," *arXiv preprint*, Sep. 2014. arXiv: 1409.3934.

[5] S. Morino, *Sqaod*, Nov. 2018. [Online]. Available: https://github.com/shinmorino/sqaod.

[6] H. Goto, K. Tatsumura, and A. R. Dixon, "Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems," *Science Advances*, vol. 5, no. 4, 2019. DOI: 10.1126/sciadv.aav2372.

[7] S. Morita and H. Nishimori, "Mathematical foundation of quantum annealing," *Journal of Mathematical Physics*, vol. 49, no. 12, p. 125 210, 2008.

[8] S. Boixo, T. Rønnow, S. Isakov, Z. Wang, D. Wecker, D. Lidar, J. Martinis, and M. Troyer, "Quantum annealing with more than one hundred qubits," *arXiv preprint*, 2013. arXiv: 1304.4595.

[9] T. Albash, T. Rønnow, M. Troyer, and D. Lidar, "Reexamining classical and quantum models for the D-Wave One processor," *The European Physical Journal Special Topics*, vol. 224, no. 1, pp. 111–129, Feb. 2015. DOI: 10.1140/epjst/e2015-02346-0.

[10] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. Katzgrabeer, "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer," *Frontiers in Physics*, vol. 7, p. 48, 2019.

[11] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, "20k-spin Ising chip for combinational optimization problem with CMOS annealing," in *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*, IEEE, 2015, pp. 1–3.

[12] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018, ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79.

[13] Y. Wang, S. Wu, J. Zou, *et al.*, "Quantum annealing with Markov chain Monte Carlo simulations and D-Wave quantum computers," *Statistical Science*, vol. 31, no. 3, pp. 362–398, 2016.

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

[15] Martoňák, Roman and Santoro, Giuseppe E and Tosatti, Erio, "Quantum annealing by the path-integral Monte Carlo method: The two-dimensional random Ising model," *Physical Review B*, vol. 66, no. 9, p. 094 203, 2002.

[16] S. Tanaka, R. Tamura, and B. K. Chakrabarti, *Quantum spin glasses, annealing and computation*. Cambridge University Press, 2017.

[17] E. Crosson and A. W. Harrow, "Simulated quantum annealing can be exponentially faster than classical simulated annealing," in *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, 2016, pp. 714–723.

[18] J. King, S. Yarkoni, J. Raymond, I. Ozfidan, A. D. King, M. M. Nevisi, J. P. Hilton, and C. C. McGeoch, "Quantum annealing amid local ruggedness and global frustration," *Journal of the Physical Society of Japan*, vol. 88, no. 6, p. 061 007, 2019.

[19] J. Luitjens, *Increase performance with vectorized memory access*, Dec. 2013. [Online]. Available: https://devblogs.nvidia.com/cuda-pro-tip-increase-performance-with-vectorized-memory-access/.

[20] ——, *Faster parallel reductions on Kepler*, Feb. 2014. [Online]. Available: https://devblogs.nvidia.com/faster-parallel-reductions-kepler/.

[21] J. Marathe, *New compiler features in CUDA 8*, Nov. 2016. [Online]. Available: https://devblogs.nvidia.com/new-compiler-features-cuda-8/.

[22] D. Padilha, "Solving NP-Hard problems on an adiabatic quantum computer," *UNSW School of Mechanical and Manufacturing Engineering*, Oct. 2014. DOI: 10.13140/RG.2.2.19230.43842.

[23] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, pp. 268–308, Jan. 2001. DOI: 10.1145/937503.937505.

[24] M. Hodson, K. M. Zick, K. Jones, and D. Padilha, "A novel embedding technique for optimization problems of fully-connected integer variables," in *Fourth Conference in Adiabatic Quantum Computing (AQC 2015)*.

[25] R. Babbush, B. O'Gorman, and A. Aspuru-Guzik, "Resource efficient gadgets for compiling adiabatic quantum optimization problems," *Annalen der Physik*, vol. 525, no. 10-11, pp. 877–888, 2013.

[26] S. Pakin, "Targeting classical code to a quantum annealer," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Pro-*

gramming Languages and Operating Systems, ser. ASPLOS '19, Providence, RI, USA: ACM, 2019, pp. 529–543. DOI: 10.1145/3297858.3304071.

[27] K. Tanahashi, S. Takayanagi, T. Motohashi, and S. Tanaka, "Application of Ising machines and a software development for Ising machines," *Journal of the Physical Society of Japan*, vol. 88, no. 6, p. 061 010, 2019. DOI: 10.7566/JPSJ.88.061010.

[28] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, *et al.*, "Quantum annealing with manufactured spins," *Nature*, vol. 473, no. 7346, p. 194, 2011.

[29] IBM. (2019). Quantum systems – IBM Q, [Online]. Available: https://www.research.ibm.com/ibm-q/technology/devices/.

[30] E. A. Sete, W. J. Zeng, and C. T. Rigetti, "A functional architecture for scalable quantum computing," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, Oct. 2016, pp. 1–6. DOI: 10.1109/ICRC.2016.7738703.

[31] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," *Nature Physics*, vol. 14, no. 6, p. 595, 2018.

[32] V. Choi, "Minor-embedding in adiabatic quantum computation: I. the parameter setting problem," *Quantum Information Processing*, vol. 7, no. 5, pp. 193–209, Oct. 2008. DOI: 10.1007/s11128-008-0082-9.

[33] Z. Bian, F. Chudak, R. B. Israel, B. Lackey, W. G. Macready, and A. Roy, "Mapping constrained optimization problems to quantum annealing with application to fault diagnosis," *Frontiers in ICT*, vol. 3, p. 14, 2016. DOI: 10.3389/fict.2016.00014.

[34] A. Parent, M. Roetteler, and K. M. Svore, "Reversible circuit compilation with space constraints," *arXiv preprint*, 2015. arXiv: 1510.00377.

[35] S. Hadfield, "Quantum algorithms for scientific computing and approximate optimization," *arXiv preprint*, 2018. arXiv: 1805.03265.

[36] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, Springer, 1972, pp. 85–103.

[37] D. S. Johnson and M. A. Trick, *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*. American Mathematical Soc., 1996, vol. 26. [Online]. Available: http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark.

[38] C. S. Calude, M. J. Dinneen, and R. Hua, "QUBO formulations for the graph isomorphism problem and related problems," *Theoretical Computer Science*, vol. 701, pp. 54–69, 2017.

[39] S. V. Isakov, G. Mazzola, V. N. Smelyanskiy, Z. Jiang, S. Boixo, H. Neven, and M. Troyer, "Understanding quantum tunneling through quantum Monte Carlo simulations," *Physical review letters*, vol. 117, no. 18, p. 180 402, 2016.

[40] S. Morita and H. Nishimori, "Convergence theorems for quantum annealing," *Journal of Physics A: Mathematical and General*, vol. 39, no. 45, p. 13 903, 2006.

[41] V. S. Denchev, S. Boixo, S. V. Isakov, N. Ding, R. Babbush, V. Smelyanskiy, J. Martinis, and H. Neven, "What is the computational value of finite-range tunneling?" *Physical Review X*, vol. 6, no. 3, p. 031 015, 2016.

[42] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, "Quantum Boltzmann machine," *Physical Review X*, vol. 8, no. 2, p. 021 050, 2018.

[43] B. Sallans and G. E. Hinton, "Reinforcement learning with factored states and actions," *Journal of Machine Learning Research*, vol. 5, pp. 1063–1088, Aug. 2004.

[44] S. H. Adachi and M. P. Henderson, "Application of quantum annealing to training of deep neural networks," *arXiv preprint*, 2015. arXiv: 1510.06356.

[45] M. Henderson, J. Novak, and T. Cook, "Leveraging quantum annealing for election forecasting," *Journal of the Physical Society of Japan*, vol. 88, no. 6, p. 061 009, 2019.

[46] D. Crawford, A. Levit, N. Ghadermarzy, J. S. Oberoi, and P. Ronagh, "Reinforcement learning using quantum Boltzmann machines," *arXiv preprint*, 2016. arXiv: 1612.05695.

[47] M. Benedetti, J. Realpe-Gómez, R. Biswas, and A. Perdomo-Ortiz, "Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning," *Physical Review A*, vol. 94, no. 2, p. 022 308, 2016.

[48] D. Venturelli, D. J. Marchand, and G. Rojo, "Quantum annealing implementation of job-shop scheduling," *arXiv preprint*, 2015. arXiv: 1506.08479.

[49] F. Neukart, G. Compostella, C. Seidel, D. Von Dollen, S. Yarkoni, and B. Parney, "Traffic flow optimization using a quantum annealer," *Frontiers in ICT*, vol. 4, p. 29, 2017.

[50] G. Rosenberg, P. Haghnegahdar, P. Goddard, P. Carr, K. Wu, and M. L. De Prado, "Solving the optimal trading trajectory problem using a quantum annealer," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 6, pp. 1053–1060, 2016.

[51] A. Perdomo-Ortiz, N. Dickson, M. Drew-Brook, G. Rose, and A. Aspuru-Guzik, "Finding low-energy conformations of lattice protein models by quantum annealing," *Scientific reports*, vol. 2, p. 571, 2012.

[52] D. Padilha, "Training L0-regularised linear regression and classification models with a quantum annealer," in *Sixth Conference in Adiabatic Quantum Computing (AQC 2017)*.

[53] C. C. McGeoch, "Principles and guidelines for quantum performance analysis," in *Quantum Technology and Optimization Problems*, S. Feld and C. Linnhoff-Popien, Eds., Cham: Springer International Publishing, 2019, pp. 36–48, ISBN: 978-3-030-14082-3.