HyQSAT: A Hybrid Approach for 3-SAT Problems by Integrating Quantum Annealer with CDCL

Siwei Tan , Mingqian Yu , Andre Python , Yongheng Shang , Tingting Li , Liqiang Lu*, and Jianwei Yin*

Zhejiang University, P.R. China
{*siweitan, yumq, apython, yh_shang, litt2020, liqianglu, zjuyjw*}*@zju.edu.cn*

***Abstract*—Propositional satisfiability problem (SAT) is represented in a conjunctive normal form with multiple clauses, which is an important non-deterministic polynomial-time (NP) complete problem that plays a major role in various applications including artificial intelligence, graph colouring, and circuit analysis. Quantum annealing (QA) is a promising methodology for solving complex SAT problems by exploiting the parallelism of quantum entanglement, where the SAT variables are embedded to the qubits. However, the long embedding time fundamentally limits existing QA-based methods, leading to inefficient hardware implementation and poor scalability.**

In this paper, we propose HyQSAT, a hybrid approach that integrates QA with the classical Conflict-Driven Clause Learning (CDCL) algorithm to enable end-to-end acceleration for solving SAT problems. Instead of embedding all clauses to QA hardware, we quantitatively estimate the conflict frequency of clauses and apply breadth-first traversal to choose their embedding order. We also consider the hardware topology to maximize the utilization of physical qubits in embedding to QA hardware. Besides, we adjust the embedding coefficients to improve the computation accuracy under qubit noise. Finally, we present how to interpret the satisfaction probability based on QA energy distribution and use this information to guide the CDCL search. Our experiments demonstrate that HyQSAT can effectively support larger-scale SAT problems that are beyond the capability of existing QA approaches, achieve up to 12.62X end-to-end speedup using D-Wave 2000Q compared to the classic CDCL algorithm on Intel E5 CPU, and considerably reduce the QA embedding time from 17.2s to 15.7μs compared to the D-Wave Minorminer algorithm [11].

I. Introduction

A propositional satisfiability (SAT) problem is to find an assignment for each variable to satisfy a given Boolean formula. A typical SAT problem is composed of multiple clauses represented as a disjunction of Boolean variables. 3-SAT problem is a special case of the SAT problem that has no more than three variables in each clause. It is a fundamental problem in various applications, including artificial intelligence [16], circuit analysis [81], protein structure prediction [55] and computer security [69]. Since the 3-SAT problem is an NP-complete problem [1], the time complexity of classical algorithms increases exponentially with the numbers of clauses and variables [20].

Among various classical SAT algorithms [42], [48], [79], the conflict-driven clause learning (CDCL) algorithm is currently the state-of-the-art approach for large-scale SAT problems [43]. It performs tree-like searching and propagation with a strategy to memorize the conflict [43]. It solves the problem at around 8000μs over an M1 CPU (Figure 1). The CDCL computational time depends mainly on the warm-up stage [26], [68], where CDCL randomly searches the space without an efficient method to guide the exploration. Heuristics of CDCL also require learning the potential assignments of variables through searching [12], [25], [43], [49], which is time-consuming at the beginning.

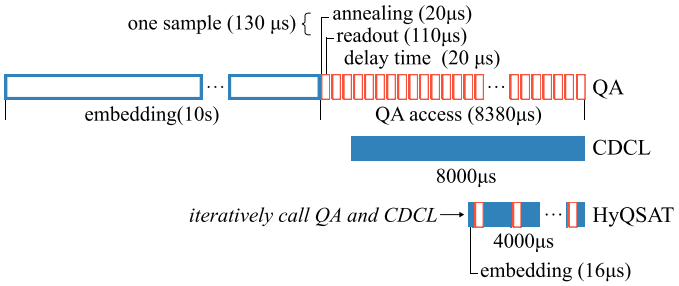


Fig. 1. End-to-end time to solve a 3-SAT problem using different approaches. The 3-SAT problem contains 128 variables and 150 clauses.

There are designs that leverage the parallelism of GPU [57], [61] or FPGA [31], [80] to accelerate the searching. However, solving a SAT problem is an iterative process where the next iteration depends on the previous one, which limits the overall speedup for these data-parallel-based architectures. Quantum computing offers an alternative to solve 3-SAT problems by providing exponential quantum speedup through qubit entanglement [7]. Quantum annealing (QA) has the theoretical advantage of low time complexity and high convergence speed compared to the state-of-the-art classical algorithm [5]. It solves the 3-SAT problem by encoding it into a minimization problem and considers the stabilized state as the solution [78]. Quantum annealer (QA hardware) is a specialized quantum computer to execute QA, which includes D-Wave 2000Q [47] and D-Wave Advantage [46], etc. It accommodates a relatively large number of physical qubits ($\sim$2000 qubits) compared to a gate-based quantum computer ($\sim$100 qubits). For example, both Grover algorithm [30] and Variational Quantum Eigensolver (VQE) [60] can be used to solve 3-SAT problems. They can only be implemented on a gate-based quantum computer, restricting their scope to small-scale problems.

Existing QA-based approaches suffer from long embedding time to map variables to physical qubits, resulting in inefficient hardware implementation and poor scalability [7], [8]. The

*Corresponding Author: Jianwei Yin, Liqiang Lu

embedding process takes about 10s using the state-of-the-art scheme of D-Wave Minorminer [11] (Figure 1), where iterative routing and adjustment steps tend to be time consuming [71]. The routing needs to calculate the shortest path between physical qubits allocated to the same variable. The adjustment that aims to embed all clauses into hardware is usually tedious. A complex embedding process hinders QA from handling large-scale 3-SAT problems, e.g., it is difficult for D-Wave 2000Q to solve the problem with 128 variables and 200 clauses according to our test.

The noise involved in the QA hardware inevitably brings inaccurate results, leading to more sampling time to improve accuracy. Theoretically, the noise comes from the environment [24], crosstalk [53], [75] and readout [22], which decrease the probability of finding the solution when solving SAT problems. To reduce noise overhead, it requires accessing multiple samples to maintain the high probability of getting correct results [73], e.g., local searching [65], or majority voting [63], which incurs additional time. Figure 1 shows that it requires $(10 + 110)\mu s \times 60 + 20\mu s \times 59 = 8380\mu s$ access time to execute 60 samples with a $20\mu s$ delay between two samples. Furthermore, we observe that for the problems with 128 variables and 200 clauses, the probability of getting correct results by D-Wave 2000Q is usually close to zero.

To overcome these challenges, we propose HyQSAT, a hybrid approach to enable end-to-end acceleration for tackling large-scale SAT problems, which leverages both advantages of QA and CDCL. We observe that the warm-up time of CDCL depends on the conflict frequency of clauses, while the embedding time of QA increases with the number of clauses. To overcome these limitations, HyQSAT tries to identify a subset of the clauses that are hard to solve by CDCL while being friendly to QA hardware. HyQSAT iteratively sends part of the clauses to QA hardware and interprets the QA results to accelerate the CDCL search. Our method requires 4000 μs only to solve the problem with an embedding time smaller than $16\mu s$ (see Figure 1). The speedup can be further improved for a larger problem scale.

To bridge the computational paradigm gap between quantum computing and classical computing, HyQSAT includes a frontend and a backend for the integration of QA and CDCL. The purpose of the frontend is to find "hard" clauses and embed them into QA hardware. We generate a clause queue according to their conflict frequency and embed the clause in order until the QA hardware is fully utilized. Since we do not need to embed all clauses, our embedding eliminates the iterative adjustment and routing, achieving linear time complexity of the number of clauses. When allocating variables to physical qubits, we also consider the physical qubits topology of QA hardware, which reduces the routing time of embedding. To mitigate noise overhead and improve computation accuracy, we propose to adjust the coefficient of the QA objective function. The backend reads the QA output and uses it to guide the CDCL search. Based on the energy distribution of QA hardware, we divide four types of clause satisfaction probability via a Gaussian Naive Bayes model. We then adopt different search strategies to prune the CDCL search space for different satisfaction probabilities, which significantly reduces the total number of iterations required to find the solution.

The contributions can be summarized as follows:

- We propose HyQSAT, a hybrid quantum-classical approach, which is the first work that realizes an end-to-end quantum speedup for solving SAT problems.
- We design the frontend and backend of HyQSAT to integrate QA with CDCL seamlessly. Our frontend effectively reduces the embedding time, while our backend improves the search efficiency of CDCL.
- We propose multiple noise optimization techniques for modern noisy intermediate-scale quantum (NISQ) devices, including a coefficient adjustment to increase the energy gap and a confidence interval partition to classify satisfaction probability.

Experiments show that HyQSAT can handle various large-scale SAT problems on 14 benchmarks from 7 different domains. It reduces 86.83% of the number of search iterations and achieves up to 12.62X end-to-end speedup using D-Wave 2000Q compared to classic CDCL on an Intel E5 CPU. Compared to the state-of-the-art D-Wave Minorminer [11], HyQSAT considerably reduces the QA embedding time from 17.2s to 15.7μs.

II. Background

A. 3-SAT problem

The 3-SAT problem consists of determining whether an assignment that satisfies a given Boolean formula exists. The formula is satisfiable if a correct assignment can be found. For the sake of simplicity, the value 1 will refer to *true* and 0 to *false* in the following sections. Generally, a Boolean formula C is represented in a conjunctive normal form with multiple clauses c_k. Each clause is a disjunction of Boolean variables $X = \{x_1, \ldots, x_n\}$,

$$\begin{aligned} &C = c_1 \wedge c_2 \ldots, c_{m-1} \wedge c_m \\ &c_k = l_1 \vee l_2 \vee l_3 \quad l_i = \{x_j, \neg x_j\}, \end{aligned} \tag{1}$$

where l_i is the literal that can choose x_j or its inverse $\neg x_j$ (not x_j). Each clause has no more than three Boolean variables.

Figure 2 (a) illustrates a 3-SAT problem, which contains two clauses c_1, c_2, and four Boolean variables x_1, x_2, x_3, x_4. Each clause takes three variables as input. The solution $x_1 = x_2 = 0, x_3 = x_4 = 1$ (Figure 2 (f)) satisfies c_1, c_2 and C. The solving time of the 3-SAT problem grows exponentially with the number of clauses m and the number of variables n. A nontrivial up bound of the time complexity is $\mathcal{O}(m^2 2^{n-2\sqrt{n/log_2 n}})$ [20].

B. CDCL Algorithm

The state-of-the-art classical SAT algorithm is CDCL. It performs a tree-like searching (Figure 2 (b)). Given a 3-SAT problem, an iteration of the searching consists of three steps: *a) Decision.* A variable is picked and assigned with 1 or 0 value (e.g. $x_1 = 0$, $x_2 = 0$), where many heuristics are

HyQSAT is publicly available on https://github.com/JanusQ/HyQSAT.

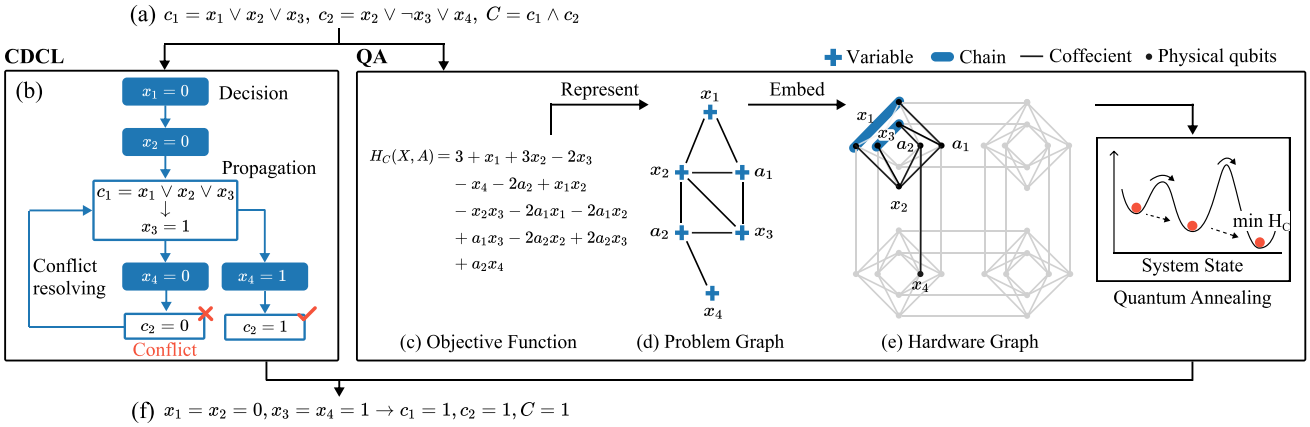


Fig. 2. Solving a 3-SAT problem using CDCL and QA. (a) The input SAT problem; (b) the CDCL algorithm that includes iterative decision, propagation, and conflict resolving steps; (c) the objective function for QA minimization; (e) the problem graph of the objective function; (e) the embedding results on QA hardware topology; (f) the final solution.

proposed [25], [49] *b) Propagation*. When a clause has only one unassigned variable, this variable is deduced to satisfy this clause. For example, after x_1 and x_2 are assigned 0, $x_3 = 1$ is deduced to satisfy $c_1 = x_1 \vee x_2 \vee x_3$. *c) Conflict Resolving*. If an unsatisfied clause has no unassigned variable (e.g. $c_2 = 0$ after x_4 is assigned 0), conflict arises. The decision history is memorized using the Davis-Putnam resolution [23], [43] to avoid repeated conflicts. The solver will then backtrack to the former step.

Warm up-stage. A key advantage of CDCL is that it memorizes the conflict to prune the search space. Many heuristics [25], [49] guide the search by learning from previous conflict clauses. For example, VSIDS [49] achieved around 4X speedup compared to the random search method. However, the learning mechanism works poorly at the beginning and has to randomly search for several iterations since there are not enough conflict clauses to learn. Recent works proposed a warm-up stage that searches the first few iterations using local search [12], deep learning [38], and look-ahead mechanism [32]. However, they show limited improvement due to the high complexity of the extra computation, e.g., inference of the deep learning model.

C. Quantum Annealing Algorithm

QA can find the global minimum of a two-degree objective function, which has $\mathcal{O}(e^{\sqrt{L}})$ [51] time complexity compared to the $\mathcal{O}(e^{L})$ time complexity of classical algorithms. Current QAs solve a 3-SAT problem C by encoding it into a minimization problem as follows:

$$\arg\min_{X} \ H_C(X) = I + \sum_{i=1}^{L} B_i x_i + \sum_{i=1}^{L} \sum_{j=i+1}^{L} J_{i,j} x_i x_j, \qquad (2)$$

where $I, B_i, J_{i,j}$ represents the intercept, the coefficient of a one-order item, and the coefficient of a two-order item, respectively. When $\min H_C = 0$, problem C is satisfiable, with the assignments of $\min H_C$ corresponding to the problem solution. When $\min H_C > 0$, problem C is unsatisfiable.

Mathematically, any 3-SAT problem can be encoded into Equation 2, where each clause is decomposed into two sub-clauses by introducing an auxiliary variable. For example, clause c_k in Equation 1 is decomposed into

$$\begin{aligned} c_{k,1} &= a_k \leftrightarrow (l_1 \vee l_2) \\ c_{k,2} &= l_3 \vee a_k, \end{aligned} \qquad (3)$$

where $\leftrightarrow$ refers to the logical equality. It means that if a_k equals $l_1 \vee l_2$, $c_{k,1}$ is satisfied. Each sub-clause is then encoded into a two-degree objective function as follows:

$$\begin{aligned} H_{c_{k,1}}(a_k, x_1, x_2) &= a_k + H_{l_1}(x_1) + H_{l_2}(x_2) - 2a_k H_{l_1}(x_1) \\ &\quad - 2a_k H_{l_2}(x_2) + H_{l_1}(x_1) H_{l_2}(x_2) \\ H_{c_{k,2}}(a_k, x_3) &= 1 - a_k - H_{l_3}(x_3) + a_k H_{l_3}(x_3), \end{aligned} \qquad (4)$$

where $H_{l_i}(x_i) = x_i$ for $l_i = x_i$ and $H_{l_i}(x_i) = 1 - x_i$ for $l_i = \neg x_i$. Sub-clause $c_{k,j}$ is satisfiable if the global minimum of its objective function $\min H_{c_{k,j}}$ is 0. The overall objective function of the 3-SAT problem is formulated as the sum of all sub-clauses, as follows:

$$H_C(X, A) = \sum_{k=1}^{m} (\alpha_{k,1} H_{c_{k,1}}(X, A) + \alpha_{k,2} H_{c_{k,2}}(X, A)), \qquad (5)$$

where $\alpha_{k,1} > 0$ or $\alpha_{k,2} > 0$ is the coefficient of each sub-clause and A is the set with all the auxiliary variables introduced. Figure 2 (c) illustrates the objective function of the problem in Figure 2 (a).

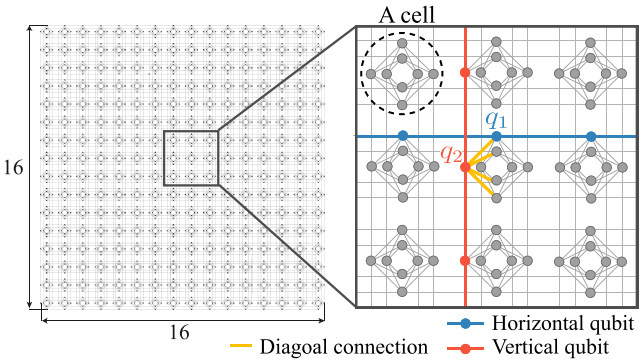


Fig. 3. Hardware graph of the D-Wave 2000Q.

D. Real-world QA Hardware

The QA algorithm can be implemented in a real physical quantum system (described in the Ising model [45]). The objective function of the minimization problem is represented as a problem graph (Figure 2 (d)), where each vertex is a variable or auxiliary variable, and two vertices are connected if they have a non-zero quadratic coefficient. The coefficients B_i and $J_{i,j}$ correspond to the weights of the vertices and edges, respectively. This problem graph is then embedded in the hardware graph, as shown in Figure 2 (e). The *embedding* aims to find a mapping from each vertex of the problem graph to several physical qubits (qubits) to enable the implementation of a non-zero quadratic coefficient on the QA chip topology. This paper defines *qubit chain* as the qubits that are connected to represent the same variable or auxiliary variable. Figure 2 (e) shows that two qubits are used as a qubit chain for variable x_1. In addition to the embedding step, there is a *normalization* step aiming at scaling the weights of vertex (B_i) and edge ($J_{i,j}$) to target ranges, where $B_i \in [-2, 2]$, $J_{i,j} \in [-1, 1]$.

In this work, we use D-Wave 2000Q QA topology, which is one of the most popular QA chips [47]. This topology is represented as a Chimera hardware graph with 2048 qubits in a 16×16 grid cell (Figure 3). Each cell contains 8 qubits (vertices) with two types of qubits: ① 4 horizontal qubits (e.g. q_1) that are located on the horizontal lines; ② 4 vertical qubits (e.g. q_2) that are positioned on the vertical lines. Each horizontal (vertical) qubit is connected with the other 4 vertical (horizontal) qubits via 4 diagonal edges, which refer to couplers in QA hardware.

III. HyQSAT Overview

To seamlessly integrate QA with CDCL, HyQSAT overcomes the following challenges:

- The solving time of CDCL is bounded by the clauses that are difficult to solve in the warm-up stage. It is important to identify these hard clauses and deploy them to QA. During deployment, QA hardware also features a specific topology, which restricts the mapping from variables to qubits.
- Modern NISQ computers always involve hardware noise, which may change the final value of each qubit. To avoid redundant sampling time and improve computation accuracy, we need to optimize the noise without executing multiple samples iteratively.
- The information from QA hardware cannot be directly used to determine the satisfaction due to noise. This requires the development of an interpretation mechanism to guide the CDCL search using QA results.

Assuming the total number of iterations is K for classic CDCL (estimated based on the numbers of variables and clauses), we empirically consider the first $\sqrt{K}$ iterations as the warm-up stage, where we employed QA on these $\sqrt{K}$ iterations to guide the CDCL search, with the remaining iterations applied in classic CDCL. By solving hard clauses in QA, the number of remaining iterations can be considerably reduced.

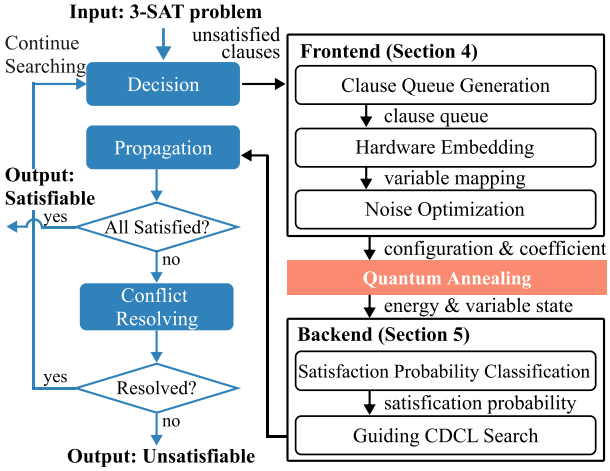


Fig. 4. HyQSAT workflow.

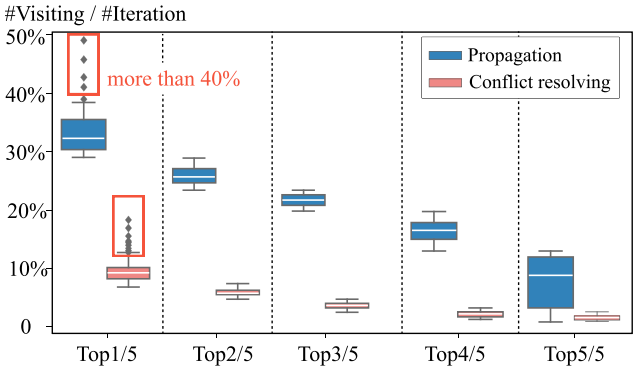
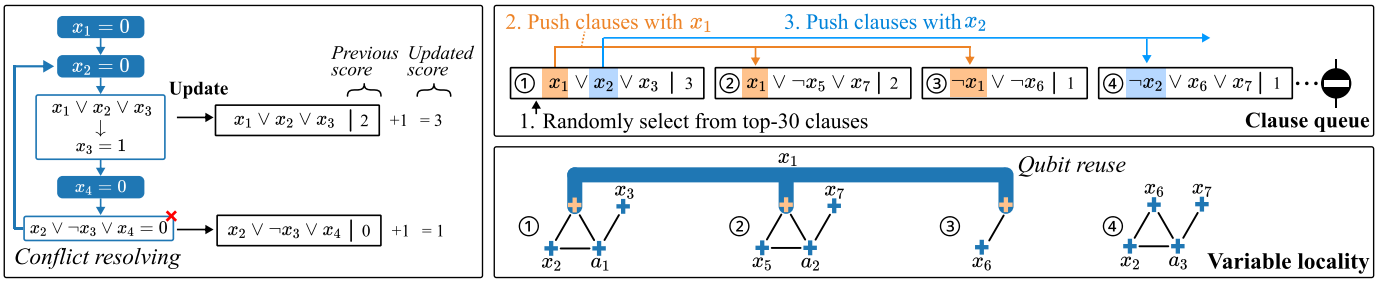


Fig. 5. Visiting frequency of clauses in the propagation step and the conflict resolving step using 100 3-SAT problems that contain 860 clauses and 200 variables from [33]. Clauses are divided into five parts based on their number of visits.

Figure 4 describes the overall workflow. HyQSAT features a cross-iterative process, where the CDCL algorithm leverages the interpretation results of QA to guide the search and sends new clauses to the QA hardware for acceleration. The frontend receives unsatisfied clauses from the decision step and finds hard clauses that need to be deployed to the QA hardware. We perform a breadth-first traversal to generate a clause queue, where the head clauses have higher priority to be accelerated by QA. We then embed the clauses following the order to the QA hardware under topology constraints. Our embedding scheme effectively eliminates the time-consuming routing and adjustment, which reduces the embedding time to linear complexity. To minimize noise overhead, we also adjust the coefficients in the QA objective function.

The frontend generates the embedding configuration with normalized coefficients and sends them to the QA hardware for acceleration. QA only needs to execute one sample rather than multiple samples as errors can be resolved during the next CDCL conflict resolving. The backend reads the outputs, e.g., energy, and variable state, to guide the CDCL exploration. In the backend, we estimate the satisfaction probability of



(a) Activity score updating (b) Breadth-first traversal

Fig. 6. An example of a clause queue generation. (a) an example of activity score updating during the conflict resolving, where activity scores of clauses involved in the resolving are increased; (b) clause queue generated by a breadth-first traversal which starts from a clause with a top-30 activity score.

the embedded clauses and adapt several feedback strategies. Similar to classic CDCL, HyQSAT follows propagation and conflict resolving steps. If all clauses are satisfied, HyQSAT outputs "satisfiable", or it tries to resolve the conflict and returns to the decision step. In a word, HyQSAT algorithm takes advantage of both quantum computing and classical computing.

IV. Frontend: from CDCL to QA

The embedding process of existing QA approaches can be time-consuming or even fail due to limited qubits and couplers. To minimize the embedding time, we choose to only embed a part of the clauses to QA hardware, which are hard to solve by CDCL. In Section IV-A, we propose to generate a clause queue that helps identify clauses with the following features: a) The satisfaction solution of the clauses is hard to find by classic CDCL. In contrast, it has a preference for the QA algorithm; b) When embedding to hardware, the clauses are compatible under the hardware graph topology, which can utilize a large proportion of the qubits and couplers.

After generating the clause queue, we deploy the clauses to QA hardware, following the queue order (Section IV-B). When mapping variables to qubits, we try to maximize the locality of variables and maintain a relatively short qubit chain to improve qubit utilization. Finally, Section IV-C introduces our optimization technique to reduce hardware noise overhead.

A. Clause Queue Generation

In general, the search time of CDCL is determined by how many times a clause is visited. Figure 5 evaluates the number of times that clauses are visited during the entire CDCL search. Here the top 1/5 clauses are visited in 42% iterations, which are decomposed into 33% in propagation and 9% in conflict resolving. Some clauses are visited in more than 50% of the iterations. We also observe that the propagation time and the conflict resolving time are positively correlated. By deploying the frequently-visited clauses to QA hardware, we can effectively reduce the overall search time.

We start the clause queue generation by quantitatively evaluating how frequently a clause is visited. Since the repeated visiting comes from the conflicts, we maintain an activity score for each clause, which measures the conflict frequency during the search. The activity score is set to 1 initially. When resolving a conflict clause, the search is backtracked to a predecessor variable assignment. The activity score of the involved clauses in the backtrack increases by a constant. Figure 6 (a) illustrates an example where x_1 and x_2 are both assigned with 0 in two decision steps. To make $x_1 \wedge x_2 \wedge x_3$ equal to 1, x_3 is deduced as 1. After x_4 is also assigned, a conflict arises as $x_2 \vee \neg x_3 \vee x_4$ is unsatisfied, which needs to backtrack to the assignment of x_2. Such conflict changes the activity score of the involved clauses, where both scores associated with $x_1 \vee x_2 \vee x_3$ and $x_2 \vee \neg x_3 \vee x_4$ increase by 1.

To build the queue, we randomly select one clause from the clauses with top-30 activity scores as the queue head. A random selection avoids deploying the same clauses in the queue when there is no updating of scores. After the selection, the clause queue is generated via breadth-first traversal. We sequentially push the clauses that share the same variable with the current clause, which maximizes the variable locality when embedding to hardware. Once all the variables of the current clause are visited, the process explores the next clause in the queue. The traversal ends until the queue size exceeds a threshold that is determined by the embedding capacity of QA hardware. As illustrated in Figure 6 (b), a queue starts with the clause $x_1 \vee x_2 \vee x_3$. The clauses $x_1 \vee \neg x_5 \vee x_7$ and $\neg x_1 \vee \neg x_6$ that also share x_1 are visited and pushed into the queue. Then, the traversal pushes the clauses that share x_2, x_3. After the variables of the clause $x_1 \vee x_2 \vee x_3$ are completed, the traversal goes to the next clause $x_1 \vee \neg x_5 \vee x_7$.

The clause queue aims to improve search efficiency while keeping it hardware-friendly. In each iteration of CDCL, QA accelerates the clauses from the current queue (the capacity of QA is around 170 clauses), where the top-30 clauses are dynamically updated when conflict arises. By doing so, we can easily maintain the queue without thoroughly visiting all clauses. Since we maximize the variable locality during the queue generation, the utilization of qubits is improved when mapping adjacent clauses to the hardware graph. In other words, the couplers of the hardware graph will be efficiently reused to enable the connection between variables. For example, by adjacently embedding the first three clauses, we can reuse the qubits of x_1 (Figure 6 (b)).

B. Hardware Embedding

The clauses in the queue are embedded in the QA hardware graph, which allocates each variable to a qubit chain to match the problem graph. For a variable, we define that its chain is

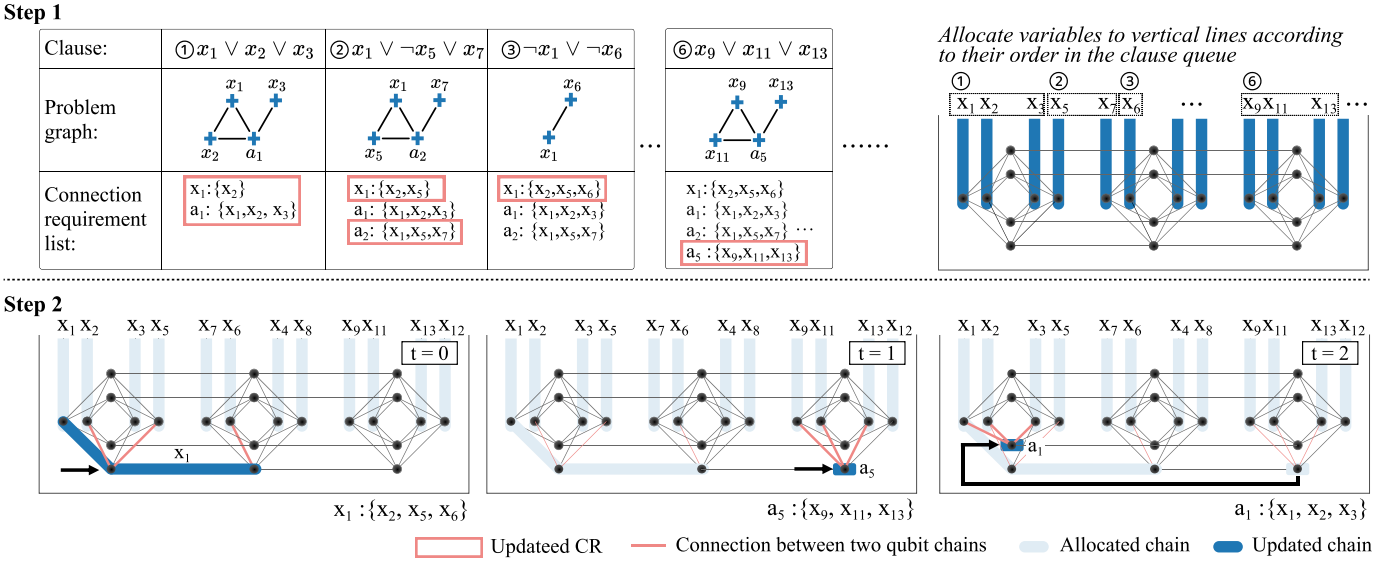


Fig. 7. Embedding the clause queue to Chimera hardware graph with 12 vertical lines and 4 horizontal lines.

composed of multiple qubits that are connected via a vertical line and a horizontal line. For the sake of simplicity, we only consider left-to-right connections when connecting qubits of two variables. Figure 7 (Step 2, t = 0) shows the qubit chain of variable x_1, which consists of three qubits with one on the vertical line and two on the horizontal line. Here, x_1 is physically connected to x_2, x_5, x_6 via a Chimera graph topology (red lines). For auxiliary variables, we limit their allocation to horizontal lines, which allows for allocating more variables to vertical lines. Figure 7 (Step 2, t = 1) shows that the auxiliary variable a_5 is allocated to a qubit in the horizontal line on the bottom, where it is connected to x_9, x_{11}, and x_{13}.

To reduce computation overhead and maximize hardware utilization, we propose an embedding scheme in two steps, following the allocation rules mentioned above. This scheme focuses on the Chimera topology of the hardware graph, e.g., D-Wave 2000Q, where each cell is located on 4 horizontal lines and 4 vertical lines. In **step 1**, we pop clauses from the clause queue and allocate variables to the vertical lines according to their order in the queue. Then, we define a connection requirement list (*CRL*) to record the connection requirements of qubit chains during the embedding. A connection requirement is represented as $x_i : \{x_j, \ldots x_k\}$, which means that the qubit chain of variable x_i is required to connect to the qubit chains of variables $\{x_j, \ldots x_k\}$. *CRL* is updated after a clause is popped according to its objective function. After all vertical lines are used, **step 2** aims to meet the requirement in *CRL* by allocating variables to the qubits in horizontal lines. The allocation starts with the bottom horizontal line from left to right and builds the connection with vertical lines via the diagonal coupler. Concretely, we adopt a greedy method that always tends to maximize the utilization of qubits of each horizontal line by allowing out-of-order allocation of variables. Similarly, when one horizontal line is fully utilized, the embedding process goes to the next horizontal line until *CRL* is empty.

Figure 7 shows an illustrative flow example. In **step 1**, we pop clauses and allocate their variables to vertical lines. The first clause popped is $x_1 \vee x_2 \vee x_3$. Its three variables are allocated to the first three vertical lines. Based on the problem graph of the clause, *CRL* is $x_1 : \{x_2\}$ and $a_1 : \{x_1, x_2, x_3\}$. When popping the second clause, CRL is updated with $x_1 : \{x_2, x_5\}$ and $a_2 : \{x_5, x_7\}$. The popping continues until 12 variables are allocated to all 12 vertical lines. In **step 2**, the embedding starts from the left-most qubit x_1. For x_1, its CRL is $x_1 : \{x_2, x_5, x_6\}$. Thus, we allocate x_1 to the qubit chain with three qubits to build the connection with $\{x_2, x_5, x_6\}$. In t = 1, since the left two qubits are used by x_1, to fully utilize the bottom horizontal line, our greedy method allocates auxiliary variable a_5 to the last qubit in the bottom horizontal line to meet $a_5 : \{x_9, x_{11}, x_{13}\}$ (Requirements of a_1 and a_2 cannot be satisfied by this qubit). Since the bottom horizontal line is fulfilled, the allocation of a_1 moves to the second horizontal line (t = 2).

Comparing to previous embedding schemes. Assuming the qubit number is N_q, and the number of nodes associated with the problem graph is N_p, the greedy embedding scheme [9] has $\mathcal{O}(N_q^4)$ time complexity, which finds the local optimal allocation with $\mathcal{O}(N_q^3)$ complexity in each iteration. Minorminer [11] requires $\mathcal{O}(N_q N_p)$ iterations to embed all clauses. In each iteration, the routing has $\mathcal{O}(N_p log N_p)$ complexity. Thus, Minorminer shows overall $\mathcal{O}(N_q N_p^2 log N_p)$ complexity. The complexity of our scheme consists of two parts: a) popping clauses and allocating their variables to vertical qubits, where the time complexity is linear to the number of the vertical qubits; b) traversing each horizontal qubit once to satisfy a connection requirement, which only has a constant time complexity via a hash table. By benefiting from our topology-aware mapping scheme, our time complexity is $\mathcal{O}(N_q)$. As the allocation to the vertical lines follows the order of the queue, we can maximize the variable locality during embedding, thus further improving the hardware utilization.

Since the auxiliary variables only connect three other variables, they are only allocated to the horizontal lines, saving qubits on vertical lines.

C. Noise Optimization

Theoretically, a noise-free QA is able to find the global minimum. However, noise may trap QA at a local minimum, reducing the computation accuracy. Part of noise is amplified in the normalization step that rescales the coefficients of the objective function to fit the hardware constraints, making the energy curve less steep. *Energy gap* is used to measure this steepness, which is defined as the minimum output of the objective function (Equation 5) when the clause is unsatisfiable. A higher energy gap suggests a higher probability that QA can converge to the global minimum [3], [24], [50], [73]. Mathematically, the normalized objective function H_C^{norm} can be expressed as:

$$\begin{aligned} H_C^{norm}(X,A) &= \frac{H_C(X,A)}{d_*} \\ d_* &= \max\left\{\max_{x\in X}\left(\frac{|B_x|}{2}\right), \max_{x_1,x_2\in X}(|J_{x_1,x_2}|)\right\}, \end{aligned} \tag{6}$$

where d_* is the maximum coefficient among the items $\frac{|B_x|}{2}$ and $|J_{x_1,x_2}|$. Thus, after the normalization, the final energy gap is divided by d_*. All prior works [7], [8] set the coefficient $\alpha_{i,j}$ of the objective function of each sub-clause to 1, where some coefficients may be extremely large compared to other coefficients, thus, leading to a low energy gap after dividing d^*. We try to increase the small coefficients in H_C, while keeping d_* as the same. We first use $\alpha_{i,j} = 1$ to calculate the objective function and define $d_{i,j}$ for each clause $c_{i,j}$ as follows:

$$d_{i,j} = \max\left\{\max_{x\in c_{i,j}}\left(\frac{|B_x|}{2}\right), \max_{x_1,x_2\in c_{i,j}}(|J_{x_1,x_2}|)\right\}, \tag{7}$$

which corresponds to the maximum coefficient of the items involved in the sub-clause $c_{i,j}$ objection function. Since $d_* \geq d_{i,j}$, we then increase each sub-clause coefficient $\alpha_{i,j}$ to $\frac{d_*}{d_{i,j}}$, and modify the objective function.

We use $c_1 = x_1 \vee x_2 \vee x_3$ as a clause example to illustrate the coefficient optimization. First, c_1 is decomposed into two sub-clauses $c_{1,1} = a_1 \leftrightarrow x_1 \vee x_2$ and $c_{1,2} = a_1 \vee x_3$ with $\alpha_{1,1} = \alpha_{1,2} = 1$. We then calculate the coefficient of each item in H_C and obtain d_*, $d_{1,1}$, $d_{1,2}$ according to Equation 6 and Equation 7, where

$$\begin{aligned} H_C(X,A) &= \alpha_{1,1}H_{c_{1,1}}(X,A) + \alpha_{1,2}H_{c_{1,2}}(X,A) \\ &= x_1 + x_2 - x_3 + x_1x_2 - 2a_1x_1 \\ &\quad - 2a_1x_2 + a_1x_3 + 1\ , \\ d_* &= 2,\ d_{1,1} = 2,\ d_{1,2} = 1. \end{aligned} \tag{8}$$

For example, the objective function of $c_{1,2}$ has items a_1, x_3, and a_1x_3 with coefficients 0, -1, and 1, respectively. Thus, $d_{1,2} = 1$. We adjust the sub-clause coefficients to $\alpha'_{1,1} = \frac{d^*}{d_{1,1}} = 1$ and $\alpha'_{1,2} = \frac{d^*}{d_{1,2}} = 2$. The optimized objective function becomes:

$$\begin{aligned} H'_C(X,A) &= \alpha'_{1,1}H_{c_{1,1}}(X) + \alpha'_{1,2}H_{c_{1,2}}(X) \\ &= x_1 + x_2 - 2x_3 - a_1 + x_1x_2 \\ &\quad - 2a_1x_1 - 2a_1x_2 + 2a_1x_3 + 2\ , \end{aligned} \tag{9}$$

where $d'_* = d_*$. Our approach only needs one additional calculation of the objective function. While some methods may also increase the energy gap, their iterative approaches to finding better coefficients are time-consuming [73] or require a large number of qubits for quantum error-correcting codes [4], [77].

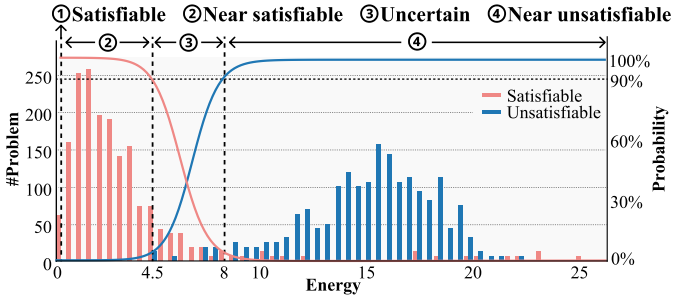


Fig. 8. Using a Gaussian Naive Bayes model to fit the energy distribution. Each problem is randomly generated with 50-200 variables and 50-160 clauses based on [33].

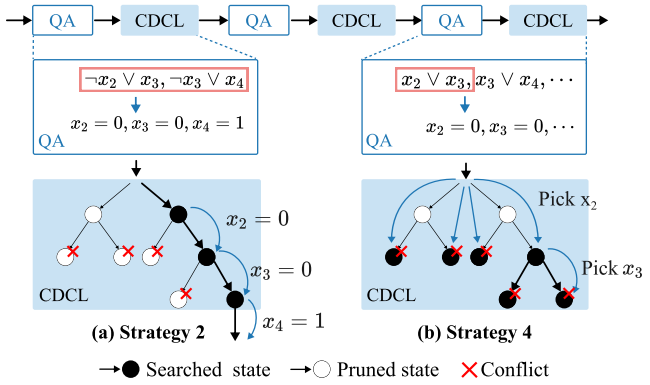


Fig. 9. Illustrative example using feedback strategy 2 and strategy 4 to prune the search space.

V. Backend: From QA to CDCL

After quantum annealing, we obtain two types of information: 1) the final assignment of each variable, obtained by measuring the qubit state; 2) the value of the minimized objective function, represented as the energy of the QA system. The backend leverages this information to guide the CDCL search, reducing the overall latency for solving the SAT problem. In this section, we first classify four basic types of satisfaction probability of the embedded clauses according to the energy distribution. We then propose several feedback strategies to accelerate the CDCL search based on these types.

A. Satisfaction Probability Classification

The SAT problem can only be satisfiable if the objective function can reach a global minimum of 0 (Equation 5). However, for NISQ devices, zero energy is hard to achieve

due to hardware noise. Some satisfiable clauses may be misclassified as unsatisfiable. To avoid misleading interpretation, we propose to classify the embedded clauses into different types according to their probability of being satisfiable.

As the energy distribution of QA hardware varies with the noise overhead and device scale, we use an empirical method to estimate the satisfaction probability. We test 1000 satisfiable problems and 1000 unsatisfiable problems with different numbers of clauses and variables on D-Wave 2000Q, and plot their energy distribution (Figure 8). We then apply a Gaussian Naive Bayes model to fit this distribution. Specifically, we choose 90% as the factor to partition the energy axis into multiple confidence intervals. For example, the partition point '4.5' means that for a 3-SAT problem with an energy value of 4.5, it has a 90% probability to be satisfiable. Finally, we interpret the results of embedded clauses by identifying the following confidence intervals:

- **Satisfiable** problem with interval $[0, 0]$.
- **Near satisfiable** problem with interval $(0, 4.5]$.
- **Uncertain** problem with interval $(4.5, 8]$.
- **Near unsatisfiable** problem with interval $(8, +\infty]$.

B. Guiding CDCL Search

HyQSAT is a cross-iterative process, including a) the QA part which accelerates the clauses from the clause queue and sends the satisfaction probability with variable assignments to the CDCL part; b) the CDCL part that receives the interpretation results of QA to guide the search, and sends new clauses to QA for acceleration. Depending on the number of embedded clauses and their satisfaction probability, we divide them into four cases and propose several feedback strategies to prune the CDCL search space.

<table>
<tr><th></th><th>Satisfiable</th><th>Near satisfiable</th><th>Uncertain</th><th>Near unsatisfiable</th></tr>
<tr><td>All embedded</td><td>Strategy 1</td><td rowspan="2">Strategy 2</td><td rowspan="2">Strategy 3</td><td rowspan="2">Strategy 4</td></tr>
<tr><td>Not all embedded</td><td></td></tr>
</table>

Feedback strategy 1 is adopted when all clauses are embedded and demonstrated as satisfiable, which ends the CDCL search and outputs the variable assignment. **Feedback strategy 2** is applied in the case that only part of the clauses are embedded and satisfiable, or the QA result is near-satisfiable. For this case, we maintain the variable assignments during the CDCL search as these assignments have the highest probability of being close to the final solution. Figure 9 (a) illustrates a case where the assignments $\{x_2, x_3, x_4\} = \{0, 0, 1\}$ from QA can be directly used in the next search state of the CDCL part instead of thoroughly visiting the entire space. **Feedback strategy 3** is executed when QA cannot distinguish the satisfaction of clauses, which contributes to no acceleration for the classic CDCL. **Feedback strategy 4** helps to prune the unnecessary CDCL search state when the embedded clauses are hard to be satisfied. Concretely, near-unsatisfiable implies that any assignment of the variables involved in QA cannot satisfy the embedded clauses. Thus, we prioritize these variables in the CDCL search to speed up the backtrack from the conflict state. Figure 9 (b) illustrates a case where the clause $x_2 \vee x_3$ shows near-unsatisfiable results. We choose to assign these two variables in the decision of CDCL so that the search can quickly reach the conflict state without visiting x_4.

VI. Evaluation

A. Experiment Setup

Benchmark. We evaluate HyQSAT with 14 major benchmarks from various domains, including graph colouring (CG), circuit fault analysis (CFA), block planning (BP), inductive inference (II), integer factorization (IF), cryptography (CRY), and artificial intelligence (AI), which are listed in Table I. Their number of variables ranges from 48 to 6325, and the number of clauses ranges from 186 to 131973. For AI problems, each benchmark contains specially-designed 3-SAT problems that are hard to solve using the classic CDCL algorithm. Due to the access limitations of D-Wave, we randomly select 10 problems per benchmark. All test data are derived from open-source SAT benchmarks [33], [67].

Implementation. We apply MiniSAT [70] and KisSAT [14] as classic CDCL baselines. MiniSAT is a popular CDCL implementation written in C++ with VSIDS heuristic [49]. KisSAT is another implementation which applies similar learning mechanisms [43] and heuristics [40] as MiniSAT and wins the SAT competition in 2022. We build a noise-free HyQSAT simulator based on the D-Wave's QA simulator with a long timeout to avoid simulation error [19]. We also implement HyQSAT on a real-world QA platform using D-Wave 2000Q via its internet API. All experiments of classical algorithms are conducted on Intel E5 1.7GHz CPUs with 32Gb memory. We set the annealing time and the readout time of QA to $20\mu s$ and $110\mu s$, respectively [15], [65]. The flux-bias offset and readout thermalization of QA are both configured to 0.

Methodology. For the noise-free HyQSAT simulator, we compare the number of iterations with the classic CDCL algorithm. With a total number of iterations K of classic CDCL, we empirically consider the first $\sqrt{K}$ iterations as the warm-up stage. We observe that deploying more iterations to HyQSAT does not contribute to a computational gain. The CDCL heuristic works better than QA with existing hardware after the warm-up stage. The number of iterations of AI5 would increase by 20% if we deploy all iterations to QA. For the real-world QA, we evaluate the end-to-end execution time of HyQSAT, which consists of frontend time on CPU and QA execution time on D-Wave 2000Q, backend time on CPU, and the remaining CDCL time on CPU. The baseline is the MiniSAT and KisSAT time spent on the CPU. Since existing QA-based approaches embed the entire 3-SAT problem to QA [7], [8], which shows limited scalability, e.g., [8] can only handle 80 variables and 110 clauses. Our benchmarks cannot be solved on the QA hardware using existing approaches. We only compare their embedding schemes, including the Minorminer algorithm [11] and the place and route algorithm [8].

B. Iteration Reduction in the Noise-free Simulator

Table I shows the number of iterations using the classic CDCL algorithm (MiniSAT) and HyQSAT. Here, one

TABLE I
THE ITERATION NUMBER OF 14 BENCHMARKS USING CLASSIC CDCL AND HYQSAT.

Domain	Benchmark	#Variable	#Clause	#Problem	CDCL [70] #Iteration	HyQSAT #Iteration	Avg reduction	Geomean reduction	Max reduction	Min reduction
Graph Coloring	GC1: Flat150-360	450	1680	100	434	179	2.75	2.42	9.26	1.46
	GC2: Flat175-417	525	1951	100	666	238	3.22	2.79	9.97	1.41
	GC3: Flat200-479	600	2237	100	1206	414	3.35	2.91	9.92	1.37
Circuit Fault Analysis	CFA	435-1027	1027-34238	4	381	189	83.21	17.28	329.00	0.90
Block Planning	BP	48-6325	261-131973	5	7	1	7.00	6.74	10.00	4.00
Inductive Inference	II	66-1728	186-24792	41	235	113	6.82	3.05	25.00	0.89
Integer Factorization	IF1: EzFact	193-1729	1113-11001	30	13708	7521	33.92	19.25	59.37	1.60
	IF2: Lisa	1201-1453	6563-7967	14	345620	179590	3.06	2.40	10.60	1.34
Cryptography	CRY: Cmpadd	272-4584	780-13236	5	180	6	37.56	37.48	40.00	35.13
Artificial Intelligence	AI1: UF150-645	150	645	100	2724	642	4.13	3.32	13.33	1.53
	AI2: UF175-753	175	753	100	6191	2953	3.65	2.70	30.05	1.18
	AI3: UF200-860	200	860	100	13090	5416	4.38	2.97	46.87	1.21
	AI4: UF225-960	225	960	100	32332	14133	8.89	3.86	60.37	1.23
	AI5: UF250-1065	250	1065	100	103200	50202	6.72	3.10	134.85	1.03
Average							14.11	7.56	53.47	3.81

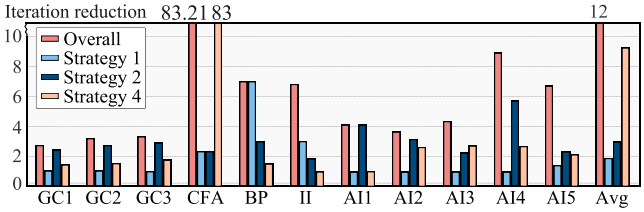


Fig. 10. Reduction ablation of different feedback strategies on the backend.

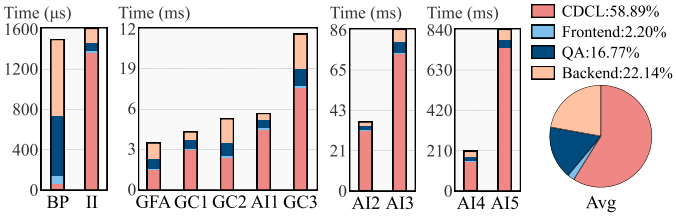


Fig. 11. Time spent in each part of HyQSAT.

iteration includes three steps (*decision, propagation, conflict resolving*) of CDCL. The iteration reduction is defined as $\frac{CDCL\#Iteration}{HyQSAT\#Iteration}$. Overall, HyQSAT outperforms the classic CDCL algorithm in all 14 benchmarks with 14.11X average reduction, 83.21X maximum reduction, and 2.75X minimum reduction. HyQSAT achieves a maximum reduction (329X) in CFA by detecting conflicts in the first several iterations via feedback strategy 4 in the backend, which prunes a large amount of search space at the beginning.

We observe that HyQSAT is particularly suitable for solving difficult problems. For example, AI4 and AI5 exhibit a higher reduction compared to AI1-AI3 as they require more iterations. Artificial intelligence benchmarks have fewer variables and clauses than the graph colouring benchmarks but show higher reductions. It is attributed to the low satisfiability of clauses. In sum, HyQSAT is capable of dealing with large-scale and difficult SAT problems, which is a fundamental improvement compared to existing QA-based approaches [7], [8].

We also quantitatively analyze the reduction ablation by ablating different feedback strategies of the HyQSAT backend (Figure 10). We do not illustrate the reduction of feedback

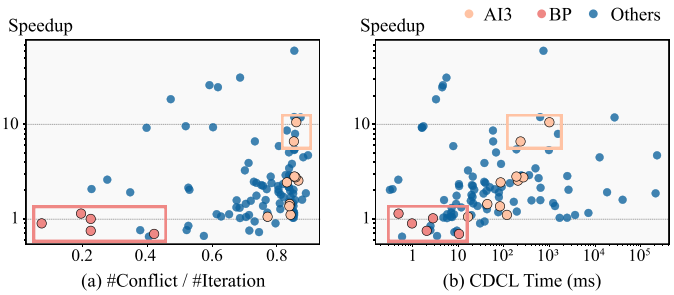


Fig. 12. Evaluating the problem difficulty and the speedup.

strategy 3, as it does not guide the CDCL search. We can observe that all these three strategies contribute to the overall reduction, which suggests that our backend can effectively reduce the CDCL search space. Strategy 1 provides less reduction because this strategy is rarely used because most embedded clauses cannot reach zero energy. The reduction of strategy 4 on the CFA benchmark is close to the overall reduction. This is because CFA is an unsatisfiable benchmark, where strategy 4 is frequently applied to identify some useful conflicts.

C. Speedup on the Real-World QA

In this section, we implement HyQSAT on real-world QA and compare the end-to-end running time with the classic CDCL algorithm on the Intel E5 CPU. Table II lists the execution time of HyQSAT compared to MiniSAT and KisSAT. HyQSAT achieves speedup in 12 of 14 benchmarks compared to MiniSAT and 13 of 14 benchmarks compared to KisSAT, ranging from 1.48X to 12.62X, which outperforms classical CPU with quantum speedup. We observe that HyQSAT does not show computational advantages compared to Minisat on II and BP benchmarks. The clause of these two benchmarks shows low conflict frequency. Thus, they can be easily solved by classic CDCL in a few iterations. In the benchmarks that require more iterations, e.g., IF1 and AI5, the speedups with real-world QA are closer to the noise-free simulator result.

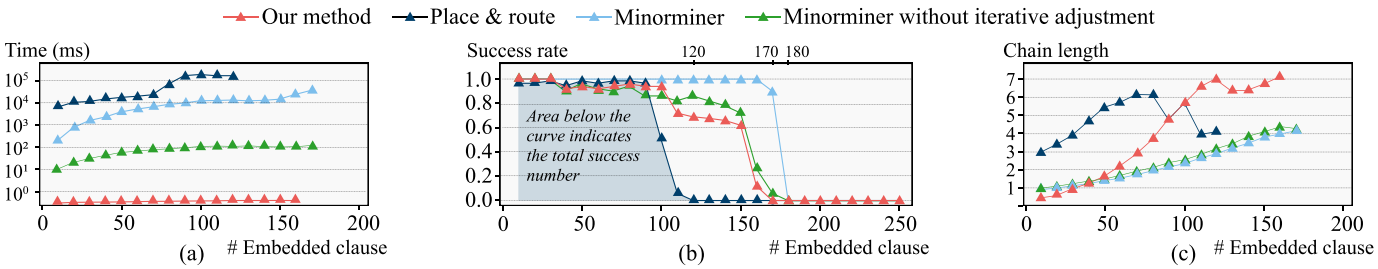


Fig. 13. Evaluating the embedding efficiency in terms of embedding time, success rate, and chain length.

TABLE II
RUNNING TIME COMPARISON BETWEEN CDCL ON INTEL E5 CPU AND HYQSAT ON D-WAVE 2000Q.

	Minisat [70] time (ms)	Kissat [14] time (ms)	HyQSAT time (ms)	Speedup (Minisat)	Speedup (Kissat)	#Iteration variance
GC1	7.08	9.60	5.45	2.19	1.86	0.93
GC2	12.09	17.29	5.48	1.92	2.45	0.93
GC3	27.1	30.95	12.52	2.61	3.31	1.04
CFA	5.03	11.66	3.03	1.76	3.26	5.46
BP	1.40	3.23	1.67	0.81	3.01	0.49
II	1.48	4.18	1.58	0.89	3.53	1.53
IF1	582.92	1249.12	292.13	5.89	12.62	2.41
IF2	34322.7	30137.88	12451.43	1.91	3.14	1.37
CRY	0.22	0.88	0.12	1.48	5.90	1.16
AI1	20.00	39.38	5.97	5.32	8.82	1.01
AI2	72.41	130.99	36.93	2.86	7.88	1.27
AI3	180.72	196.68	82.88	3.72	4.82	0.60
AI4	494.63	318.17	204.05	4.69	4.38	0.98
AI5	1650.92	1307.23	828.8	5.33	3.71	1.42

We also evaluate the noise effect (Table II) which is defined as $\frac{QA\ \#iteration}{Simulator\ \#iteration}$. For most benchmarks, the number of iterations of the real-world QA is similar to the noise-free simulator, suggesting that our method is noise-tolerant. On several benchmarks, HyQSAT requires fewer iterations compared to the noise-free simulator. We hypothesize that it may be the result of randomness introduced by the noise, which possibly leads to a better search direction. This advantage primarily benefits from our hybrid methodology where errors of QA are not likely to affect the overall correct assignment. The frontend of HyQSAT also adjusts the clause coefficients of the objective function, which helps reduce the noise overhead. The number of iterations on CFA increases the most on D-Wave 2000Q because CFA is unsatisfiable. HyQSAT requires more iterations to locate the conflict position under hardware noise.

Figure 11 shows the breakdown of the HyQSAT execution time, including the warm-up stage (frontend, QA part and backend) and the remaining CDCL search. We observe that it takes 41.11% of the time taken in the warm-up stage, where HyQSAT is applied to speed up the solving time. In our frontend, the hardware embedding is pipelined with the clause queue generation to reduce the execution time. Thus, the frontend requires less time (2.2%) on the CPU. The QA execution time on most benchmarks only accounts for a small part, as a single sampling time of QA is about 130μs. BP benchmark shows a high QA time that represents 39.64% of

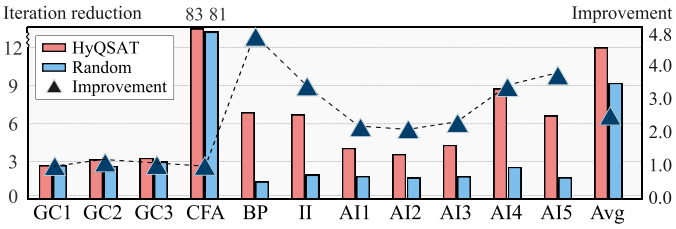


Fig. 14. Iteration reduction comparison between our method and a method that randomly generates clause queue.

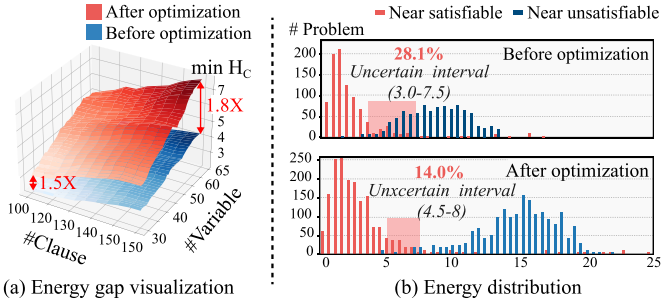
the computational time. This is because this benchmark has fewer overall iterations, leading to more iterations conducted on QA hardware. In the backend, satisfaction probability classification only requires near-constant time. Most backend time is spent on feedback strategies. Note that although CDCL takes nearly half of the overall time, this time is also optimized as we reduce the total number of iterations.

Figure 12 shows the results of an analysis of the relationship between the problem difficulty and speed. In Figure 12 (a), problems with a higher conflict proportion have higher speedup results from the pruned search space by HyQSAT. In contrast, benchmark II has less than 1X speedup due to a relatively small conflict proportion. In Figure 12 (b), problems that require long classic CDCL times tend to have higher speedup. This is because the number of warm-up iterations is large, which can benefit from the acceleration of quantum computing.

D. Embedding Efficiency

This section evaluates the embedding efficiency of HyQSAT by comparing two state-of-the-art embedding schemes, Minorminer scheme [11] and the place and route scheme (P&R) [8]. We choose 50 different clause queues. Each queue has 250 clauses. We test the embedding time, success rate, and chain length with different numbers of embedded clauses (Figure 13 (a-c)). The embedding is considered a failure if it exceeds the timeout of the 300s.

Embedding time. Compared to Minorminer and P&R schemes, we achieve 8.95×10^5X and 2.6×10^6X speedup, respectively. These two schemes show a polynomial complexity with an initial time. The initialization is used to analyze the structures of the problem graph and hardware graph to update their heuristics. However, using HyQSAT, the embedding process is paralleled with the popping operation of the clause queue without investigating the whole graph.



(a) Energy gap visualization (b) Energy distribution

Fig. 15. Evaluation of the noise optimization.

Success rate. We evaluate the embedding ability of different schemes (Figure 13 (b)). The maximum number of embeddable clauses of our method (170) is slightly smaller than the Minorminer scheme (180) and larger than the P&R scheme (120). P&R has a time-consuming heuristic for allocating variables, which makes it easily exceed the limited time when the number of clauses increases. The success rate of the Minorminer scheme is relatively stable until the number of clauses reaches 180, which benefits from iterative adjustment. Without adjustment, Minorminer has a similar success rate compared to ours. We also calculate the number of successfully embedded clauses (the area below the curve). Our method is 82% (667 / 815) of Minorminer for successful clauses, which is acceptable when considering the speedup of embedding time and longer chain length. HyQSAT considers the topology of the 3-SAT problem graph, such as the sparse connection between variables and auxiliary variables. Thus, it does not require complicated cost functions and adjustment operations. As a result, our embedding method is faster than the Minorminer scheme while showing a similar embedding capacity.

Chain length. A shorter chain length produces less hardware noise. When the number of clauses reaches the maximum capacity, the average chain length of our method is about 1.59X longer than the other two schemes. Such limitation arises from the absence of a complicated routing and adjustment to optimize the chain length between qubits. With about 10^6 speedup in embedding time along with the ability of our approach to tolerate noise, we consider that an increased chain length is acceptable for solving 3-SAT problems.

E. Effect of the Clause Queue Generation

This section compares the clause queue generation of the HyQSAT frontend to random generation. The baseline is the number of iterations of the classic CDCL algorithm. Figure 14 illustrates a 2.77X improvement compared to the random method. The improvement results from the application of an activity score to identify the clauses with high conflict frequency. By deploying the hard clauses from the queue head to QA, we reduce the number of iterations by transferring a large number of conflicts to high-parallelism QA. We observe that the iteration reduction improves more in the last 7 benchmarks. These benchmarks have more complex clauses that are hard to solve by CDCL and benefit more from our algorithm.

TABLE III
EVALUATION OF HYQSAT SCALABILITY.

Benchmark	16×16 grid	24×24 grid	32×32 grid	64×64 grid
AI1	4.09	341.51	344.79	348.71
AI2	3.32	1035.21	1034.00	1037.66
AI3	4.04	2357.17	2321.82	2208.66
AI4	5.70	5183.60	4971.09	5074.64
AI5	6.17	1.51×10^4	1.52×10^4	1.52×10^4
Var500	5.67	7.97	2.20×10^5	2.31×10^6

F. Evaluation on Noise Optimization.

As mentioned earlier, we optimize the coefficients in the objective function to increase the QA energy gap so that QA can have a higher probability of converging to the global minimum. Figure 15 (a) draws the energy surface before and after optimization under different numbers of clauses and variables. Overall, our optimization helps to increase up to 1.8X energy gap. We also observe a larger increase for problems with more variables and clauses. For problems with 65 variables and 145 clauses, the energy gap improvement is 1.8X on average. In problems with 30 variables, the improvement is about 1.5X. When the optimized objective function is applied to QA, a higher energy gap can separate the near-satisfiable interval from the near-unsatisfiable interval, which helps the backend make a more precise decision. After noise optimization, the near-unsatisfiable interval is pulled out, where there is less overlapping with the near-satisfiable interval, reducing the uncertainty interval from 28.1% to 14.0% (Figure 15 (b)). The mean accuracy of the Gaussian Naive Bayes model trained exhibits a 12.77% improvement (84.76% → 97.53%).

G. Scalability

We simulate the speedup of HyQSAT on Chimera graph topologies with different sizes (Table III) compared to MiniSAT, where a 10% bit flipping error is added to noise-free simulation results. The results show that our approach can scale to larger hardware grids. For AI problems, the 32×32 grid QA topology can nearly embed all clauses, solving the problem in a few iterations, which leads to more than 348X iteration reduction. We simulate 10 problems with 500 variables. Most of its clauses can be embedded by the 64×64 grid QA topology with a significant speedup.

VII. DISCUSSION

A. Switching Latency

In practice, the switching latency between classical computers and QA consists of: *a)* communication time between classical computers and peripheral devices; *b)* pre-processing time for peripheral devices to generate pulses to control the quantum system; *c)* post-processing time to interpret pulses from QAs. The communication time can be eliminated by implementing the CDCL part on FPGAs of peripheral devices, which has a similar computational latency per iteration as the CPU-based CDCL (16ns per iteration on average [31]). The pre-processing time between FPGA and QA only takes 160ns using the customized FPGAs [17], [28]. In addition, the real-time feedback techniques [10], [18] can reduce the

post-processing time to 500ns. Based on these techniques, the switching latency is within microseconds, which can be covered by the QA execution time ($130\mu s$).

B. From 3-SAT to K-SAT Problem

3-SAT problem is a representative branch of SAT problems and has been widely studied recently. Theoretically, any K-SAT problem can be encoded into a minimization problem and embedded into QA hardware. However, the encoding of K-SAT can introduce more auxiliary variables, resulting in hardware inefficiency. For example, embedding a clause with 26 variables requires introducing 24 auxiliary variables. Such limitation comes from the non-all-to-all hardware topology of qubits connection since it is hard to physically implement a fully-entangled quantum chip. We consider two potential approaches to extend HyQSAT to K-SAT problems. One might assign variables in the decision step to transform clauses to 3-SAT clauses in the frontend. This method may sacrifice some potential speedup as it involves more iterations in CDCL. Another approach might design an application-specific QA architecture similar to [39], [52], where the connection topology is specified for K-SAT problems within a specific domain.

VIII. Related work

A. SAT Algorithm

CDCL has been widely used to solve SAT problems due to its high speedup [12], [25], [41], [49]. The look-ahead solver explores one level of the tree search to detect potential conflicts [32]. We can explore a higher number of levels and get more information thanks to quantum speedup. VSIDS heuristic helps to find variables with high conflict frequencies that need to be assigned at first [49]. We extend this method to identify hard clauses when generating a clause queue. To improve search efficiency, recent CDCL solvers tend to replace heuristic with local search [12] and reinforcement learning [38]. However, these methods show poor scalability. Their feedback strategies are hard to be applied to QA hardware because of the noise overhead.

Complementary quantum SAT approaches are based on Grover algorithm [13], quantum approximate optimization [2], or quantum random walks [35]. They fail to handle large-scale problems with real-world gate-based quantum computers [64], [72], [74]. Existing QA approaches are also inefficient as they embed all clauses to hardware, leading to long frontend time and high noise overhead [7], [8]. Hybrid quantum-classical algorithms were introduced to overcome the challenges of existing quantum computing [44], [72]. Some algorithms were designed to solve other problems [29], [54], [59], [60], e.g., variational quantum algorithms [74]. To the best of our knowledge, our method is the first hybrid quantum-classical algorithm that provides an end-to-end quantum speedup for SAT problems.

B. Embedding

The properties of hardware embedding are derived from graph theory [66]. [8] propose an algorithm to learn from a conventional circuit mapping algorithm where sub-functions are placed into a hardware graph and connected by routing [27], [34]. [71] designed an approach aiming at finding small chain lengths using a fast cost function for placement. Recent works have tried to reduce embedding time by exploiting QA topology [9], [11], [37]. For example, some works leverage tree-width of the problem graph [9], [36], or node connectivity as heuristics to optimize the allocation [11]. [37], [56] also achieve linear time complexity for densely-connected problem graphs. However, the problem graph of SAT problem shows a sparse connection due to auxiliary variables.

C. Noise Optimization

For the noise introduced before quantum annealing, there are works that studied coupler strength and annealing time via empirical method [15], [65] and mathematical method [3], [50], [73]. Introducing auxiliary qubits [4], [21], [76], [77] and energy penalty [62] can increase the energy gap and reduce noise, however, it decreases the embedding scale of instances. [73] suggest an approach that iteratively increases the coupler strength to find suitable configuration parameters. For the noise introduced after quantum annealing, there are several calibration methods to reduce the effect of the noise [58], such as majority voting [62], [63] and greedy descent [6]. All these methods ignore the noise introduced in the normalization step of QA. By adjusting the coefficient of sub-clauses, our method can increase the energy gap while maintaining the embedding scale.

IX. Conclusion

We propose a hybrid approach for solving 3-SAT problems, called HyQSAT, which takes advantage of both quantum parallelism and classical algorithms. Depending on the conflict frequency of clauses, our frontend generates a clause queue and embeds the head clauses into hardware. In the backend, we propose a methodology to leverage QA results to guide the CDCL search. We employ some optimization techniques to reduce noise overhead, including coefficient adjustment and confidence interval partition. Our experiments demonstrate that HyQSAT can achieve real quantum speedup (up to 12.62X) on D-Wave 2000Q, compared to the state-of-the-art classical methods.

X. acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant (No. 61825205). This work was also funded by Zhejiang Pioneer (Jianbing) Project (No. 2022C01G2013292). AP acknowledges support from the National Key Research and Development Program of China (No. 2021YFC2701905).

REFERENCES

[1] A. V. Aho and J. E. Hopcroft, *The design and analysis of computer algorithms*. Pearson Education India, 1974.

[2] V. Akshay, H. Philathong, M. E. Morales, and J. D. Biamonte, "Reachability deficits in quantum approximate optimization," *Physical review letters*, vol. 124, no. 9, 2020.

[3] T. Albash, S. Boixo, D. A. Lidar, and P. Zanardi, "Quantum adiabatic markovian master equations," *New Journal of Physics*, vol. 14, no. 12, 2012.

[4] P. Aliferis, D. Gottesman, and J. Preskill, "Quantum accuracy threshold for concatenated distance-3 codes," *Quantum Info. Comput*, vol. 6, no. 2, 2006.

[5] M. H. Amin, "Searching for quantum speedup in quasistatic quantum annealers," *Physical Review A*, vol. 92, no. 5, 2015.

[6] R. Ayanzadeh, J. Dorband, M. Halem, and T. Finin, "Multi-qubit correction for quantum annealers," *Scientific Reports*, vol. 11, no. 1, 2021.

[7] D. A. Battaglia, G. E. Santoro, and E. Tosatti, "Optimization by quantum annealing: Lessons from hard satisfiability problems," *Physical Review E*, vol. 71, no. 6, 2005.

[8] Z. Bian, F. Chudak, W. Macready, A. Roy, R. Sebastiani, and S. Varotti, "Solving sat and maxsat with a quantum annealer: Foundations and a preliminary report," in *International Symposium on Frontiers of Combining Systems*. Springer, 2017.

[9] T. Boothby, A. D. King, and A. Roy, "Fast clique minor generation in chimera qubit connectivity graphs," *Quantum Information Processing*, vol. 15, no. 1, 2016.

[10] C. C. Bultink, M. Rol, T. O'Brien, X. Fu, B. Dikken, C. Dickel, R. Vermeulen, J. De Sterke, A. Bruno, R. Schouten *et al.*, "Active resonator reset in the nonlinear dispersive regime of circuit qed," *Physical Review Applied*, vol. 6, no. 3, p. 034008, 2016.

[11] J. Cai, W. G. Macready, and A. Roy, "A practical heuristic for finding graph minors," *arXiv preprint arXiv:1406.2741*, 2014.

[12] S. Cai and X. Zhang, "Deep cooperation of cdcl and local search for sat," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2021.

[13] N. J. Cerf, L. K. Grover, and C. P. Williams, "Nested quantum search and structured problems," *Physical Review A*, vol. 61, no. 3, 2000.

[14] M. S. Cherif, D. Habet, and C. Terrioux, "Kissat mab: Combining vsids and chb through multi-armed bandit," *SAT COMPETITION 2021*, p. 15, 2021.

[15] V. Choi, "Minor-embedding in adiabatic quantum computation: Ii. minor-universal graph design," *Quantum Information Processing*, vol. 10, no. 3, 2011.

[16] M. S. Chowdhury, J. You *et al.*, "Guiding cdcl sat search via random exploration amid conflict depression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 02, 2020.

[17] A. D. Córcoles, M. Takita, K. Inoue, S. Lekuch, Z. K. Minev, J. M. Chow, and J. M. Gambetta, "Exploiting dynamic quantum circuits in a quantum algorithm with superconducting qubits," *Physical Review Letters*, vol. 127, no. 10, p. 100501, 2021.

[18] J. Cramer, N. Kalb, M. A. Rol, B. Hensen, M. S. Blok, M. Markham, D. J. Twitchen, R. Hanson, and T. H. Taminiau, "Repeated quantum error correction on a continuously encoded qubit by real-time feedback," *Nature communications*, vol. 7, no. 1, pp. 1–7, 2016.

[19] D-wave, "An implementation of a simulated annealing sampler." https://github.com/dwavesystems/dwave-neal, May 2018.

[20] E. Dantsin, E. A. Hirsch, and A. Wolpert, "Algorithms for sat based on search in hamming balls," in *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, 2004.

[21] P. Das, C. A. Pattison, S. Manne, D. M. Carmean, K. M. Svore, M. Qureshi, and N. Delfosse, "Afs: Accurate, fast, and scalable error-decoding for fault-tolerant quantum computers," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 259–273.

[22] P. Das, S. Tannu, and M. Qureshi, "Jigsaw: Boosting fidelity of nisq programs via measurement subsetting," in *In Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

[23] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[24] N. G. Dickson, M. Johnson, M. Amin, R. Harris, F. Altomare, A. Berkley, P. Bunyk, J. Cai, E. Chapple, P. Chavez *et al.*, "Thermally assisted quantum annealing of a 16-qubit problem," *Nature communications*, vol. 4, no. 1, 2013.

[25] J. Elffers, J. Giráldez-Cru, S. Gocht, J. Nordström, and L. Simon, "Seeking practical cdcl insights from theoretical sat benchmarks." in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[26] A. B. K. F. M. Fleury and M. Heisinger, "Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020," *SAT COMPETITION 2020*, 2020.

[27] B. Fu and J. Kim, "Footprint: Regulating routing adaptiveness in networks-on-chip," in *In 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[28] X. Fu, M. A. Rol, C. C. Bultink, J. Van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, J. De Sterke, W. Vlothuizen, R. Schouten *et al.*, "An experimental microarchitecture for a superconducting quantum processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 813–825.

[29] J. Gao, "Hybrid quantum and molecular mechanical simulations: an alternative avenue to solvent effects in organic chemistry," *Accounts of chemical research*, vol. 29, no. 6, 1996.

[30] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[31] K. Gulati, S. Paul, S. P. Khatri, S. Patil, and A. Jas, "Fpga-based hardware acceleration for boolean satisfiability," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, no. 2, pp. 1–11, 2009.

[32] M. Heule, M. Dufour, J. Van Zwieten, and H. Van Maaren, "March_eq: Implementing additional reasoning into an efficient look-ahead sat solver," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2004.

[33] H. H. Hoos and T. Stützle, "Satlib: An online resource for research on sat," *Sat*, vol. 2000, 2000.

[34] H. Kasan, G. Kim, Y. Yi, and J. Kim, "Dynamic global adaptive routing in high-radix networks," in *In 2022 ACM/IEEE 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022.

[35] J. Kempe, "Quantum random walks: an introductory overview," *Contemporary Physics*, vol. 44, no. 4, 2003.

[36] T. Kloks, *Treewidth: computations and approximations*. Springer, 1994.

[37] C. Klymko, B. D. Sullivan, and T. S. Humble, "Adiabatic quantum programming: minor embedding with hard faults," *Quantum information processing*, vol. 13, no. 3, 2014.

[38] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver?" *Advances in Neural Information Processing Systems*, vol. 33, pp. 9608–9621, 2020.

[39] G. Li, Y. Shi, and A. Javadi-Abhari, "Software-hardware co-optimization for computational chemistry on superconducting quantum processors," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 832–845.

[40] J. H. Liang, H. G. VK, P. Poupart, K. Czarnecki, and V. Ganesh, "An empirical study of branching heuristics through the lens of global learning rate," in *International conference on theory and applications of satisfiability testing*. Springer, 2017, pp. 119–135.

[41] M. Luo, C.-M. Li, F. Xiao, F. Manya, and Z. Lü, "An effective learnt clause minimization approach for cdcl sat solvers," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[42] R. Marino, G. Parisi, and F. Ricci-Tersenghi, "The backtracking survey propagation algorithm for solving random k-sat problems," *Nature communications*, vol. 7, no. 1, 2016.

[43] J. Marques-Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning sat solvers," in *Handbook of satisfiability*. ios Press, 2021.

[44] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, 2016.

[45] B. M. McCoy and T. T. Wu, *The two-dimensional Ising model*. Harvard University Press, 2013.

[46] C. McGeoch and P. Farré, "The d-wave advantage system: An overview," *D-Wave Systems Inc., Burnaby, BC, Canada, Tech. Rep*, 2020.

[47] C. C. McGeoch, R. Harris, S. P. Reinhardt, and P. I. Bunyk, "Practical annealing-based quantum computing," *Computer*, vol. 52, no. 6, 2019.

[48] B. Molnár, F. Molnár, M. Varga, Z. Toroczkai, and M. Ercsey-Ravasz, "A continuous-time maxsat solver with high analog performance," *Nature communications*, vol. 9, no. 1, 2018.

[49] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th annual Design Automation Conference (DAC)*, 2001.

[50] A. Mott, J. Job, J.-R. Vlimant, D. Lidar, and M. Spiropulu, "Solving a higgs optimization problem with quantum annealing for machine learning," *Nature*, vol. 550, no. 7676, 2017.

[51] S. Mukherjee and B. K. Chakrabarti, "Multivariable optimization: Quantum annealing and computation," *The European Physical Journal Special Topics*, vol. 224, no. 1, 2015.

[52] P. Murali, D. M. Debroy, K. R. Brown, and M. Martonosi, "Architecting noisy intermediate-scale trapped ion quantum computers," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 529–542.

[53] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[54] G. Nannicini, "Performance of hybrid quantum-classical variational heuristics for combinatorial optimization," *Physical Review E*, vol. 99, no. 1, 2019.

[55] S. Nerli, V. S. De Paula, A. C. McShan, and N. G. Sgourakis, "Backbone-independent nmr resonance assignments of methyl probes in large proteins," *Nature communications*, vol. 12, no. 1, 2021.

[56] S. Okada, M. Ohzeki, M. Terabe, and S. Taguchi, "Improving solutions by embedding larger subproblems in a d-wave quantum annealer," *Scientific reports*, vol. 9, no. 1, 2019.

[57] M. Osama, A. Wijs, and A. Biere, "Sat solving with gpu accelerated inprocessing," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2021, pp. 133–151.

[58] T. Patel and D. Tiwari, "Qraft: reverse your quantum circuit and know the correct program output," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 443–455.

[59] E. Pelofske, G. Hahn, and H. N. Djidjev, "Solving larger optimization problems using parallel quantum annealing," *arXiv preprint arXiv:2205.12165*, 2022.

[60] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, no. 1, 2014.

[61] N. Prevot, M. Soos, and K. S. Meel, "Leveraging gpus for effective clause sharing in parallel sat solving," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2021, pp. 471–487.

[62] K. L. Pudenz, T. Albash, and D. A. Lidar, "Error-corrected quantum annealing with hundreds of qubits," *Nature communications*, vol. 5, no. 1, 2014.

[63] K. L. Pudenz, T. Albash, and D. A. Lidar, "Quantum annealing correction for random ising problems," *Physical Review A*, vol. 91, no. 4, 2015.

[64] G. S. Ravi, K. N. Smith, P. Gokhale, A. Mari, N. Earnest, A. Javadi-Abhari, and F. T. Chong, "Vaqem: A variational approach to quantum error mitigation," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 288–303.

[65] J. Raymond, N. Ndiaye, G. Rayaprolu, and A. D. King, "Improving performance of logical qubits by parameter tuning and topology compensation," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2020.

[66] N. Robertson and P. D. Seymour, "Graph minors. xiii. the disjoint paths problem," *Journal of combinatorial theory, Series B*, vol. 63, no. 1, 1995.

[67] L. Simon, D. Le Berre, and E. A. Hirsch, "The sat2002 competition (preliminary draft)," 2002.

[68] M. Soos, "The cryptominisat 5 set of solvers at sat competition 2016," *Proceedings of SAT Competition*, 2016.

[69] M. Soos, K. Nohl, and C. Castelluccia, "Extending sat solvers to cryptographic problems," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2009.

[70] N. Sörensson, "Minisat 2.2 and minisat++ 1.1," *A short description in SAT Race*, vol. 2010, 2010.

[71] J. Su and L. He, "Fast embedding of constrained satisfaction problem to quantum annealer with minimizing chain length," in *Proceedings of the 54th annual Design Automation Conference (DAC)*. IEEE, 2017.

[72] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi, "Cutqc: using small quantum computers for large quantum circuit evaluations," in *Proceedings of the 26th ACM International conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[73] D. Venturelli, S. Mandrà, S. Knysh, B. O'Gorman, R. Biswas, and V. Smelyanskiy, "Quantum optimization of fully connected spin glasses," *Physical Review X*, vol. 5, no. 3, 2015.

[74] H. Wang, Y. Ding, J. Gu, Y. Lin, D. Z. Pan, F. T. Chong, and S. Han, "Quantumnas: Noise-adaptive search for robust quantum circuits," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 692–708.

[75] L. Xie, J. Zhai, Z. Zhang, J. Allcock, S. Zhang, and Y.-C. Zheng, "Suppressing zz crosstalk of quantum computers through pulse and scheduling co-optimization," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022.

[76] K. C. Young, R. Blume-Kohout, and D. A. Lidar, "Adiabatic quantum optimization with the wrong hamiltonian," *Physical Review A*, vol. 88, no. 6, 2013.

[77] K. C. Young, M. Sarovar, and R. Blume-Kohout, "Error suppression and error correction in adiabatic quantum computation: Techniques and challenges," *Physical Review X*, vol. 3, no. 4, 2013.

[78] B. Zhang, A. Sone, and Q. Zhuang, "Quantum computational phase transition in combinatorial problems," *npj Quantum Information*, vol. 8, no. 1, pp. 1–11, 2022.

[79] W. Zhang, Z. Sun, Q. Zhu, G. Li, S. Cai, Y. Xiong, and L. Zhang, "Nlocalsat: Boosting local search with solution prediction," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI) 2020*, C. Bessiere, Ed. ijcai.org, 2020, pp. 1177–1183. [Online]. Available: https://doi.org/10.24963/ijcai.2020/164

[80] P. Zhong, M. Martonosi, and P. Ashar, "Fpga-based sat solver architecture with near-zero synthesis and layout overhead," *IEE Proceedings-Computers and Digital Techniques*, vol. 147, no. 3, pp. 135–141, 2000.

[81] H. Zhou, R. Jiang, and S. Kong, "Cycsat: Sat-based attack on cyclic logic encryptions," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017.