International Journal of Innovative
Computing, Information and Control
Volume **19**, Number **1**, February **2023**

ICIC International ©2023 ISSN 1349-4198
pp. **61–72**

PROGRAMMING QUANTUM BACKTRACKING FOR CIRCUIT SATISFIABILITY PROBLEM WITH DIVIDE AND CONQUER APPROACH

Vionie Laorensa and Arya Wicaksana*

Department of Informatics
Universitas Multimedia Nusantara
Jl. Scientia Boulevard, Gading Serpong, Tangerang 15810, Indonesia
vionie.laorensa@student.umn.ac.id; *Corresponding author: arya.wicaksana@umn.ac.id

Received June 2022; revised September 2022

Abstract. *The circuit satisfiability problem (CIRCUIT-SAT) is part of the satisfiability problem (SAT) that is very difficult to be solved efficiently (NP-complete problem). The quantum backtracking algorithm developed for solving constraint satisfaction problems (CSPs) could be used for CIRCUIT-SAT to gain quantum speedups and provide valuable insights. In this study, we show the programming of the quantum backtracking algorithm, including the implementation and testing of it using a reference circuit of five input pins, 38 logical gates, and five output pins. Rigetti forest software development kit (SDK) and quantum virtual machine (QVM) are used and run on a classical computer with 32GB of memory. A workaround is required by performing the divide and conquer approach on the problem due to the limited number of qubits available for simulation. The quantum program successfully solves the CIRCUIT-SAT problem with the shortest execution time of 381 ms for five qubits simulation, and the longest is 21 minutes 17 seconds 515 ms for 24 qubits simulation. The simulation accuracy is 100%, yielding an F-score of 1 for all test cases. The results are obtained with the same collapse probability for all input states.*

Keywords: CIRCUIT-SAT, Divide and conquer, Montanaro's algorithm, Quantum backtracking, Quantum programming, Quantum walk, Rigetti forest

1. **Introduction.** The satisfiability problem (SAT) is a computational problem to produce specific outputs under certain constraints [1]. SAT problem, i.e., automated pattern generation and the circuit satisfiability problem (CIRCUIT-SAT), is challenging to solve efficiently using existing approaches. Quantum computing unveils a new computation era with broad applications and potential [2-4], including solving problems in classical computer science more efficiently using quantum algorithms [5-9]. Quantum algorithms provide quantum speedup over classical algorithms on problems such as CIRCUIT-SAT.

The optimized searching algorithms for the problem, such as backtracking, branch and bound, and pruning are widely used to solve the problem [10,11]. The quantum backtracking algorithm developed by Montanaro [12] offers quantum speedups over the classical algorithm. Martiel and Remaud [13] came up with the memory-efficient implementation of the quantum backtracking algorithm for solving the constraint satisfaction problem (CSP). Thus, this study expands the works previously done by Montanaro, Martiel, and Remaud by implementing the quantum backtracking algorithm for practical application, i.e., solving the CIRCUIT-SAT problem.

This study's main contribution is applying the quantum backtracking algorithm (Montanaro's algorithm) [12,13] for the satisfiability problem: CIRCUIT-SAT. Montanaro does

DOI: 10.24507/ijicic.19.01.61

the theoretical work on the algorithm, and the practical approach to Montanaro's algorithm is made by Martiel and Remaud, both for CSP. This study demonstrates the application of the algorithm for satisfiability problems by programming and implementing it using Python and pyQuil code [14]. Another contribution is the use of a state-of-the-art quantum simulator from Rigetti and the workaround required to reduce the use of the number of qubits using the divide and conquer approach. The reference circuit for the CIRCUIT-SAT problem is designed in classical gates and recreated with quantum gates for the quantum program to solve. Test and evaluation aim to validate the application's functionality and correctness and find the proposed solution's performance in terms of accuracy and F-score.

The rest of this paper is organized as follows. Section 2 briefly describes the related work related to the algorithm, and Section 3 proposes research methods. Section 4 presents the experimental results, including the analysis. Finally, Section 5 concludes this paper with some discussions on future work.

2. **Preliminaries.**

2.1. **Quantum backtracking algorithm.**

The quantum backtracking algorithm is an algorithm in quantum computing used as a search algorithm for tree-based problems. Tree problems could be CSP (constraint satisfaction problem) and SAT (satisfiability problem), with m constraints over n variables. The quantum backtracking algorithm or Montanaro's algorithm is based on quantum walk on trees. The quantum walk operates on the space spanned by $\{|x\rangle; x \in \{r\} \cup [1, T-1]\}$, and starts in the state $|r\rangle$ in a rooted tree with T vertices: $r, 1, \ldots, T-1$, and the vertex r is the root. It is based on a set of diffusion operators D_x, where D_x is the identity if x is a solution, otherwise, diffuses on the subspace spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$, which mean that y is a child of x in the tree. A quantum walk step involves applying the operator $R_B R_A$ as ruled out in Equation (1) [12,13].

$$R_A = \oplus_{x \in A} D_x \text{ and } R_B = |r\rangle\langle r| + \oplus_{x \in B} D_x \tag{1}$$

A solution in a tree is detected by using the operator $R_B R_A$. The phase estimation of the operator $R_B R_A$ allows the quantum walker to go through the paths leading to a solution in the tree. Let A be the set of vertices at an even distance from the root (including the root itself), and let B be the set of vertices at an odd distance from the root. The algorithm to detect and find a solution in a tree is shown in Figures 1 and 2 [12,13].

Require: Operators R_A, R_B, a failure probability δ, upper bounds on the depth n and the number of vertices T. Let $\beta, \gamma > 0$ be universal constants to be determined.
1: **Repeat** $K = \lceil \gamma \log 1/\delta \rceil$ times:
2: Apply phase estimation to the operator $R_B R_A$ with precision $\beta/\sqrt{Tn}$.
3: **If** the eigenvalue is 1 **then** accept
4: **If** number of acceptances $\geq 3K/8$ **then return** "Solution exists"
5: **else return** "No solution"

FIGURE 1. Detecting a solution (Algorithm A) [12,13]

The steps described in Figures 1 and 2 require quantum computing. The quantum backtracking algorithm recursively goes through all possibilities to find the solution. A solution is detected by running quantum phase estimation to operator $R_B R_A$, and the solutions are searched by looking out through all outputs from all branches. R_A looks for

1: Apply algorithm A to the entire tree.
2: **If** it outputs "No solution" **then return**
3: Apply algorithm A to each child of the root until one outputs "Solution exists".
4: **If** this child is a solution **then** output its label and **return**
5: **else** go back to step 3 with this child as the root.

FIGURE 2. Finding a solution [12,13]

Require: A basis state $|\ell\rangle\,|v_1\rangle \ldots |v_n\rangle \in \mathbb{C}^{n+1} \otimes (\mathbb{C}^{d+1})^{\otimes n}$ corresponding to a partial assignment $x_1 = v_1, \ldots, x_\ell = v_\ell$. Ancilla registers : $\mathcal{H}_{\text{anc}}, \mathcal{H}_{\text{children}}$, storing a tuple (a, S), where $a \in \{*\} \cup [d]$, $S \subseteq [d]$, initialized to $a = *$, $S = \emptyset$.
1: If ℓ is odd, swap a with v_ℓ.
2: Compute $P(x)$.
3: If $P(x)$ is true, go to step 8.
4: If $a \neq *$, subtract 1 from ℓ.
5: For each $w \in [d]$, if $P(v_1, \ldots, v_\ell, w)$ is not false, set $S = S \cup \{w\}$.
6: If $\ell = 0$, $i = n$, else, $i = 1$. Perform $I - 2\,|\phi_{i,S}\rangle\langle\phi_{i,S}|$ on $\mathcal{H}_{\text{anc}}$.
7: Revert steps 5 and 4.
8: Revert steps 2 and 1.
R_B is similar, except that: step 1 is preceded by the check "If $\ell = 0$, return"; "odd" is replaced with "even" in step 1; and the check "If $\ell = 0$ is removed" from step 6.

FIGURE 3. Implementation of R_A operator (Algorithm 4) [13]

the D_x operator's diffusion result on all x on even branches of the tree (right side). R_B performs the projection from the root and adds the D_x operator's diffusion result to all x's on the odd branch of the tree (left side). The modified algorithm for R_A and R_B as in [13] is shown in Figure 3.

In step 1, ℓ is the level for the vertex in a rooted tree. Compute $P(x)$ determines the output of input x on predicate P. If the result of $P(x)$ is "true", it means that x is a solution, "indeterminate" if it is valid and "false" otherwise. In step 5, "false" and "indeterminate" are checked. If it is indeterminate, then x is the solution. The variable S is a collection of state vectors: the solution of $P(x)$. In step 6, the measurement is applied to the quantum circuit, which is created to produce the final output of the quantum circuit [13].

The R_B operator does a similar thing to R_A, only having a few different values. In step 1, odds are changed to an event because R_A is for the right branch (even), and R_B is for the left branch (odd). In step 6, "if $\ell = 0$" is omitted because the odd branch does not have a root value [13].

2.2. **Circuit satisfiability problem.** The circuit used in this research is designed for proof-of-concept and simulation purposes. The circuit drawn in Figure 4 consists of 38 classical logical gates with five input and output pins. Due to the limitation of quantum simulator specification available at the time of this study. It is necessary to break down the classical circuit into subcircuits to reduce the number of qubits required for the quantum simulation. The divide and conquer approach breaks down the circuit into subcircuits. The objective is to find all input combinations that yield the output value of TRUE.

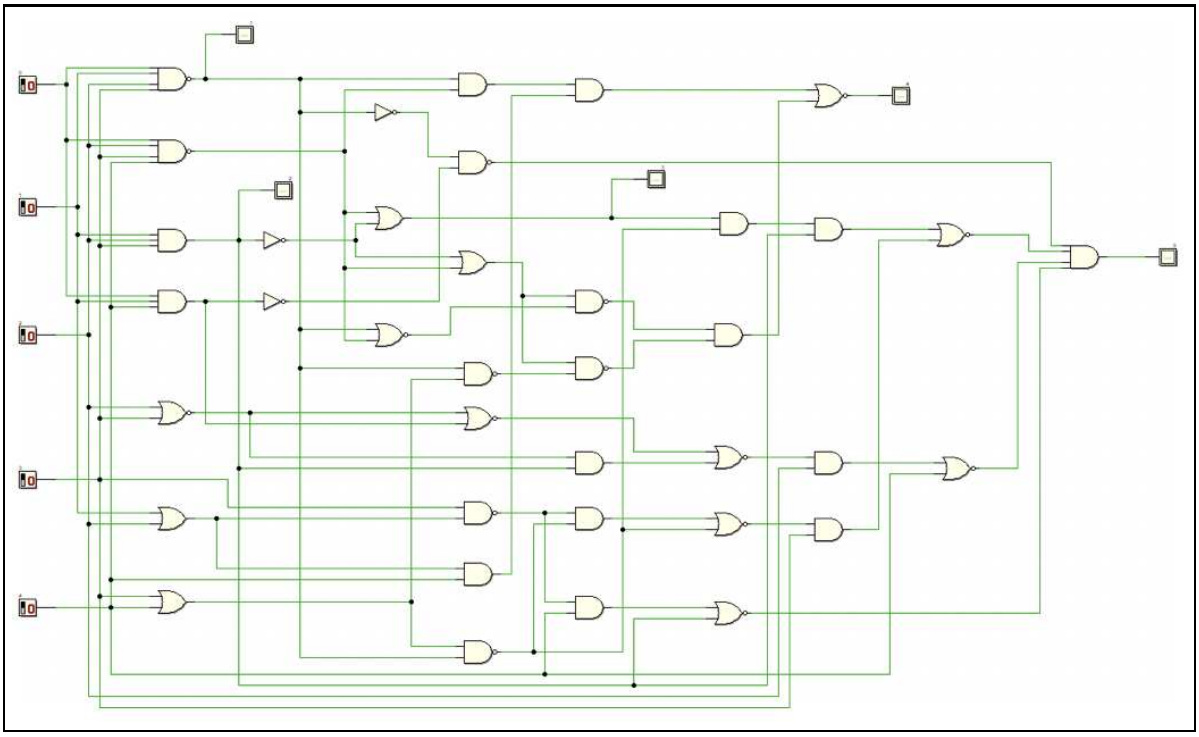

FIGURE 4. The reference circuit

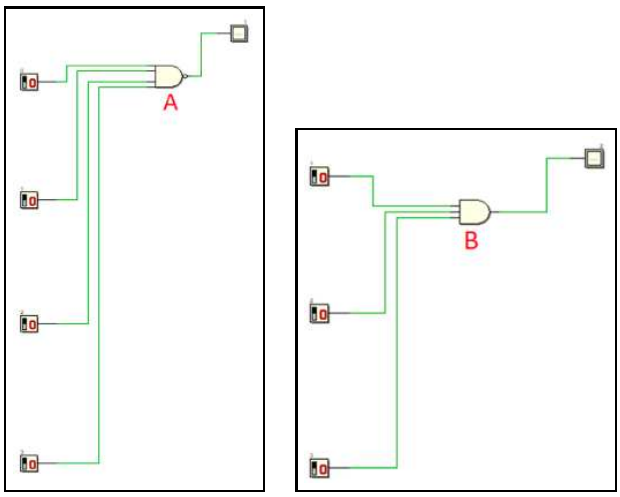

FIGURE 5. Subcircuit 1 (left) and subcircuit 2 (right)

3. **Methods.** The application of the quantum backtracking algorithm for the CIRCUIT-SAT is not straightforward. The preprocessing stage is required to prepare the problem for quantum computation to solve. In this stage, the first step is to divide the circuit into subcircuits based on each output pin. This step is required due to the limitations of the quantum bits available for the simulator to simulate. The reference circuit in Figure 4 is broken down into five subcircuits, as shown in Figures 5-8.

In this step, each gate in each subcircuit is labeled for the recreation of the classical gate in the quantum gate. The second step in the preprocessing stage is the recreation of the problem in quantum gates. If the recreated quantum subcircuit requires more quantum bits for the simulation than the quantum simulator can provide, the preprocessing stage repeats itself in step 1. The corresponding subcircuit is divided even more to produce

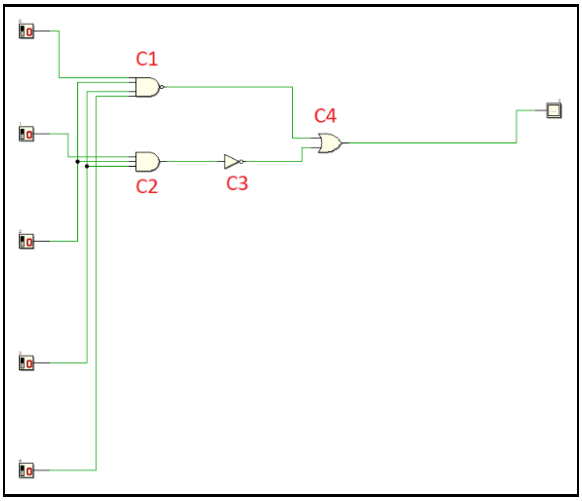

FIGURE 6. Subcircuit 3

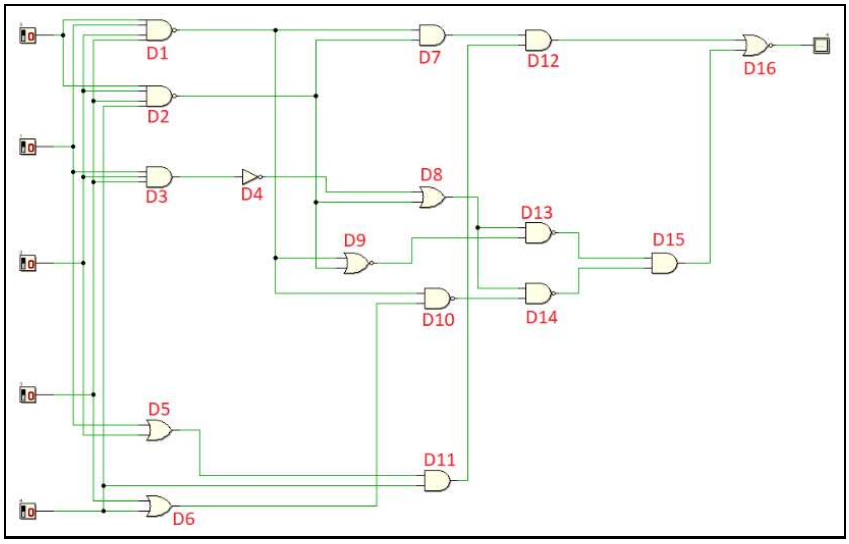

FIGURE 7. Subcircuit 4

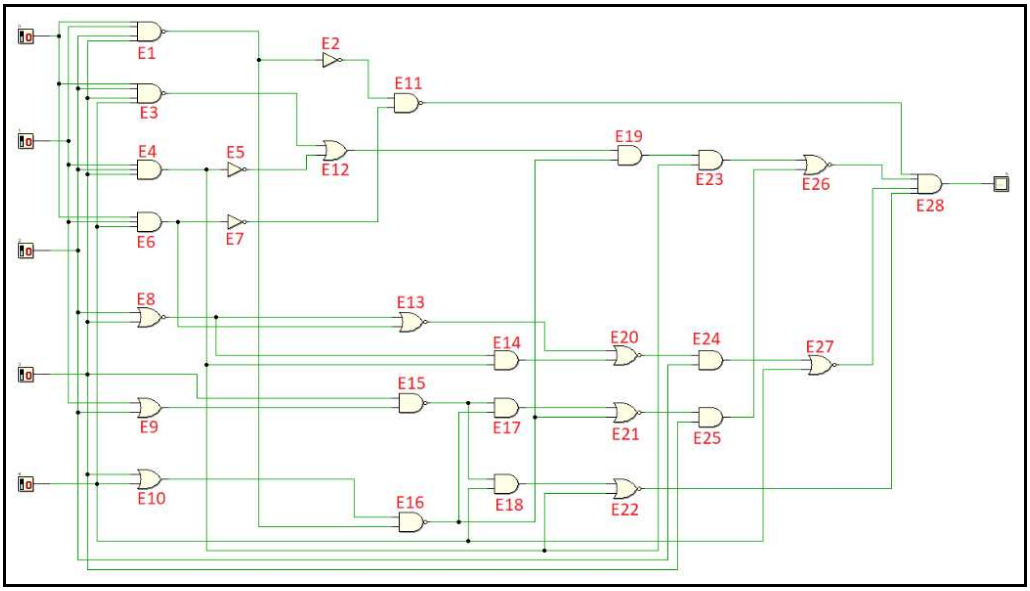

FIGURE 8. Subcircuit 5

smaller sub subcircuits. The process continues to step 2 and proceeds with the quantum simulation of Montanaro's algorithm (quantum backtracking algorithm) for the problem.

This study uses up to 29 qubits for the simulation. Thus, 13 subcircuits are recreated for the quantum simulation purpose in this work. The circuit shown in Figure 5 has to be reconstructed in the quantum system for the quantum backtracking program to solve the problem. This is achieved in this work by using predefined quantum gates. NOT logic is reconstructed using the Pauli-X quantum gate, and AND logic is reconstructed using the Toffoli quantum gate. The OR logic is reconstructed using the CNOT quantum gates and the Toffoli quantum gate. After the preprocessing stage is completed, the method for programming Montanaro's algorithm is summarized in Figure 9.

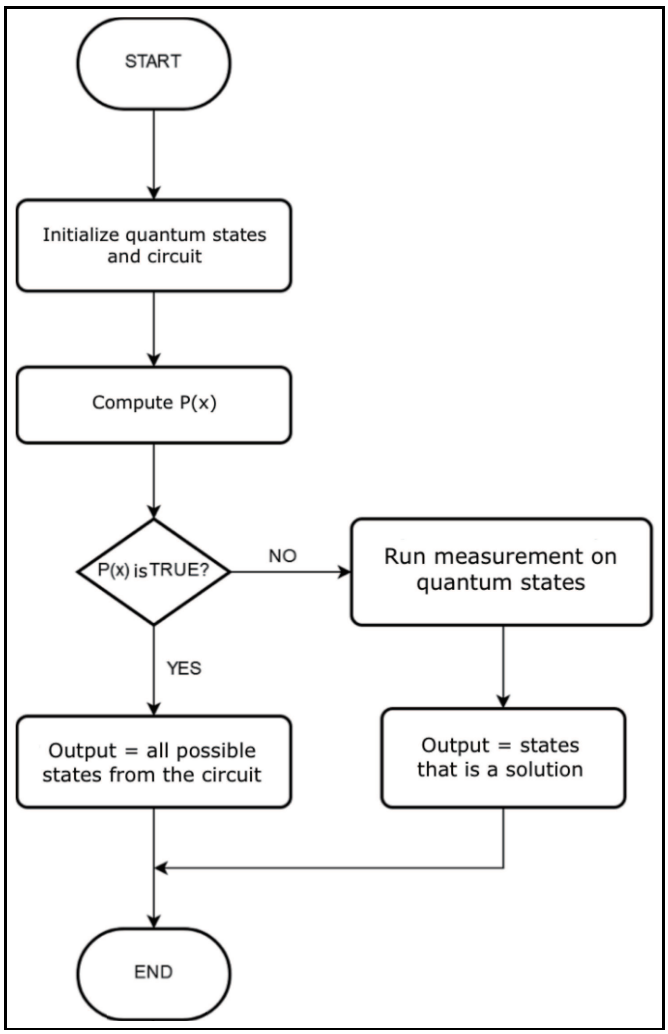

FIGURE 9. Quantum backtracking application for CIRCUIT-SAT

This study summarizes the three main steps in applying the quantum backtracking algorithm: initialization, Compute $P(x)$, and checking for a solution in case Compute $P(x)$ fails. The initialization step is done by preparing the qubits and functions for the quantum gate implementation and its inverse function. The qubits used are prepared by making qubits for input, and ancilla qubits are used according to the number used in the circuit. The qubits used in the program are the combined qubits of the input qubit and the ancilla qubit. The inverse function of the implementation is created by performing the inverse of the desired function using the dagger function. The inverse function is mandatory to guarantee the reversibility property of the quantum program, which is the

second postulate of quantum mechanics that transformations between quantum states must be unitary. Thus, the derived quantum gates have to be reversible.

The Compute $P(x)$ is implemented in Figure 10, with the number of trials set to 1,024. This variable is adjustable depending on the quantum simulator specification. The quantum circuit of the problem is simulated, and the resulting waveform is stored in the variable "result". The result is checked for the wanted states which produce an output state of "1" or "TRUE". The process requires quantum measurement when Compute $P(x)$ fails to find a solution. This step is done by implementing the circuit function, checking all possible states, and implementing the circuit inverse function. The output state is obtained from the last ancilla qubit value. Figure 11 shows the quantum module that searches for a solution.

```
trial = 1024
result = wf_sim.run_and_measure(circuit_1, trials=trial)
p = [0] * state
p_mark = 0
px = TRUE
for i in range(trial):
    index = result[i][0] + result[i][1]*2 + result[i][2]*4 + result[i][3]*8
    if result[i][wanted_qubit_1] == 1:
        if p[index] == 0:
            p[index] = 1
            p_mark = p_mark + 1
            if p_mark == state:
                break;
    else:
        px = FALSE
        break;
```

FIGURE 10. Compute $P(x)$

```
trial = 1024
result = wf_sim.run_and_measure(circuit_1, trials=trial)
p_1 = [0] * state
visited = [0] * state
p_mark = 0
i = 0
for i in range(trial):
    idx = result[i][0] + result[i][1]*2 + result[i][2]*4 + result[i][3]*8
    if visited[idx] == 0:
        temp = ""
        for j in range(len(result[i])):
            temp = str(result[i][j]) + temp
        states[idx] = temp
        p_1[idx] = result[i][wanted_qubit_1]
        visited[idx] = 1
        p_mark = p_mark + 1
        if p_mark == state:
            break
```

FIGURE 11. Checking for solution

4. **Results and Discussion.** The result of each state is stored in the last ancilla qubit of the subcircuits. When the quantum system is measured, the collapsed states are observed, and the last qubit of the collapsed state is the output of the corresponding input state. The measurement results for each state are shown in Table 1. The last qubit of the collapse state is the first qubit.

When the system is measured, a state's probability of collapse is the same for each input state in every quantum system for each subcircuit. The time needed to simulate

TABLE 1. Simulation results

No.	Collapse state	Input	Output
1	001100110100011010000000	00000	0
2	001100110100011010000001	00001	0
3	001100110100011010000010	00010	0
4	001100110100011010100011	00011	0
5	001100110100011010000100	00100	0
6	001100110100011010000101	00101	0
7	001100110100011010000110	00110	0
8	001100110100011010100111	00111	0
9	111000110111111010001000	01000	1
10	111000110111111010001001	01001	1
11	111000110111111010001010	01010	1
12	111000110111111010101011	01011	1
13	111000110111111011001100	01100	1
14	111000110111111011001101	01101	1
15	111000111111101011001110	01110	1
16	001101011111101001101111	01111	0
17	111000110111011010010000	10000	1
18	111000110111011110010001	10001	1
19	111000110111011010010010	10010	1
20	111000110111011110110011	10011	1
21	111000110111011010010100	10100	1
22	111000110111011110010101	10101	1
23	111000110111011010010110	10110	1
24	111000110111011110110111	10111	1
25	111000110110111010011000	11000	1
26	111000110110111110011001	11001	1
27	111000110110111010011010	11010	1
28	111000110110111110111011	11011	1
29	111000110110111011011100	11100	1
30	111001111010110111011101	11101	1
31	111000111110101011011110	11110	1
32	111110000010100101111111	11111	1

each subcircuit varies and is shown in Table 2. Table 2 shows the simulation time of each quantum backtracking program implemented in this study, and Table 3 gives the simulation time for each subcircuit related to the number of qubits required. In Table 3, the time used for each simulation increases along with the number of qubits used. Figure 12 and Figure 13 illustrate the increase of time needed with each qubit used in the implementation.

The ancilla qubit's value for the output is the same for every state in each subcircuit. The ancilla qubit for each circuit is different as it correlates to its structure. For accuracy and the research's F-score, it is necessary to take the true positive, true negative, false positive, and false negative values of each subcircuit result. Simulation for each subcircuit is done by measuring each subcircuit 1,024 times. The output result of each state collapses in each measurement is recorded. Ground truth data on input and output states of each subcircuit is used to calculate the confusion matrix. This ground truth data is obtained

TABLE 2. Simulation time

Subcircuit	Time (ms)			
	Initialization	Compute $P(x)$	Check for solution	Total simulation time
First	255.782	69.690	67.180	392.652
Second	247.452	67.310	62.060	376.822
Third	349.080	283.300	280.300	912.680
Fourth	1,250.074	2,082.000	2,054.000	5,386.074
Fifth	230,915.175	523,400.000	523,200.000	1,277,515.175
Sixth	253.947	66.540	64.060	384.547
Seventh	270.356	194.400	186.500	651.256
Eighth	28,351.215	66,470.000	66,620.000	161,441.215
Ninth	28,348.088	66,870.000	65,860.000	161,078.088
Tenth	255.622	62.820	64.840	383.282
Eleventh	114,246.116	258,800.000	258,700.000	631,746.116
Twelfth	809.076	1,115.600	1,135.800	3,060.476
Thirteenth	250.662	73.490	69.250	393.402

TABLE 3. The number of qubits

Qubits	Subcircuit	Total time (ms)
5	2nd Subcircuit	376.822
5	10th Subcircuit	383.282
5	6th Subcircuit	384.547
7	1st Subcircuit	392.652
7	13th Subcircuit	393.402
10	7th Subcircuit	651.256
13	3rd Subcircuit	912.680
15	12th Subcircuit	3,060.476 (3 seconds 60 ms)
16	4th Subcircuit	5,386.074 (5 seconds 386 ms)
21	9th Subcircuit	161,078.09 (2 minutes 41 seconds 441 ms)
21	8th Subcircuit	161,441.22 (2 minutes 41 seconds 78 ms)
22	11th Subcircuit	631,746.12 (10 minutes 31 seconds 746 ms)
24	5th Subcircuit	1,277,515.2 (21 minutes 17 seconds 515 ms)

by examining each subcircuit using the Deeds simulator. Table 4 shows the values for accuracy and F-score of the thirteen subcircuits.

The results of this study indicate that the quantum backtracking algorithm is applicable for satisfiability problems, i.e., CIRCUIT-SAT. The work done by Martiel and Remaud expands Montanaro's algorithm by reducing the need for space. Montanaro's algorithm assumes access to a large number of quantum bits which is not yet fully supported by any state-of-the-art quantum simulator. This application's accuracy, precision, and recall show that an improvement could be made compared to the previous work of Martiel and Remaud.

5. **Conclusions.** This study demonstrated the application of quantum backtracking programming (Montanaro's algorithm) for the circuit satisfiability problem (CIRCUIT-SAT). The study finds that the practical implementation of the algorithm in Python and pyQuil using a state-of-the-art quantum simulator from Rigetti is feasible and able to solve the

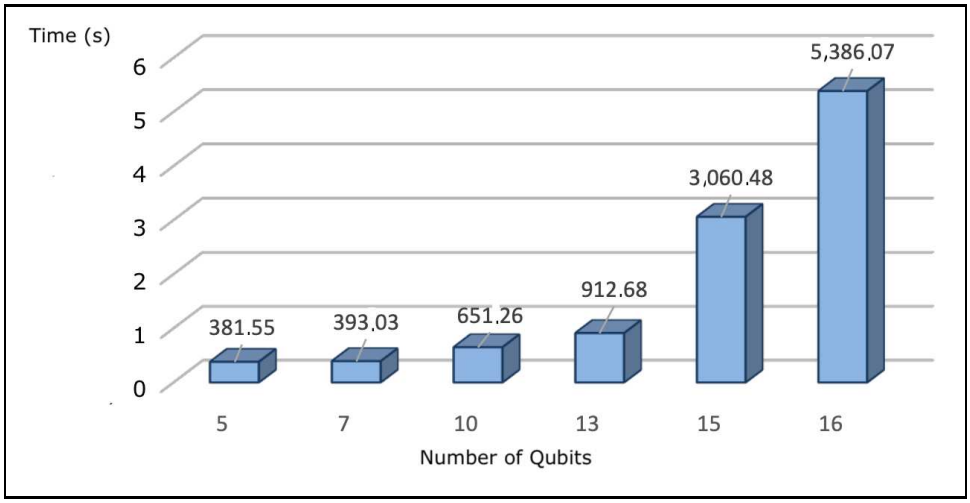

FIGURE 12. Simulation time below 10 seconds

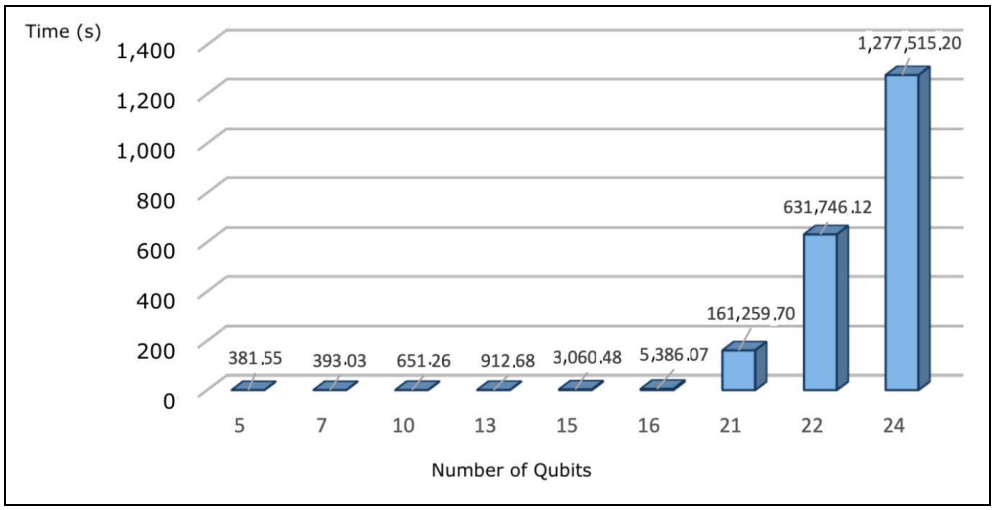

FIGURE 13. The rise in simulation time above 3 seconds

TABLE 4. Accuracy and F-score

Subcircuit	True positive	True negative	False positive	False negative	Accuracy	F-score
First	959	65	0	0	1	1
Second	136	888	0	0	1	1
Third	991	33	0	0	1	1
Fourth	302	722	0	0	1	1
Fifth	730	294	0	0	1	1
Sixth	262	762	0	0	1	1
Seventh	984	40	0	0	1	1
Eighth	26	998	0	0	1	1
Ninth	454	570	0	0	1	1
Tenth	266	758	0	0	1	1
Eleventh	521	503	0	0	1	1
Twelfth	582	442	0	0	1	1
Thirteenth	55	969	0	0	1	1

CIRCUIT-SAT problem in this study. The quantum application finds all assignments (solutions) for the reference circuit with perfect accuracy and F-score.

The application of the algorithm for the CIRCUIT-SAT is not straightforward and requires a preprocessing stage. The reference circuit for the CIRCUIT-SAT has to be recreated in quantum gates. Another challenge is the limitation of the number of qubits available for the simulation. Workaround involved breaking the reference circuit into smaller subcircuits to reduce the simulation's number of quantum bits (qubits) required. The divide and conquer approach is used for this purpose.

The completion time of the algorithm is faster than any classical algorithm implementation for the problem, with 1,277,515 ms (21 minutes 17 seconds 515 ms) to complete a simulation using 24 qubits. Thus, the quantum backtracking application found all solutions for the reference circuit in under 22 minutes. This is due to the divide and conquer approach, which allows individual simulation for each subcircuit to run separately.

The shortcoming is that the simulation could not produce the desired output automatically. Every possible state has the same probability of collapse, and this causes the simulation to give the correct value for each state. It requires manual observation to find the final results. Here are some suggestions for future work.

1) Access to a more significant amount of qubits for simulations.

2) Bridging the classical circuit for the quantum simulator and performing divide and conquer the classical circuit autonomously.

Acknowledgment. The authors would like to thank Universitas Multimedia Nusantara for the support of this research work.

REFERENCES

[1] J. Gu, P. W. Purdom, J. Franco and B. W. Wah, Algorithms for the satisfiability (SAT) problem, in *Handbook of Combinatorial Optimization*, D. Z. Du and P. M. Pardalos (eds.), Boston, MA, Springer, 1999.

[2] A. Wicaksana, Anthony and A. W. Wicaksono, Web-app realization of Shor's quantum factoring algorithm and Grover's quantum search algorithm, *Telkomnika (Telecommunication Comput. Electron. Control.)*, DOI: 10.12928/TELKOMNIKA.v18i3.14755, 2020.

[3] Anthony and A. Wicaksana, Implementation of Grover's quantum search algorithm using Rigetti forest, *Int. J. Eng. Adv. Technol.*, vol.8, no.6S3, DOI: 10.35940/ijeat.F1018.0986S319, 2019.

[4] A. W. Wicaksono and A. Wicaksana, Implementation of Shor's quantum factoring algorithm using ProjectQ framework, *Int. J. Eng. Adv. Technol.*, DOI: 10.35940/ijeat.F1009.0986S319, 2019.

[5] H. Matsuura, Probability of fuzzy set theory and probability amplitude of quantum neurons (similarities and physical quantities of quantum neural networks), *International Journal of Innovative Computing, Information and Control*, vol.12, no.2, pp.503-518, 2016.

[6] A. Ambainis and M. Kokainis, Qantum algorithm for tree size estimation, with applications to backtracking and 2-player games, *Proc. of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC2017)*, DOI: 10.1145/3055399.3055444, 2017.

[7] D. J. Moylett, N. Linden and A. Montanaro, Quantum speedup of the traveling-salesman problem for bounded-degree graphs, *Phys. Rev. A*, DOI: 10.1103/PhysRevA.95.032323, 2017.

[8] J. Islam, S. T. Meraj, B. M. Negash, K. Biswas, H. K. Alhitmi, N. I. Hossain, P. M. Vasant and J. Watada, Modified quantum particle swarm optimization for selective harmonic elimination (SHE) in a single-phase multilevel inverter, *International Journal of Innovative Computing, Information and Control*, vol.17, no.3, pp.959-971, 2021.

[9] Y.-T. Chen and W.-J. Chen, Optimizing the obstacle avoidance trajectory and positioning error of robotic manipulators using multigroup ant colony and quantum-behaved particle swarm optimization algorithms, *International Journal of Innovative Computing, Information and Control*, vol.17, no.2, pp.595-611, 2021.

[10] M. Mneimneh, I. Lynce, Z. Andraus, J. Marques-Silva and K. Sakallah, A branch-and-bound algorithm for extracting smallest minimal unsatisfiable formulas, in *Theory and Applications of Satisfiability Testing*, F. Bacchus and T. Walsh (eds.), Berlin, Springer, 2005.

[11] O. Strichman, Pruning techniques for the SAT-based bounded model checking problem, in *Correct Hardware Design and Verification Methods. CHARME 2001. Lecture Notes in Computer Science*, T. Margaria and T. Melham (eds.), Berlin, Heidelberg, Springer, 2001.
[12] A. Montanaro, Quantum-walk speedup of backtracking algorithms, *Theory Comput.*, DOI: 10.4086/toc.2018.v014a015, 2019.
[13] S. Martiel and M. Remaud, Practical implementation of a quantum backtracking algorithm, in *Theory and Practice of Computer Science. SOFSEM 2020. Lecture Notes in Computer Science*, A. Chatzigeorgiou, R. Dondi, H. Herodotou et al. (eds.), Cham, Springer, 2020.
[14] Rigetti Computing, *Welcome to the Docs for pyQuil!*, https://pyquil-docs.rigetti.com/en/stable/, Accessed on Sep. 05, 2022.

Author Biography

Vionie Laorensa graduated from the Department of Informatics at Universitas Multimedia Nusantara. She was a young researcher with a deep interest in quantum algorithms and application. She recently worked as an application software backend engineer.

Arya Wicaksana is a lecturer at the Department of Informatics at UMN. He received Master Degree in VLSI Engineering from Universiti Tunku Abdul Rahman. He successfully demonstrated the UTAR first-time success ASIC design methodology on a multi-processor system-on-chip project using 0.18 μm processing technology in 2015. His main research interests are blockchain applications and computational intelligence. He recently worked on a decentralized autonomous social media. He is affiliated with ACM and IEEE as a professional member. He has served as an invited reviewer in IEEE Access, IJNMT, and IFERP and an invited author in IntechOpen and other scientific publications.