Enhancing and Optimizing Grover's Algorithm for Unified Random Sampling of Software Product Lines

Anonymous Author(s)

Abstract

This paper improves upon the previous implementations of Grover's algorithm for uniform random sampling in the context of software product lines. By addressing inefficiencies in quantum circuit design for software product lines (SPLs) we manage to reduce the quantum circuit depth and width significantly. Additionally we propose novel approaches to designing a Grover oracle for software product lines. Specifically, we focus on two main tasks: Removing redundant quantum operations that increase computational overhead and introducing lookup tables to optimize qubit usage. These improvements were evaluated using SPL models by the BURST benchmarking suite and SPL tool FeatureIDE, measuring both the circuit depth and width. The results demonstrate that our optimizations lead to substantial reductions in circuit size, particularly for larger feature models, making Grover's algorithm more scalable and efficient for practical applications in SPL testing and management.

Keywords

Quantum Computing, Grover's Algorithm, Software Product Lines

ACM Reference Format:

Anonymous Author(s). 2024. Enhancing and Optimizing Grover's Algorithm for Unified Random Sampling of Software Product Lines. In *Proceedings of Internation Conference on Software Engineering (ICSE '25).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Software product lines (SPLs) represent an approach to software engineering, facilitating the systematic and efficient management of variations within a family of related software systems. SPLs permit the development of a set of features, from which a multitude of product variants can be constructed, thereby addressing the requirements of customers or market segments. However, this flexibility introduces significant complexity, particularly with regard to testing and verifying the correctness of individual product configurations.

In the context of software product lines, the term *sampling* is used to describe a process through which a subset of the total population of items is selected for further analysis. Uniform random sampling is of particular importance for unbiased testing, analysis, and quality assurance [6, 12, 13, 15], as it ensures that each product variant has an equal chance of being selected, thereby avoiding biases that might lead to incorrect conclusions about the overall product line.

The potential of quantum computing to solve complex problems more efficiently than classical approaches offers promising solutions for many computational challenges, including those found in SPLs. In this context, Grover's Quantum Algorithm represents a powerful tool for searching unsorted databases or solving satisfiability problems [8, 14]. When applied to a Boolean formula representing constraints in an SPL, Grover's algorithm can be employed to identify satisfying configurations with a quadratic speedup compared to brute forcing. This ability makes it a strong candidate for enabling uniform random sampling in SPLs, as it can efficiently identify valid product configurations from a large space.

Although there already are existing approaches for a quantum implementation, they frequently allow for optimization. Previous research on the application of Grover's algorithm to SPLs did not fully optimize the quantum circuit, resulting in inefficiencies in gate operations and qubit utilization. Specifically, the prior implementation failed to remove obsolete gates, such as pairs of X gates that cancel each other out, and did not adequately identify opportunities for qubit reuse, where qubits performing the same function could have been shared rather than redundantly created. These oversights not only increase the complexity of the quantum circuit but also reduce the overall efficiency and practicality of the quantum algorithm.

This paper builds on the foundational work of previous research [2] by introducing significant improvements to the quantum circuit design for uniform random sampling in SPLs. By optimizing the quantum gate sequence and implementing effective qubit reuse strategies, we aim to reduce the resource requirements and enhance the performance of Grover's algorithm in this context. Our contributions not only streamline the quantum computation process but also make the application of quantum algorithms to SPLs more feasible and effective, paving the way for more advanced and efficient quantum software engineering practices.

2 Background

This chapter provides the requisite background information for understanding the concepts and techniques utilized in our enhanced quantum implementation for uniform random sampling in software product lines (SPLs). It covers the necessary fundamentals of SPLs, quantum computing principles, and Grover's algorithm. This background chapter will only explain the knowledge needed to understand the improvement implemented and proposed by this paper.

2.1 Software Product Lines

A software product line (SPL) represents a strategic approach to software engineering that prioritizes the systematic production of a

ICSE '25, April 27–May 3, 2025, Ottawa, Ontario, Canada

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

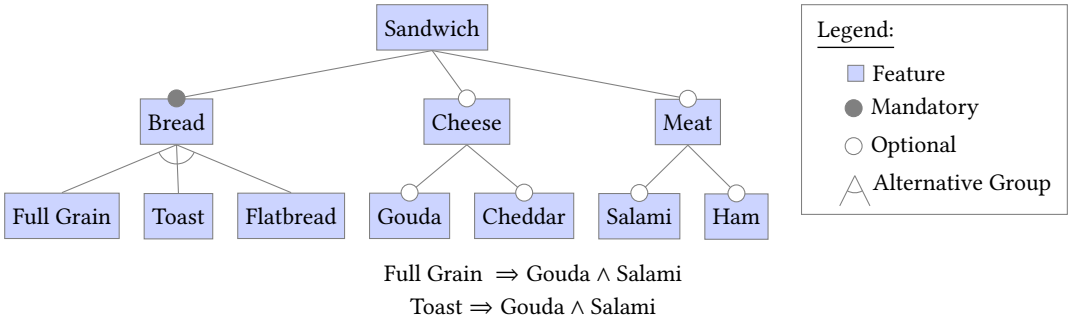


Figure 1: Feature Model describing a Sandwich with additional constraints.

family of related software products [10, 16]. In contrast to the development of each product from its inception, SPLs prioritize the reuse of fundamental assets, including code, designs, and documentation, across a range of products [3]. This methodology is particularly advantageous in domains where products exhibit a shared set of characteristics while necessitating adaptations for diverse markets, customer requirements, or operational contexts. SPLs endeavor to enhance productivity, expedite time-to-market, and elevate product quality by capitalizing on this common ground while navigating the inherent variability.

In the context of software product lines, constraints are defined as rules or conditions that dictate the valid combinations of features within a given product configuration. These constraints guarantee that the selected features are compatible and that the resulting product adheres to the intended design and functionality. Constraints may be expressed in a number of ways, including as mandatory features, mutual exclusions (where the inclusion of one feature precludes the inclusion of another), or dependencies (where the inclusion of one feature necessitates the inclusion of another). Such constraints are typically represented as Boolean formulae, where each feature is a variable, and the relationships between features are expressed through logical operators.

Feature models are a fundamental instrument utilized in the context of SPLs to illustrate the commonalities and variabilities among the products within a product line. They precisely describe the space of all *valid* configurations of an SPL. This description can be translated into a conjunctive normal form (CNF), which can then be solved using SAT solvers, as demonstrated by *Sunderman et al.* in [18]. A feature model offers a visual representation of the features that may be present in a product within a given product line, organized into a hierarchical structure. This structure encompasses mandatory features, optional features, and alternative or mutually exclusive features. Feature models facilitate comprehension and management of the configuration options available, ensuring that any derived product aligns with the constraints and dependencies defined in the model. Feature models can additionally easily be translated into a SAT problem that can be solved in classical computing by using various SAT solvers.

For this paper we provide an example feature model described in Figure 1. It represents a product line for creating sandwiches, with the core components being Bread, Cheese, and Meat. Bread is a mandatory feature with options for Full Grain, Toast, and Flatbread, while Cheese and Meat are optional, offering choices such as Gouda, Cheddar, Salami, and Ham. The model also includes additional constraints: if Full Grain bread is chosen, Gouda and Salami must be included, and similarly, selecting Toast mandates the inclusion of Gouda and Salami.

2.2 Quantum Computing

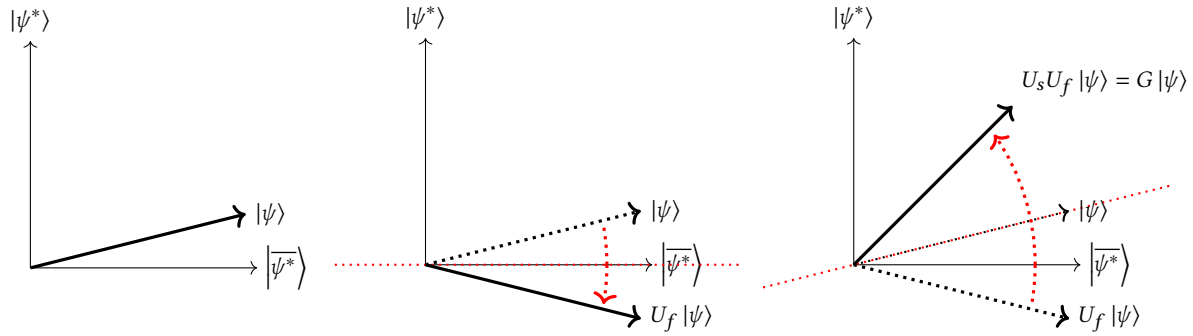
Quantum computing offers a new paradigm in computation, leveraging principles from quantum mechanics such as superposition and entanglement. Unlike classical computers, which rely on binary bits, quantum computers use qubits that can exist in multiple states simultaneously. While the technology is still in its early stages, it has the potential to provide advantages in specific problem areas like cryptography, optimization, and material science. However, it is important to note that quantum computers are not universally faster than classical computers and are best suited to particular types of computational problems.

The fundamental unit of quantum computing is the quantum gate, which is analogous to the classical logic gates utilized in traditional computing and serves as the basic component of a quantum circuit. These gates facilitate the manipulation of qubits (quantum bits) through operations such as superposition and entanglement. It is noteworthy that certain quantum operations can negate one another, such that applying a gate followed by itself has no effect on the qubit's state. This property is pivotal for error correction and algorithm design. Quantum circuits are composed of sequences of these gates, which permit the execution of complex computations through the parallel application of transformations on qubit states.

Two key metrics for assessing the complexity and efficiency of quantum circuits are depth and width.

- **Width** The width of a quantum circuit is defined as the number of qubits utilized, and it determines the amount of information that the circuit can process in parallel.
- **Depth** The depth of a quantum circuit is a measure of the number of layers of gates applied sequentially. It is a reflection of the length of time required for the computation, as each layer can be executed simultaneously.

A circuit with greater depth requires more time to execute, whereas a circuit with greater width can accommodate more complex states or parallel operations. The interplay between depth and width is



(a) Preparation (b) Oracle (c) Diffusion

Figure 2: All steps of Grover's algorithm visualized in the plane spanned by all valid configurations $|\psi^*\rangle$ and all non-valid configurations $\left|\overline{\psi^*}\right\rangle$

crucial for understanding the resource requirements and potential limitations of quantum algorithms.

2.3 Quantum Programming

Qiskit is a widely used open-source quantum computing framework developed by IBM [17], designed to enable the creation, simulation, and execution of quantum circuits on both simulators and real quantum hardware. It provides a high-level Python interface for designing quantum algorithms, optimizing quantum circuits, and performing tasks such as quantum error correction, gate decomposition, and circuit transpilation. One of its core strengths is its ability to transpile circuits to match the constraints of actual quantum processors, such as gate set compatibility and qubit connectivity. This makes it highly versatile for both research and practical applications in quantum computing.

Complementing Qiskit is Quantum Assembly Language (QASM) [5] [4], a low-level, hardware-agnostic language used to describe quantum circuits as a series of quantum gate operations and measurements. QASM serves as the interface between high-level quantum algorithms and quantum hardware, translating the abstract structure of quantum circuits into executable instructions. This combination of Qiskit's high-level capabilities and QASM's precise control over quantum operations enables researchers and developers to experiment with quantum algorithms on various platforms, ranging from simulators to quantum hardware.

2.4 Grover's Algorithm

One of the most notable quantum algorithms is Grover's algorithm. Published by L. K. Grover in 1996 [7], it exemplifies the power of quantum circuits by markedly accelerating the search process in unsorted databases, offering a quadratic speedup compared to classical algorithms. As searching solutions to logical formulae can be translated into finding possible solutions to the formula in a database containing all possible variable assignments, it can be seen that Grover's algorithm can also be used to solve logical formulae. Grover can be mainly divided into three important steps

(1) **Preparation:** Prepares all n non-ancilla qubits into a superposition state $|\psi\rangle$ consisting of 2^n possible states. In the context of Software Product Lines, n describes the amount of features.
(2) **Oracle:** Marks all states that are valid solutions of our problem by negating their phase. In our context this means to mark all states that are valid configurations of our feature model. In a mathematical context this means to multiply our state $|\psi\rangle$ by the oracle operator/matrix U_f resulting in our temporary state $U_f\,|\psi\rangle$.
(3) **Diffusion:** In the final step of Grover's algorithm we take the state $U_f\,|\psi\rangle$ and reflect it about the prepared state $|\psi\rangle$. Mathematically this means we multiply by another operation matrix U_s, giving us the state $U_sU_f\,|\psi\rangle$.

This process is visualized in Figure 2. It displays the current state of our quantum system as a bold arrow in the plane of all possible 2^n configurations. This plane is spanned by the two states $|\psi^*\rangle$ which describes a state consisting only of *valid* configurations of our feature model, and state $\left|\overline{\psi^*}\right\rangle$ which consists of all *invalid* configurations. The last two steps, namely *Oracle* (U_f) and *Diffusion* (U_s) together are also called a *Grover iteration* (U_sU_f). This Grover iteration has to be repeated for $\sim \frac{\pi}{4}\sqrt{N/M}$ solutions, where $N = 2^n$ is the number of all *possible* configurations, while M describes all *valid* configurations.

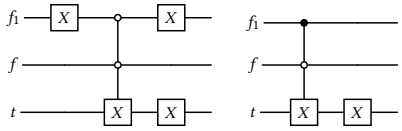
3 State of the Art

This subsection will provide the necessary background knowledge of our previous work [9], which this paper extends. In reference to the cited thesis, we introduced a method for translating feature models that does not necessarily adhere to the CNF (Conjunctive Normal Form) format. Table 1 shows this translation. Instead, it employs a propositional representation that encompasses a broader range of logical operators, not only including the classical `NOT`, `AND`, and `OR` gates but also proposing a quantum-equivalent interpretation for the gates `Implication`, `Equivalence` and `XOR`.

Rule	Feature model primitive	propositional term
1	r is root feature	r
2	f_1 is *mandatory* sub(-feature) of f	$f_1 \iff f$
3	f_1 is *optional* sub of f	$f_1 \implies f$
4	$f_1, \ldots, f_n$ OR-subs of f	$f_1 \vee \cdots \vee f_n \iff f$
5	$f_1, \ldots, f_n$ XOR-subs of f	$f_1 \vee \cdots \vee f_n \iff f$ $\wedge \bigwedge_{i<j} \neg(f_i \wedge f_j)$
6	f_1 requires f_2	$f_1 \implies f_2$
7	f_1 excludes f	$\neg(f_1 \wedge f_2)$

Table 1: Feature model primitives and their corresponding, propositional conjunction term.

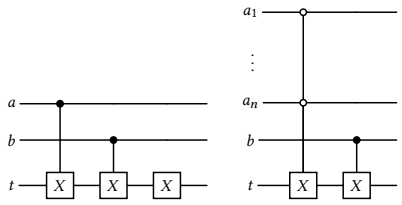
Implication The new implementation of an `Implication` gate is equal to an optimized version of the NNF version. A quantum circuit of this can be seen in Figure 3.



(a) $\neg f_1 \vee f$ (b) $\neg(f_1 \wedge \neg f)$

Figure 3: Comparison of two alternative implementations of implication using quantum gates.

Equivalence If we take a look at Table 1, we can see that our `Equivalence` gate is either used to act on two input qubits, or a conjunction of input qubits and a single qubit.
Our gate circuit for both of these cases can be seen in Figure 4.



(a) $a \Leftrightarrow b$ (b) $\bigvee_{i=1\ldots n} a_i \Leftrightarrow b$

Figure 4: Example circuits for both rules that include an `Equivalence`.

XOR While normally a n-ary `XOR` turns on for an uneven amount of toggled input bits, in the context of this paper a n-ary `XOR` refers to an operation that flips it's target when only a single input qubit is 1. This is due to semantics of alternative groups in feature models. XOR operations on only 2 operands (qubits) are basically the negated version of our Equivalence gate, checking exclusive disjunction for n. Rule 5 in Table 1, however, shows that we have to operate with multiple (n) input qubits. The new implementation of `XOR` is introduced as **Disjunction of possibilities**:
This is designed by building one term for each possibility, so n in total. For the i-th term we check if only the i-th qubit is active while the others are inactive. Finally we build an OR with all our term ancillas as controls. As a formula this would look like:

$$\bigvee_{i=0}^{n} (f_i \wedge \bigwedge_{\substack{j=0 \\ j\neq i}}^{n} \neg f_j) = (f_0 \wedge \neg f_1 \wedge \ldots \neg \wedge f_n)$$
$$(\vee \neg f_0 \wedge f_1 \wedge \neg f_2 \wedge \cdots \wedge \neg f_n)$$
$$\vdots$$
$$(\vee \neg f_0 \wedge \cdots \wedge \neg f_{n-1} \wedge f_n) \quad (1)$$

While this works, utilizing the fact that always only one of these terms can be achieved, helps us to drastically reduce the width. Figure 5 shows this optimized circuit.

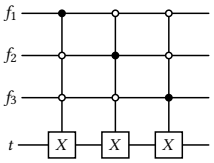


Figure 5: Optimized circuit for possibility disjunction for $n = 3$.

4 Implementation

This implementation section outlines the practical steps and theoretical enhancements made to improve upon the work of *Kadner et al.* [9]. It details optimizations at the quantum circuit and gate level, focusing on reducing circuit depth and width for better efficiency. We also provide insights into how these improvements were translated into code, bridging the gap between theory and practical application. By explaining both the conceptual and technical aspects, this section highlights the key strategies used to enhance the performance of Grover's algorithm in the context of software product lines.

4.1 Redundant Operations

In Quantum Computing, some operations are self-inverse, meaning that their operation matrix is involutory. For example, given the Hadamard Gate $H = -\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ we see that multiplying H by

itself will result in an identity matrix:

$$H^2 = -\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} - \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \quad (2)$$

To further optimize the quantum circuit, we eliminate these self-inverse operations. This is done by creating a two-dimensional array that tracks the operations for each qubit. We then iteratively traverse the array, identifying and removing successive pairs of self-inverting gates. This process is repeated until no more such pairs are found. While this optimization reduces the circuit depth in the generated QASM output, it does not affect the depth when transpiled using Qiskit, since its *transpile()* method inherently removes redundant gates. Nevertheless, this improvement results in a shorter depth in the QASM representation, potentially providing future flexibility by reducing dependencies on specific frameworks or libraries.

4.2 Lookup Tables

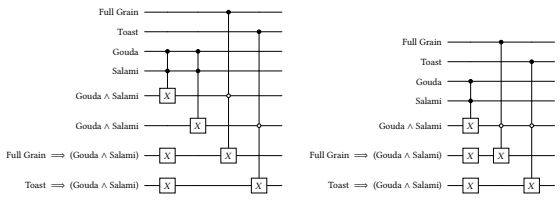
It is not uncommon for multiple constraints within one feature model to exhibit similarities in their constituent parts, given that features often interact in ways that are not mutually exclusive. For example, certain features may be required or excluded together across different constraints, leading to the emergence of repeated patterns in the Boolean formulae. These repeated patterns are also represent in the previous implementation of the quantum circuit. As shown in Table 1 all constraints require one or even multiple additional qubits, so called *ancilla qubits*. These qubits store temporary results that are needed for entangling the constraint to the final result state. The repeated patterns thus cause some ancilla qubits to store the same state and hence be duplicates.

This is where our optimization comes in. When creating the circuit, we try to build a lookup table for all previously created (ancilla) qubits and what logical formulae they represent. When creating the qubits for constraints we look for qubits that store the same state as is required for our constraint. If such qubit is found, we use it instead of creating a new one, saving on both depth and width. When looking at our example *Sandwich* feature model shown in Figure 1, we can see that both constraints contain the term "Gouda ∧ Salami".

Figure 6 shows the comparison of the resulting quantum circuits that represent these constraints. Subfigure (a) was generated using the previous version of the implementation, whereas (b) was created using a revised version that incorporates lookup tables. We can see that the circuit width of our new implementation was reduced by one qubit. This was due to the removal of the duplicate qubit "Gouda ∧ Salami" in the old implementation. In the new implementation both final constraints qubits ("Toast $\implies$ (Gouda ∧ Salami)" and "Full Grain $\implies$ (Gouda ∧ Salami)") use the same qubit as control instead of creating a new one as it was done in the old implementation.

5 Related Work

In the following section we will take a look into current literature on this topic. This is important as it positions our thesis within the larger quantum computing landscape and shows how it differs from other work on the topic.



(a) Old Implementation (b) New Implementation

Figure 6: Quantum circuits satisfying the constraints of our example feature model 1.

Hangleiter and Eisert explore various random sampling techniques in quantum circuits and their computational advantages in their paper [8]. They define quantum random sampling as the process of obtaining samples from random quantum computations. However, this paper takes a different approach. Instead of focusing on random quantum computations, it addresses the challenge of generating uniform random samples from a distribution under specific constraints, a task that cannot be accomplished solely through random quantum computation.

In the context of Grover's algorithm, recent advancements have explored ways to enhance its efficiency. *Xiang Li et al. (2023)* proposed in [11] several optimization techniques, including the W-cycle Oracle, Oracle compression, and Randomized Grover's Algorithm to reduce circuit complexity and increase computational efficiency.

W-cycle Oracle It allows the creation of at most 2^m boolean equations using only m ancilla qubits, at the cost of a deeper circuit.

Oracle Compression Oracle Compression is the same type of optimization as we are proposing in this paper by removing redundant gates.

Randomized Grover's Algorithm The idea for this method, is that in each iteration the amplitudes for valid solutions will always get amplified and the solutions to the specific subset of problems will get amplified but will get reduced again in other iterations. However, they concluded that convergence is not always guaranteed.

These approaches focus on improving the implementation of the Grover oracle by reducing the number of ancilla qubits, eliminating redundant gates, and applying a randomized oracle to amplify solutions in different iterations, albeit without guaranteed convergence. While these optimizations fall beyond the scope of this paper, which centers on reducing the depth and width of a giving implementation, they represent promising complementary strategies for further improving Grover's algorithm.

6 Evaluation

In this chapter, we evaluate our implementation and compare it to the previous implementation by *Kadner et al.* in [9]. We evaluate based on two critical quantum circuit metrics: Depth and Width.

Feature Model	# of features	Previous Depth	New Depth	Previous Width	New Width
Car	10	32	32	21	21
Sandwich	11	32	30	35	35
Bike	54	58	58	92	92
axTLS	96	304	304	296	271
cLib	313	902	856	1103	909
BusyBox	854	2998	2794	2558	2390
Toolkit	1179	27956	4736	22449	2558

Table 2: Width and Depth of our test models and their respective size in amount of features (# of features). The depth was calculated by transpiling the circuit onto the statevector backend. We compare the old implementation presented in [9] and the new implementation introduced in this paper.

These metrics, which are crucial for understanding the efficiency and scalability of quantum circuits, are discussed in detail in Section 2.2 of our Background. Finally, we compare the results with our previous work [9] and assess on possible improvements or deteriorations.

6.1 Methodology

The methodolgy follows the approach outlined by *Ammermann et al.* in [2] and *Kadner et al.* in [9]. For proper benchmarking and comparison we use the same feature models as in the previous work. These are feature models by the BURST benchmarking suite [1] and examples by the software product line tool FeatureIDE [19]. We begin by creating a QASM output for both the previous implementation and our proposed implementation. Afterwards we parse this output into a Qiskit object using the *parse()* method provided by Qiskit in their Python module `qiskit_qasm3_import`. Finally working on a Qiskit object allows us to utilize the *depth()* and *width()* methods it provides. The evaluation will focus on comparing the performance of our implementation against previous work, highlighting improvements in both circuit depth and width.

6.2 Depth

To evaluate the depth of our new implementation, we performed a comparison between the computed depth of our approach and the one presented in the reference work [2, 9]. The results, as shown in Table 2, indicate a significant improvement in depth, especially in the case of larger feature models. These larger models tend to introduce more complex constraints and, as a result, often contain more duplicated qubits in the previous implementation. The increased number of duplicated qubits provides an opportunity for our new method to apply more efficient optimization techniques, leading to a more substantial reduction in circuit depth. In contrast, smaller feature models with fewer constraints and duplicated qubits do not offer the same level of optimization potential, resulting in relatively smaller improvements. This demonstrates that the benefits of our implementation become more pronounced as the complexity of the feature models increases, highlighting its effectiveness for scaling to more complex quantum circuits.

6.3 Width

We observe similar improvements in the width of our new implementation as in the circuit depth. As discussed in Section 4.2, our approach significantly reduces the need to duplicate qubits, resulting in significant width savings. By minimizing qubit duplication, our method optimizes resource utilization, especially in more complex circuits. Table 2 illustrates that these gains are most evident in larger feature models, which are more prone to duplicated logical terms. In such models, the elimination of redundant qubits has a greater impact, resulting in a significant reduction in circuit width. Conversely, smaller feature models, which typically have fewer duplications, show less dramatic improvements. Nevertheless, the results show that our implementation is particularly beneficial for feature models of high complexity, where it effectively reduces both the number of qubits and the circuit overhead.

7 Conclusion

In conclusion, this paper has presented significant improvements to the application of Grover's algorithm for uniform random sampling, particularly in the context of software product lines (SPLs). By optimizing quantum circuits through the removal of redundant operations, the reuse of qubits, and the development of more efficient logic gate structures, the proposed approach reduces both the depth and width of quantum circuits. These optimizations enhance the resource efficiency of the algorithm and demonstrate scalability across larger feature models, as evidenced by the evaluation. Ultimately, these advancements facilitate the practical application of quantum computing for solving complex problems in software engineering, paving the way for future innovations in SPL management and testing.

References

[1] Mathieu Acher, Gilles Perrouin, and Maxime Cordy. 2021. BURST: a benchmarking platform for uniform random sampling techniques. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume B* (Leicester, United Kindom) *(SPLC '21)*. Association for Computing Machinery, New York, NY, USA, 36–40. https://doi.org/10.1145/3461002.3473070

[2] Joshua Ammermann, Tim Bittner, Domenik Eichhorn, Ina Schaefer, and Christoph Seidl. 2023. Can Quantum Computing Improve Uniform Random Sampling of Large Configuration Spaces? (Preprint). arXiv:2307.14703 [quant-ph] https://arxiv.org/abs/2307.14703

[3] Paul Clements and Linda Northrop. 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional.

[4] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. 2022. OpenQASM 3: A Broader and Deeper Quantum Assembly Language. *ACM Transactions on Quantum Computing* 3, 3 (sep 2022), 1–50. https://doi.org/10.1145/3505636

[5] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. 2017. Open Quantum Assembly Language. arXiv:1707.03429 [quant-ph] https://arxiv.org/abs/1707.03429

[6] Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. 2018. Efficient Sampling of SAT Solutions for Testing. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 549–559. https://doi.org/10.1145/3180155.3180248

[7] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. arXiv:quant-ph/9605043 [quant-ph] https://arxiv.org/abs/quant-ph/9605043

[8] Dominik Hangleiter and Jens Eisert. 2023. Computational advantage of quantum random sampling. *Reviews of Modern Physics* 95, 3 (July 2023). https://doi.org/10.1103/revmodphys.95.035001

[9] Marwin Kadner. 2023. *Towards Improved Uniform Random Sampling using Grover's Algorithm*. Bachelor Thesis. Karlsruhe Institute of Technology. https://github.com/kadnermarwin/Bachelorarbeit-Marwin-Kadner

[10] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Software Engineering Institute.

[11] Xiang Li, Hanxiang Shen, Weiguo Gao, and Yingzhou Li. 2023. Resource Efficient Boolean Function Solver on Quantum Computer. arXiv:2310.05013 [quant-ph]

[12] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 61–71. https://doi.org/10.1145/3106237.3106273

[13] Jeho Oh, Paul Gazzillo, and Don Batory. 2019. t-wise Coverage by Uniform Sampling. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A* (Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 84–87. https://doi.org/10.1145/3336294.3342359

[14] Jose J. Paulet, Luis F. LLana, Hernán Indíbil Calvo, Mauro Mezzini, Fernando Cuartero, and Fernando L. Pelayo. 2023. Heuristics for Quantum Computing Dealing with 3-SAT. *Mathematics* 11, 8 (2023). https://doi.org/10.3390/math11081888

[15] Quentin Plazar, Mathieu Acher, Gilles Perrouin, Xavier Devroey, and Maxime Cordy. 2019. Uniform Sampling of SAT Solutions for Configurable Systems: Are We There Yet?. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. 240–251. https://doi.org/10.1109/ICST.2019.00032

[16] Klaus Pohl and Andreas Metzger. 2018. *Software Product Lines*. 185–201. https://doi.org/10.1007/978-3-319-73897-0_11

[17] Qiskit contributors. 2023. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2573505

[18] Chico Sundermann, Thomas Thüm, and Ina Schaefer. 2020. Evaluating SAT solvers on industrial feature models. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems* (Magdeburg, Germany) *(VaMoS '20)*. Association for Computing Machinery, New York, NY, USA, Article 3, 9 pages. https://doi.org/10.1145/3377024.3377025

[19] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. *Science of Computer Programming* (01 2014). https://doi.org/10.1016/j.scico.2012.06.002

Received 11 November 2024; revised dd-mm-yyyy; accepted dd-mm-yyyy