mathematics

IMPACT FACTOR 2.592

CITESCORE 2.9

Article

# Heuristics for Quantum Computing Dealing with 3-SAT

Jose J. Paulet, Luis F. LLana, Hernán Indíbil Calvo, Mauro Mezzini, Fernando Cuartero and Fernando L. Pelayo

Special Issue
Quantum Computing Algorithms and Quantum Computing Simulators

Edited by
Dr. Fernando L. Pelayo, Dr. Mauro Mezzini and Dr. Pedro Valero-Lara

https://doi.org/10.3390/math11081888

mathematics

*Article*

# Heuristics for Quantum Computing Dealing with 3-SAT

**Jose J. Paulet [1,*], Luis F. LLana [1], Hernán Indíbil Calvo [2], Mauro Mezzini [3], Fernando Cuartero [2] and Fernando L. Pelayo [2,*]**

[1] Departamento de Sistemas Informaticos y Computacion, Facultad de Informatica, Complutense University, 28040 Madrid, Spain
[2] Departamento de Sistemas Informaticos, Escuela Superior de Ingenieria Informatica, University of Castilla—La Mancha, 02071 Albacete, Spain
[3] Dipartimento di Scienze della Formazione, Via del Castro Pretorio 20, UniRoma3, 00185 Rome, Italy
* Correspondence: jpaulet@ucm.es (J.J.P.); fernandoL.pelayo@uclm.es (F.L.P.)

**Abstract:** The SAT problem is maybe one of the most famous NP-complete problems. This paper deals with the 3-SAT problem. We follow a sort of incremental strategy to save computational costs with respect to the classical quantum computing approach. We present an heuristics that leads this strategy, improving the performance of the purely random incremental scheme. We finally validate our approach by means of a thorough empirical study.

**Keywords:** computational efficiency; DPLL algorithm; Grover algorithm; SAT problem

**MSC:** 68Q12

check for updates

**Citation:** Paulet, J.J.; LLana, L.F.; Calvo, H.I.; Mezzini, M.; Cuartero, F.; Pelayo, F.L. Heuristics for Quantum Computing Dealing with 3-SAT. *Mathematics* **2023**, *11*, 1888. https://doi.org/10.3390/math11081888

Academic Editor: Catalin Stoean

Received: 16 February 2023
Revised: 22 March 2023
Accepted: 13 April 2023
Published: 16 April 2023

## 1. Introduction

The Boolean satisfiability problem (SAT) is the first known NP-complete problem [1]. It has applications in different fields, such as cryptanalysis [2], hardware verification [3], AI planning [4] and medicine [5].

There are two main ways to face the SAT problem from the algorithmic perspective. On one hand, those supported by the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [6] focus on, in essence, backtracking, and on the other hand, those based on local searching are led by some heuristics changing from one state to another one until reaching an ending condition that corresponds to a valid interpretation. Some examples of the second option are hill climbing/gradient descent and simulated annealing [7], where the latter can be directly powered by annealing quantum computing.

SAT problems can be dealt with as a structured searching problem, as shown in [8]. This work presents a quantum algorithm for nested searching over structured problems. First, a Grover's search is carried out over the $i$ first qubits out of a total of $n$. This uses an oracle that inverts the phase of a possible solution, conditioned by the set of clauses that contains the $i$ first variables that were satisfied. The number of iterations of this first step is $\sqrt{\frac{2^i}{n_A}}$, where $n_A$ is the number of possible solutions. The second step performs $m \approx \sqrt{2^{n-i}}$ iterations of a standard search in the remaining $n-i$ qubits. The last step applies $r \approx \sqrt{\frac{n_A}{n_{AB}}}$ iterations of $I_t \cdot U^{-1} \cdot (I_s \otimes I_{s'}) \cdot U$, where:

- $U$ is the circuit of the two previous steps.
- $U^{-1}$ is the inverse ordered $U$ circuit (right to left).
- $I_s$ and $I_{s'}$ are the conditional phase inversion of initial states of the $i$ first qubits and $n-i$ qubits, respectively.
- $I_t$ is the oracle for the solutions of the problem.
- $n_{AB}$ is the number of solutions of the problem.

In order to develop such a structured quantum search, some extra information about the number of tentative solutions at different levels, together with where these levels should be located, is required. This is not available in our case.

A more restrictive version of this problem focused on exact satisfiabiality, XSAT, and occupation problems, with $n$ variables and $m$ clauses has been studied in [9] and solved by Grover's algorithm in $O\left(\sqrt{2^{n-M'}}\right)$, where $M'$ is the number of independent clauses for the corresponding XORSAT problem that the original problem is reduced to. In fact, it provides a reduction in the searching space by means of a transformation of the original problem. The Oracle used in this article is formed of three main blocks. The first block is a mapping between the $k$-configuration of the XORSAT problem and the $n$-configuration of the original problem. The second block is devoted to computing the number of satisfied clauses. The third block is a conditional phase inversion over the satisfied clauses (the phase inversion is applied when the number of satisfied clauses equals the number of clauses).

In addition, Grover's algorithm can be applied when solving a wide variety of problems, ranging from the most common decision problems to global optimization problems, as [10] shows, where finding an input element $x$ satisfying $f(x) < f(y)$ for whatever $y \neq x$ is the goal. This method starts with a random element $y$ and then repeats two steps until no solution is found. The first step means executing Grover's algorithm with $f(x) < f(y)$ as Oracle. The second step updates the value of $y$ with $x$.

Variations in this sort of algorithms range from combining them with heuristics, such as as Variable State Independent Decaying Sum (VSIDS) branching heuristics, or a passing trough focusing on parallelism, with ManySAT solvers [11] using efficient Boolean constraint propagation (BCP) together with an optimised decision strategy [12], to using hardware solvers focused on FPGAs [13].

Other quantum computing approaches used to solve this problem can be seen in [14]. This work encodes each clause as a circuit called filter box, which holds that states satisfying this clause have a higher amplitude than the rest. This is achieved by using ancillary qubits, CNOT gates, rotation gates around $y$-axis, and two non-unitary transformations named boost Hamiltonian and projection Hamiltonian. If a qubit is used for other clauses, that qubit will be teleported to the other one by the circuit teleportation box; otherwise, it will finish with a boost Hamiltonian transformation.

In [15], Leporati et al. propose three different algorithms to deal with the 3-SAT problem. These algorithms are mainly leveraged by quantum parallelism, computing the 3-SAT function for all states at once, and therefore obtaining a linear combination $\alpha|0\rangle + \beta|1\rangle$ such that either $\beta = 0$, when the instance on the problem is unsatisfiable, or $\beta$ measures a proportion of the number of solutions with respect to the domain of Boolean variables when it is satisfiable (which obviously includes the first option). These different proposals are focused on a quantum Fredkin circuit, a register in a quantum register machine, and the energy of a given membrane in a quantum P system. All these proposals assume the ability to discriminate a null vector from a non-zero one.

Other quantum computing approaches to solve 3-SAT problem can be found in [16], where a sort of diabatic quantum annealing (not universal) is proposed, and [17], using Rydberg Atom Graphs based on neutral atoms' quantum computing. Both approaches are supported by quantum systems that evolve mechanically.

Meuli et al. proposed, in [18], a way of synthesising CNOT gates in order to build the Oracle operator within SAT problem-solving.

Cheng et al. proposed, in [19], a sort of cooperative quantum searching in which Boolean variables are split into two sets. In one of them, the values of the variables are set by a local search algorithm, such as, for example, GenSAT; in the other set, Grover's searching algorithm is applied.

The SAT problem has also been faced from a hybrid Classical/Quantum perspective, as Zhang et al. addressed in [20], where an algorithm that first unfolds the branches of the tree of states until reaching a given threshold, from which Grover's algorithm is applied, is presented. Finally, the solutions must be properly linked in order to obtain the solutions of the former SAT problem. Our proposal shares some similarities with this one but, as far as we know, the amount of information we extract from the problem domain, together with the size of the experiment set we provide, makes us quite confident that our proposal is still value, and distinct from that of Zhang et al.

In this work, we focus on DPLL-based algorithms. Specifically, we focus on both the GRASP (Generic seaRch Algorithm for the Satisfiability Problem) [21], based on conflict-driven clause learning and non-chronological backtracking, and miniSAT [22], which is based on VSIDS dynamic variable order and on conflict-driven clause learning.

This paper is structured as follows: Section 2 refreshes some preliminary notions, Section 3 presents our proposal, Sections 4 and 5 are devoted to, first, describing the experimental setup, and then gathering the results to empirically validate our algorithm. Section 5 concludes the paper.

## 2. Preliminaries

### *2.1. SAT Problem*

In this section, we are going to set the notation that is to be used in the rest of the paper. We will consider a numerable $\mathsf{Bool}$ set of Boolean variables; two symbols—$\top$ for the constant true atomic formula and $\bot$ for the constant false atomic formula; the ordinary symbols $\wedge$ and $\vee$ stand for conjunction and disjunction logical operators; the negation of a formula $\varphi$ is denoted by $\overline{\varphi}$.

**Definition 1** (Popositional formula)**.** *The set of propositional formulas is generated by the following Backus–Naur Form BNF form:*

$$\varphi ::= \bot \mid \top \mid p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \overline{\varphi} \tag{1}$$

*where* $p \in \mathsf{Bool}$*.*

*We will denote the set of propositional formulas by* $\mathsf{Prop}$*.*

*A* truth assignment *is a function* $v : \mathsf{Bool} \mapsto \{0, 1\}$*. We can extend* $v$ *to the set of propositional formulas* $\hat{v} : \mathsf{Prop} \mapsto \{0, 1\}$ *in the usual way:*

- $\hat{v}(\bot) = 0$, $\hat{v}(\top) = 1$,
- *if* $p \in \mathsf{Bool}$, $\hat{v}(p) = v(p)$,
- $\hat{v}(\varphi_1 \wedge \varphi_2) = \min\{\hat{v}(\varphi_1), \hat{v}(\varphi_2)\}$, $\hat{v}(\varphi_1 \vee \varphi_2) = \max\{\hat{v}(\varphi_1), \hat{v}(\varphi_2)\}$,
- $\hat{v}(\overline{\varphi}) = (\hat{v}(\varphi) + 1) \mod 2$

*We say that* $v$ *satisfies* $\varphi$ *(written* $v \models \varphi$*) when* $\hat{v}(\varphi) = 1$*; if* $\hat{v}(\varphi) = 0$ *we write* $v \not\models \varphi$*.*

*The vocabulary of a formula* $\phi$*,* $\mathsf{voc}(\varphi)$*, is the set of Boolean variables it contains. Formally,*

$$\begin{array}{ll} \mathsf{voc}(\top) = \mathsf{voc}(\bot) = \varnothing, & \mathsf{voc}(p) = \{p\}, \\ \mathsf{voc}(\overline{\varphi}) = \mathsf{voc}(\varphi), & \mathsf{voc}(\varphi_1 \diamond \varphi_2) = \mathsf{voc}(\varphi_1) \cup \mathsf{voc}(\varphi_2) \quad \diamond \in \{\wedge, \vee\} \end{array} \tag{2}$$

A SAT problem is the problem of determining whether a truth assignment $v$ can be given to the variables of a given formula $\varphi$, such that $v \models \varphi$. If this is the case, the referred formula is said to be *satisfiable*. Otherwise, i.e., when there is no such assignment, the formula is said to be *insatisfiable*.

As SAT belongs to the class of NP-problems, there is no known algorithm capable of solving SAT in sub-exponential time. Nevertheless, we could find algorithms capable of solving quite large instances of the SAT problem by following some heuristics.

We will consider $\mathsf{Bool}$ as a numerable set of Boolean variables.

We are going to consider a restricted version of the SAT problem: k-SAT, k-satisfiability, problem. This consists of determining whether the k-SAT formula is satisfiable. K-SAT is also an NP-problem. Therefore, we then defined a k-SAT formula.

**Definition 2** (K-SAT formulas)**.** *First, define the set of atomic formulas AT*

$$AT = \{\bot, \top\} \cup \mathsf{Bool} \cup \{\overline{p} \mid p \in \mathsf{Bool}\} \tag{3}$$

*We will denote the elements of AT as $x, x_0, x_1, \ldots$. We assume $\overline{\overline{x}} = x$.*

*An* OR clause, *or just a* clause *for us, is a Boolean formula with this shape $C = x_1 \vee \ldots \vee x_n$, where $x_i \in AT$ is an atomic formula. For simplicity, we assume that $x_i \notin \{\top, \bot\}$ and $x_i \neq x_j$ for $i \neq j$. If $n = 0$, we assume that $C = \bot$.*

*A k-SAT formula is a Boolean expression shaped as $S_f = C_1 \wedge \ldots \wedge C_m$, where each $C_i$ is a clause with, at most, k variables. If $m = 0$, we assume that $S_f = \top$.*

Since the logical operators ($\wedge$ and $\vee$ are independent, commutative and associative), we can see a k-SAT formula as a set of clauses (connected by $\wedge$ operators), and a clause a set of atoms (connected by $\vee$ operators).

An important metric for dealing with k-SAT problem is the density. This metric is related to the difficulty associated with determining whether a formula is satisfiable.

**Definition 3** (Density)**.** *The density of $S_f = \bigwedge_{i=1}^{i=m} C_i$ a k-SAT formula is the ratio between the number of clauses (m) and the number of Boolean variables : $d(S_f) = \frac{m}{|\mathsf{voc}(S_f)|}$*

In the related literature, it is assumed that $\forall k(k > 1) \in \mathbb{N}$, and there exists a threshold value for the density of k-SAT formulas $d_k$ such that:

- The farther $d(S_f)$ is from $d_k$, the fewer calls to the DPLL algorithm [6] are required to solve the k-SAT problem.
- If $d(S_f) > d_k$, the formula would be unsatisfiable with high probability, but the opposite occurs when $d(S_f) < d_k$, i.e., a high $d(S_f)$ is usually associated with an unsatisfiable formula.

Some examples are:

- For the 3-SAT problem, Mitchell, Selman and Levesque estimated in 1991 [23] that $d_3 \sim 4.55$ for about 20 variables and $d_3 \sim 4.3$ for larger number of variables

## 2.2. Incremental SAT—Definition and Complexity

We perform an incremental solving approach also followed by [24]. In practice, we are going to build a sequence of k-SAT formulas, which will finish either with the former k-SAT formula $S_f$ when it is satisfiable or before in the other case. Figure 1 provides a graphical view of this process.

$$SAT_0 = \top \subsetneq SAT_1, \ldots SAT_{n-1} \subsetneq SAT_n = S_f \tag{4}$$

where $SAT_{i+1} = SAT_i \cup NC_i$, being $NC_i \subseteq S_f \setminus SAT_i$ a new set of clauses. In each step, we assume that there is $v$, such that $v \models SAT_i$. The new set of clauses $NC_i$ is defined according to the Algorithm 1. If there is no such $NC$, this identifies an unsatisfiability of $S_f$.

In [24], the authors propose a *branch and bound* algorithm that can be used to find the truth assignments in steps with numbers 5–7 within our algorithm. In our case, we will use a Quantum Algorihtm based on the Grover Algorithm to solve the problem. In our case, we propose a novel heuristic (see Section 3.3) to choose the most convenient clauses in each step.

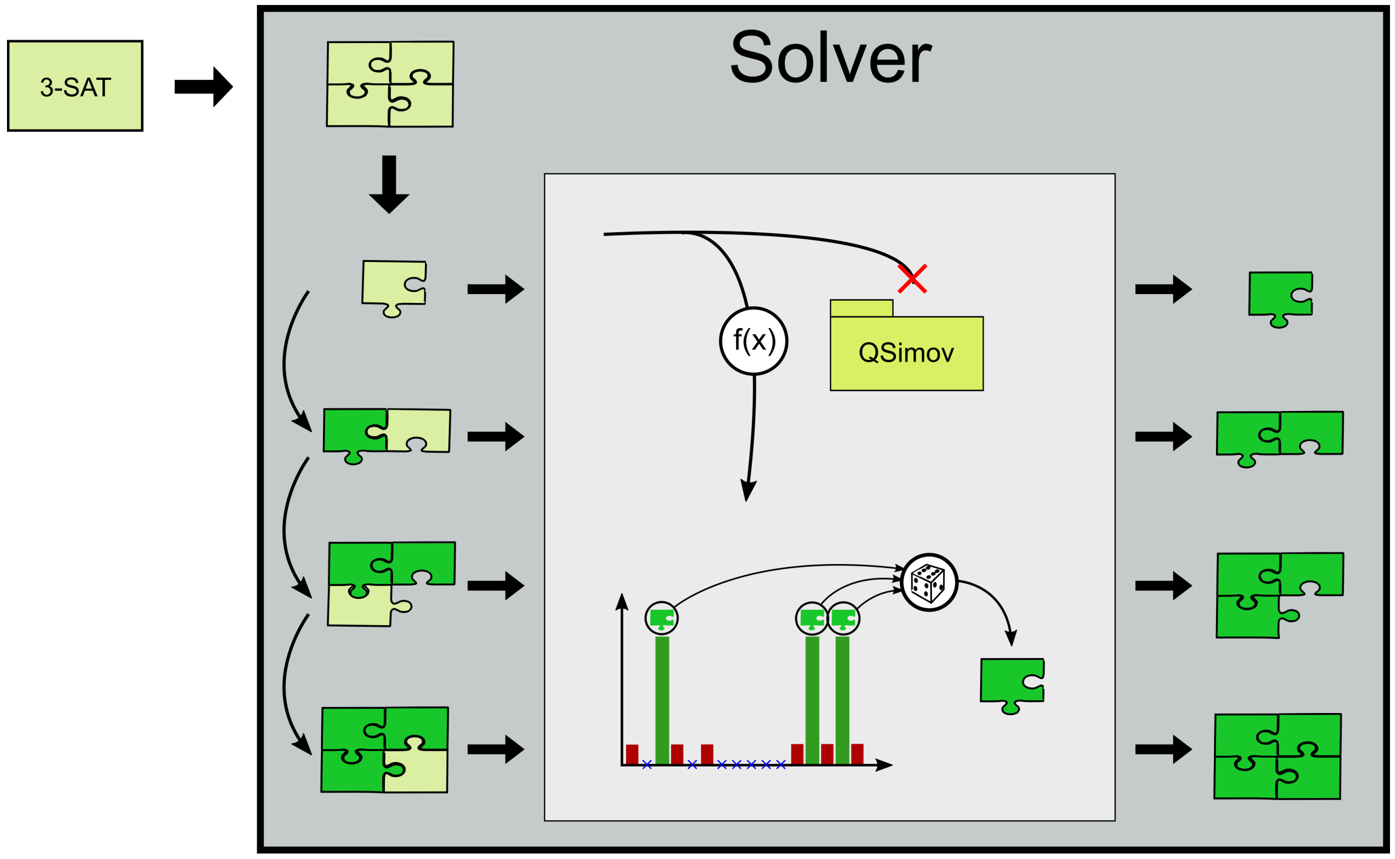


**Figure 1.** Solver diagram.

**Algorithm 1** Algorithm to extend the set of clauses

Find $NC \subseteq S_f \setminus SAT_i$, $SAT_{i+1} = SAT_i \cup NC$ if the following tests hold:

1. If $\{x\}, \{\overline{x}\} \in C$, $NC$ is unsatisfiable, it cannot be considered the new set.
2. $NC$ can be simplified by removing the trivially true clauses, so there is no $C \in NC$ such that $x, \overline{x} \in C$ or $\top \in C$.
3. $NC$ is split into two sets: $NC = NC_1 \wedge NC_2$ where $NC_2$ is the maximal set of clauses of $NC$ such that the variables in $NC_2$ do not appear in $SAT_i$:

$$\mathsf{voc}(NC_1) \cap \mathsf{voc}(SAT_i) \neq \varnothing\,, \qquad \mathsf{voc}(NC_2) \cap \mathsf{voc}(SAT_i) = \varnothing \tag{5}$$

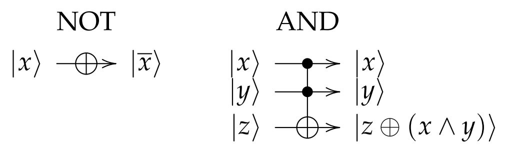
   $SAT_i$ is also split into two sets: $SAT_i = S_1 \wedge S_2$ where $S_2$ is the maximal set of $SAT_i$ whose variables are not contained in $NC_1$:

$$\mathsf{voc}(S_1) \cap \mathsf{voc}(NC_1) \neq \varnothing\,, \qquad \mathsf{voc}(S_2) \cap \mathsf{voc}(NC_1) = \varnothing \tag{6}$$

4. If $v \models NC_1$, go to step 7.
5. Look to $NC_1$ for a truth assignment $v_1$ such that $v_1 \models NC_1$ and $v_1(x) = v(x)$ for all $x \in \mathsf{voc}(S_1)$. If such a truth assignment exists, go to step 7.
6. Look for a truth assignment $v_1$ such that $v_1 \models S_1 \wedge NC_1$. If such a truth assignment does not exist, $NC$ cannot be added to $SAT_i$.
7. Look for a truth $v_2$ assignment for $NC_2$.

## 2.3. Convert SAT to Oracle

To begin with, practice with an oracle is required to test satisfiability; for this reason, we follow the usual [25] rewriting process. First, we eliminate the disjunctions according to De Morgan laws. In this way, a clause such as $C = x_1 \vee x_2 \vee \cdots \vee x_l$ is transformed into the equivalent formula $c' = \overline{\overline{x_1} \wedge \overline{x_2} \wedge \cdots \wedge \overline{x_l}}$, whose negations can easily be implemented in a quantum circuit with the $NOT$ operator. Since the conjunction is not a reversible connector, we need to implement it as an oracle $AND$:



(7)

Therefore, implementing a k-SAT formula, which is a conjunction of $m$ clauses, would require $m + \left|voc\left(\cup_{i=1}^{m} C_i\right)\right| + 1$ (one qubit per variable, one qubit per clause and one extra qubit for the final result).

## 3. Heuristics to Save Space and Computational Complexity

### 3.1. Grover's Algorithm

Grover's algorithm [26] searches for an element in an unordered space within a time belonging to $O(\sqrt{N})$, where $N$ is the size of the search space. More precisely, the input of the Grover Algorithm takes an oracle $U_f$, which implements a Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$ where the elements of the search space are encoded in $\{0,1\}^n$ and, as usual, 1 represents success (goal achieved) and 0 means failure.

One of the drawbacks of this method is the need to know the number of solutions in advance in order to determine the number of rotations to perform. This problem can be dealt by executing Grover algorithm with $1, 2, \ldots$, which can be costly; instead, Monte Carlo algorithms can be used.

### 3.2. Grover's Bounded Algorithm

In this section, we first refer to the corresponding version of Grover's Algorithm [27], based on which we developed our main contribution. This version reduces the work domain to a subset of the original set of states to be searched.

Let us assume that we want to find the solutions of $f(x) = 1$, where $f : \{0,1\}^n \mapsto \{0,1\}$.

We start from the following quantum circuit $U_f$, which stands for the phase oracle of $f(x) = 1$:

$$\left|\overline{\{0,1\}^n}\right\rangle \;\text{—}\boxed{U_f}\longrightarrow \left|\overline{\{0,1\}^n}\right\rangle \tag{8}$$

In a nutshell, our algorithm selects $X_G \subseteq X$ with $X = \{0,1\}^n$ in order to search for the solutions in $X_G$. More precisely, if $S$ was the set of solutions of $f(x) = 1$, our algorithm will find unique solutions to the set $S_G = X_G \cap S$. We denote $\overline{S_G} = X_G \setminus S$ as the complementary of $S_G$ with respect to $X_G$

As previously recalled, general Grover's algorithm starts by setting the system into a full superposed state by applying a Hadamard quantum gate over the former $n$ qubits; nevertheless, we are only going to apply a superposition to states belonging to $X_G$. We call this superposition operator $U_{X_G}$.

$$U_{X_G}|0^n\rangle = \frac{1}{\sqrt{|X_G|}} \cdot \sum_{x \,\in\, X_G} |x\rangle = |\phi\rangle \tag{9}$$

The construction of the $U_{X_G}$ operator

$$|0^n\rangle \;\text{—}\boxed{U_{X_G}}\longrightarrow |\phi\rangle \tag{10}$$

is discussed in Section 3.2 for our specific case. In general, this operator can be performed as described in [28] when no extra qubits are required, and as described in [29] when some extra ancillary qubits are required.

Figure 2 graphically presents how the first operator performs.

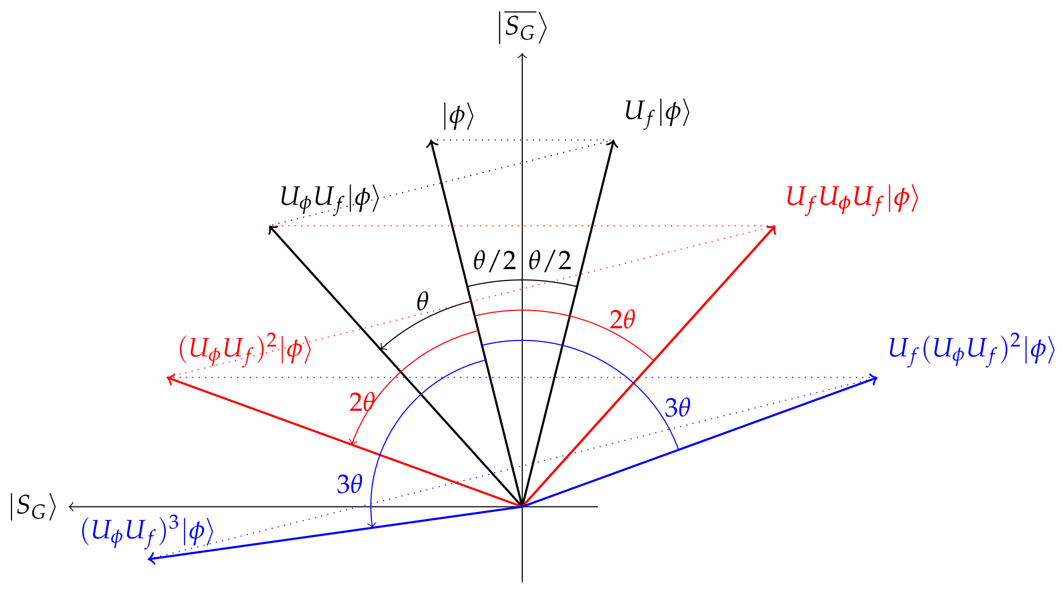


**Figure 2.** Modified Grover Rotations.

Once the previously described configuration is set, the evolution of our algorithm is similar to the original one. As we will explain, it is based on rotations towards the solution vector (see Figure 2). More specifically, we consider two ortonormal vectors: one corresponds to the states that contain a solution and the other corresponds to the states that do not.

$$|S_G\rangle = \frac{1}{\sqrt{|S_G|}} \sum_{x \in S_G} |x\rangle, \qquad |\overline{S_G}\rangle = \frac{1}{\sqrt{|\overline{S_G}|}} \sum_{x \in \overline{S_G}} |x\rangle \tag{11}$$

Then $|\phi\rangle = \sqrt{\frac{|S_G|}{|X_G|}}|S_G\rangle + \sqrt{\frac{|\overline{S_G}|}{|X_G|}}|\overline{S_G}\rangle$ belongs to the vector subspace generated by $|S_G\rangle$ and $|\overline{S_G}\rangle$.

Hence, we obtain

$$\begin{array}{l|l} U_f|\overline{S_G}\rangle = \frac{1}{\sqrt{|\overline{S_G}|}} \sum_{x \in \overline{S_G}} U_f(|x\rangle) = & U_f|S_G\rangle = \frac{1}{\sqrt{|S_G|}} \sum_{x \in S_G} U_f(|x\rangle) = \\ = \frac{1}{\sqrt{|\overline{S_G}|}} \sum_{x \in \overline{S_G}} |x\rangle = |\overline{S_G}\rangle & = \frac{1}{\sqrt{|S_G|}} \sum_{x \in S_G} (-|x\rangle) = \\ & = -\frac{1}{\sqrt{|S_G|}} \sum_{x \in S_G} |x\rangle = -|S_G\rangle \end{array} \tag{12}$$

Therefore,

$$U_f|\phi\rangle = \sqrt{\frac{|\overline{S_G}|}{|X_G|}}|\overline{S_G}\rangle - \sqrt{\frac{|S_G|}{|X_G|}}|S_G\rangle \tag{13}$$

This last equation shows that $U_f$ is symmetrical with respect to the vector $\overline{S_G}$.

The second step to be developed, as in Grover's algorithm, is the inversion about the mean, i.e., a symmetry with respect to the initial vector. In our case, we perform a symmetry with respect to $\phi$ vector. The operator that performs this symmetry is the following:

$$U_\phi = U_{X_G}(2|0^n\rangle\langle 0^n| - I)U_{X_G}^\dagger \tag{14}$$

Let us note that $2|\phi\rangle\langle\phi| - I$ is symmetrical with respect to $|\phi\rangle$ vector.

Let us consider the angle $\theta$, such that $\frac{\theta}{2}$ is the angle between $|\phi\rangle$ and $|\overline{S_G}\rangle$.

The operator $U_\phi U_f$ means a rotation of amplitude $\theta$.

All this means rotating around and amplifying the distance to the vector $\overline{S_G}$ until as close as possible to the hyper-plane $S_G$.

Since

$$|\phi\rangle = \sqrt{\frac{|S_G|}{|X_G|}}|S_G\rangle + \sqrt{\frac{|\overline{S_G}|}{|X_G|}}|\overline{S_G}\rangle = \sin\left(\frac{\theta}{2}\right)|S_G\rangle + \cos\left(\frac{\theta}{2}\right)|\overline{S_G}\rangle \tag{15}$$

We obtain

$$(U_\phi U_f)^k = \sin\left(\frac{2k+1}{2}\theta\right)|S_G\rangle + \cos\left(\frac{2k+1}{2}\theta\right)|\overline{S_G}\rangle \tag{16}$$

This procedure is graphically shown in Figure 2.

Similar to Grover's algorithm, we assume for ours that $|S_G| << |X_G|$, so $\frac{\theta}{2} \approx \sin\left(\frac{\theta}{2}\right) = \sqrt{\frac{|S_G|}{|X_G|}}$. The number of iterations $k$ is determined such that $\frac{2k+1}{2}\theta \approx \frac{\pi}{2}$; therefore, $k \approx \frac{\pi}{4}\sqrt{\frac{|X_G|}{|S_G|}} - \frac{1}{2}$. If we measure the final state, we obtain the probability of the solution states as follows, $\frac{\sin\left(\frac{2k+1}{2}\theta\right)^2}{|S_G|} \approx \frac{1}{|S_G|}$ and the probability of the non-solution states is $\frac{\cos\left(\frac{2k+1}{2}\theta\right)^2}{|\overline{S_G}|} \approx 0$.

Specific Case

Let us consider $F$ a $k - SAT$ formula (or, equivalently, a set of clauses connected by conjunctions) whose vocabulary is the set of Boolean variables in it, i.e., $\text{voc}(F) = \{p_0, p_1, \ldots, p_{n-1}\}$. We are creating an incremental approach, regarding the amount of clauses considered, such that at any step $i$, we consider a subset of clauses $SAT_i \subseteq F$ that is satisfiable. At this point, we have a partial truth assignment $v$, i.e., a pair of sets of indices $I, J \subseteq \{0, 1, \ldots, n-1\}$ such that $\forall i \in I.v(p_i) = 1$ and $\forall j \in J.v(p_j) = 0$. Let $K = \{0, 1, \ldots, n-1\} \setminus (I \cup J)$ represent the complementary set of indices, i.e., the set of indices for which the truth value of $v(p_k)$ is not decided $\forall t \in K$.

$\forall i \in \{0, 1, \ldots, n-1\}$, we define the unitary gate $M_i$ as follows:

$$M_t = \begin{cases} X & (t \in J) \\ I & (t \in I) \\ H & (t \in K) \end{cases} \tag{17}$$

The operator $U_{X_G}$ of Equation (9) in Section 3.2 is defined as follows

$$U_{X_G} = M_{n-1} \otimes M_{n-2} \cdots \otimes M_1 \otimes M_0 \tag{18}$$

where $\otimes$ stands for the tensorial product.

When this operator acts over $|0^n\rangle$, the following outcome is obtained

$$|\phi\rangle = U_{X_G}|0^n\rangle = \frac{1}{\sqrt{2^{|K|}}} \cdot \sum_{\substack{z_i=1,\ i \in I,\\ z_j=0,\ j \in J,\\ z_t \in \{0,1\},\ t \in K}} |z_{n-1} \ldots\ldots z_0\rangle \tag{19}$$

## *3.3. Clauses Heuristics*

We finally define a heuristic to lead the incorporation of clauses to the $SAT_i$ process until they conform with the whole (former) SAT formula, since the algorithm pseudo-coded in 1 does not provide any criteria to develop this "incremental" procedure.

Aiming to save computational costs, we defined a heuristic to detect conflict with the decisions already taken on the variables in either $I$ or in $J$ as quickly as possible, meaning that these unsatisfiable formulas are identified as soon as possible.

We propose an order on the clauses that extends the statistical fact that those atoms whose variables appear less in the former formula would generate a lower number of conflicts, i.e., contradictions.

**Definition 4.** *Let us consider a k-SAT formula $F$, and a Boolean variable $x$; we define* $\mathsf{noc}(x,F)$ *the number of occurrences of $x$ in $F$, and $\overline{\mathsf{noc}}(x,F)$ stands for the number of occurrences of $\overline{x}$ in $F$:*

- *If $C = \{x_0, \ldots, x_{k-1}\}$ is a clause ($C = x_0 \vee \cdots \vee x_{k-1}$)*

$$\mathsf{noc}(x,C) = |\{x_c \mid x_c \in C,\ x = x_c\}|, \quad \overline{\mathsf{noc}}(x,C) = |\{x_c \mid x_c \in C,\ \overline{x} = x_c\}| \tag{20}$$

- *If $F$ is a set of clauses ($F = C_0 \wedge C_1 \wedge \ldots$)*

$$\mathsf{noc}(x,F) = \sum_{C \in F} \mathsf{noc}(s,C), \quad \overline{\mathsf{noc}}(x,F) = \sum_{C \in F} \overline{\mathsf{noc}}(s,C) \tag{21}$$

*Finally, we define $NOC(x,F) = \min(\mathsf{noc}(x,F), \overline{\mathsf{noc}}(x,F))$.*

Intuitively, if $NOC(x,F)$ is low, the probability of conflict coming from $x$ when considering a clause containing $x$ is low. We use this value for each of the variables in a clause in order to grade the tentative conflictivity of clauses to be added to $SAT_i$, therefore leading the previously referenced incremental fashion.

**Definition 5.** *Let $F$ be a k-SAT formula and $C = \{x_0, \ldots x_{k-1}\}$ a clause within $F$. The order function is defined as follows*

$$\mathsf{order}(C,F) = \sum_{i=0}^{k-1} \frac{1}{k^{NOC(x_i,F)}} - \frac{1}{k^{\mathsf{th}(i,F)}} \tag{22}$$

*where $\mathsf{th}(i,F)$ is i-th value of the decreasingly ordered list of the NOC of variables belonging to $F$, i.e., $[NOC(x,F) \mid x \in C,\ C \in F]$.*

Finally, our incremental approach chooses the unitary set $NC = \{C\}$ at any stage, such that the clause $C$ minimizes $\mathsf{order}(C,F)$.

## 4. Testing the Proposal

### *4.1. Testbed*

In order to test the performance of the proposed algorithm, we measured four parameters:

- The first parameter is the maximum number of qubits superposed in the execution of the algorithm. This parameter, in the original Grover's algorithm, is the total number of variables in the problem, whereas in our proposed algorithm it is the maximum number of qubits superposed along all Grover's bounded algorithm circuits that are executed when applying the proposal.
- The second parameter is the maximum number of clauses in Oracle; that is, the maximum number of clauses in Oracle along all Grover's bounded algorithm circuits that are executed.
- The third parameter is the total number of quantum iterations; that is, the total number of requests to Oracle in both Grover's algorithms and in Grover's bounded algorithms.
- The last parameter analysed is the maximum number of qubits used in circuit; this is the maximum number of qubits used along all Grover's bounded algorithm circuits that are executed.

This performance testing can be reproduced by executing the code of the repository [30], typesetting the following commands to generate the test cases (Figure 3):

```
$ python generator.py -v 5 6 7 8 9 10 11 12 13 14 15 16
  --densities 2 3 4 5 -k 3 -e 10 --seed 1
```

where:

- **-v**: number of variables for the test case (ranging from 5 to 16).
- **--densities**: densities for each test case (ranging from 2 to 5).
- **-k**: number of variables per clause (3-SAT).
- **-e**: number of examples per test case (below can be found more detailed).
- **--seed**: seed governing the stochastic behaviour in the process (1 in the proposed case).

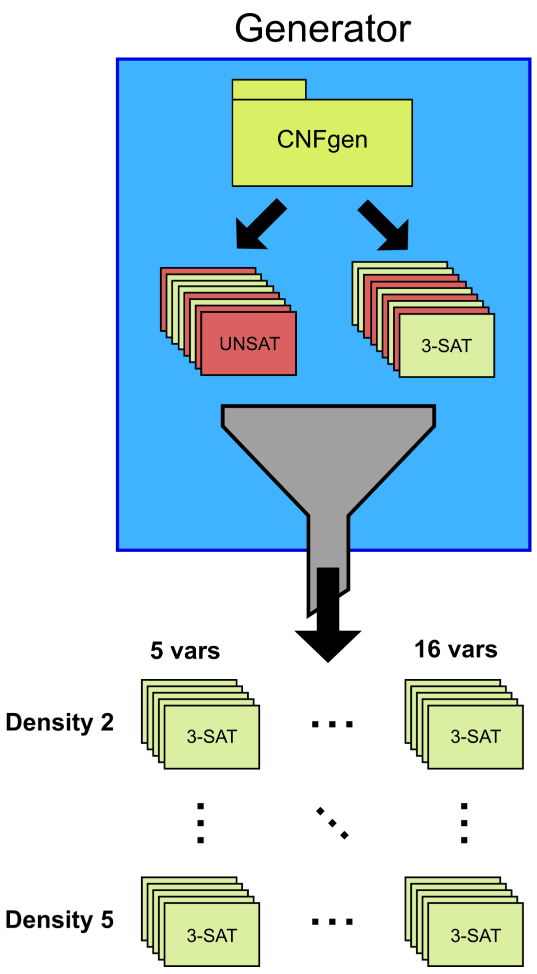


**Figure 3.** Test cases generator diagram.

Additionally, this one can be used to execute our algorithm:

```
$ python main.py
```

More on detail:

1. A test case for us is a pair (density, number of variables). Ten examples per each test case are generated.
2. The process of generating test cases was supported by **CNFgen** [31]. In particular, 10 3-SAT problems were generated for each pair (density, number of variables): in total, $4 \times 12$ different test cases, which compute 480 3-SAT problems. The density values range from 2 to 5 (both included). The number of variables ranges from 5 to 16 since, below 5, this incremental scheme is not needed because computing requirements are affordable enough and, over 16, it is more computationally expensive which, as evidence was found that the advantage we provide could be made smaller, meant that we did not consider using bigger values.

3. Each of the 480 3-SAT problems was executed in each of the three scenarios to be compared and, for random measuring issues, five different seeds were also involved; therefore, in the end, $5 \cdot 10 \cdot 4 \cdot 12$ different executions were performed over each of the three scenarios under comparison.
4. The experimental set up was powered by Qsimov Quantum Simulator [32]. Unfortunately, the space computational complexity requirements prevented us from running examples with more than five variables; therefore, for the remaining cases we computed the probability amplitudes of the states instead of simulating the whole quantum computing process, according to the well-known formulas in [33]:

$$\alpha_{t+1} = \alpha_t \cdot \left(1 - \frac{2 \cdot k}{N}\right) + \beta_t \cdot \left(2 - \frac{2 \cdot k}{N}\right) \tag{23}$$

$$\beta_{t+1} = -\alpha_t \cdot \frac{2 \cdot k}{N} + \beta_t \cdot \left(1 - \frac{2 \cdot k}{N}\right) \tag{24}$$

   where $t$ is the number of iterations, $k$ is the number of solutions and $N$ is the total number of states. Afterwards, we randomly (five previously referenced seeds) chose the final quantum state among the obtained solutions.
5. The algorithm finishes when there are no more clauses to be added before the former SAT problem is reached.

We decided to start from scratch regarding the definition of function $v : \mathsf{Bool} \mapsto \{0, 1\}$, i.e., from an empty truth assignment over the set of Boolean variables. This is because once we developed multiple experiments considering a variety of partial definitions of $v$ to start from (either considering random chosen values or values coming from a frequency data analysis of variables in the SAT formula), the results we obtained did not improve on those we have chosen; therefore, for the sake of simplicity, this is our decision.

Some other non-relevant assumptions we made are as follows:

- We know the number of solutions needed to determine the number of iterations in each execution in advance for both original and modified Grover's algorithms.
- The result of the measurement is always one of the maximum probability states. In both cases, we have to repeat the algorithm to obtain a probability distribution, so we simplified this step.
- Each 3-SAT problem has at least one solution because no version of Grover's algorithm works for problems with no solution.

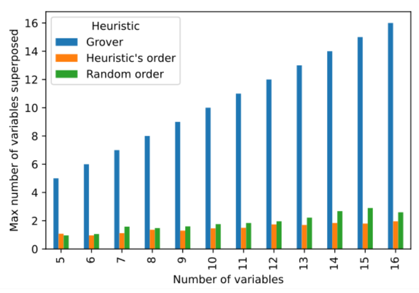
To finish with, the scenarios we considered are as follows:

1. Grover's algorithm.
2. New approach algorithm that is randomly ordered.
3. New approach algorithm with heuristic order.

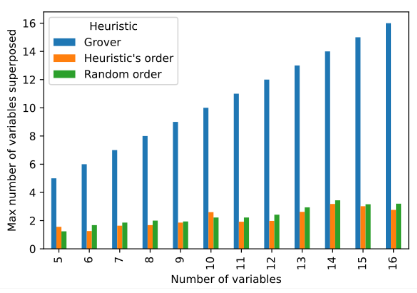
## 4.2. Results

Once the experiments were performed, we summarised the outcomes as follows:

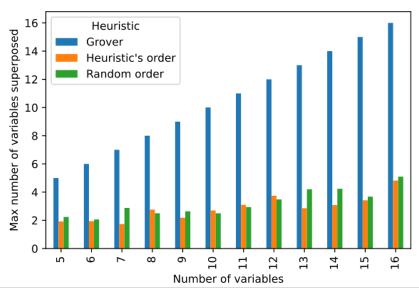
As the number of qubits on superposition is a key issue from the performance perspective, the following four graphs show a very noticeable improvement almost regardless of the density of the 3-SAT formulas under consideration. This advantage was maintained in cases in which there was no predefined order to the clauses added to our *incremental* scheme, although it was not as big as that with respect to the former Grover's algorithm (Figure 4).



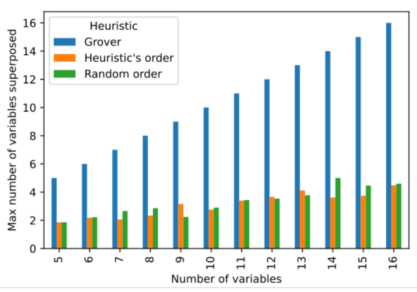
(a) #(superposed qubits) for 3-SAT with density 2



(b) #(superposed qubits) for 3-SAT with density 3



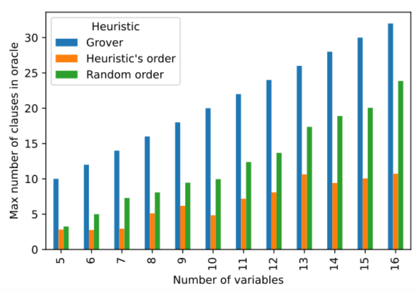
(c) #(superposed qubits) for 3-SAT with density 4



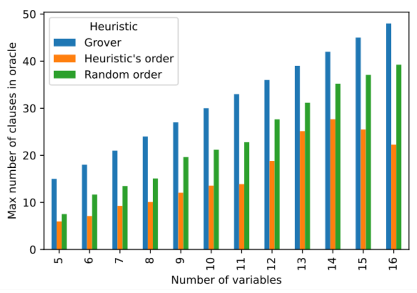
(d) #(superposed qubits) for 3-SAT with density 5

**Figure 4.** Comparisons of number of superposed qubits for 3-SAT formulas according to densities.

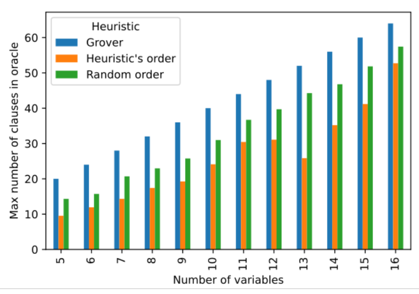
Another very important part of the quantum algorithm is performing the oracle, where the amount of clauses directly affects the number of ancillary qubits required; therefore, the next four graphs are devoted to measuring this fact, again using the densities of 3-SAT formulas. The results obtained clearly show the advantage of the proposed algorithm (Figure 5):



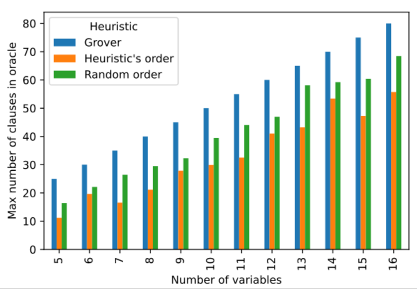
(a) #(clauses in oracle) for 3-SAT with density 2



(b) #(clauses in oracle) for 3-SAT with density 3



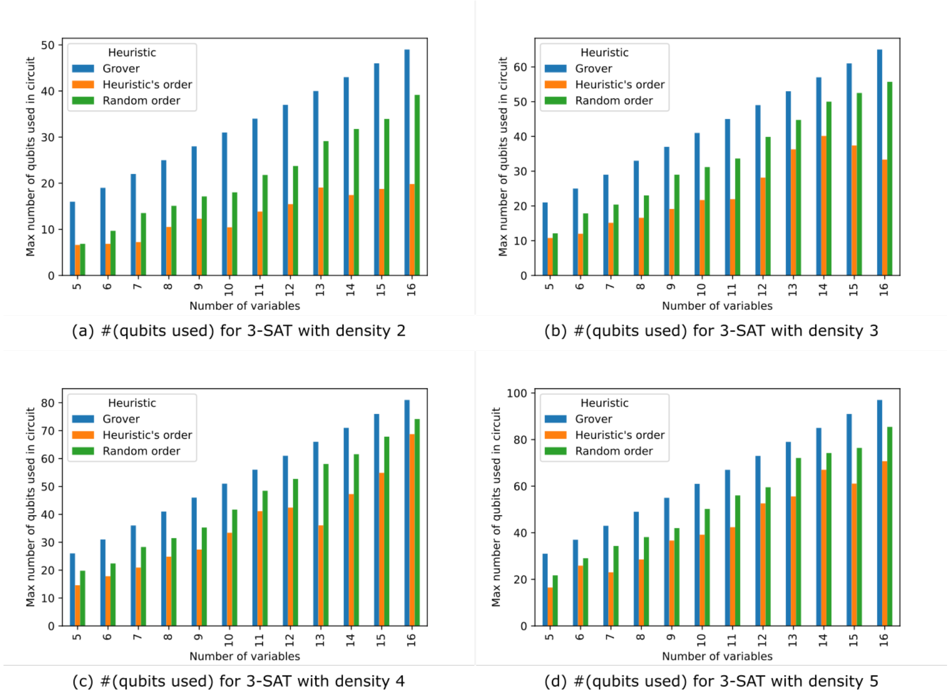
(c) #(clauses in oracle) for 3-SAT with density 4



(d) #(clauses in oracle) for 3-SAT with density 5

**Figure 5.** Comparisons of number of clauses in oracle according to densities of 3-SAT formulas.

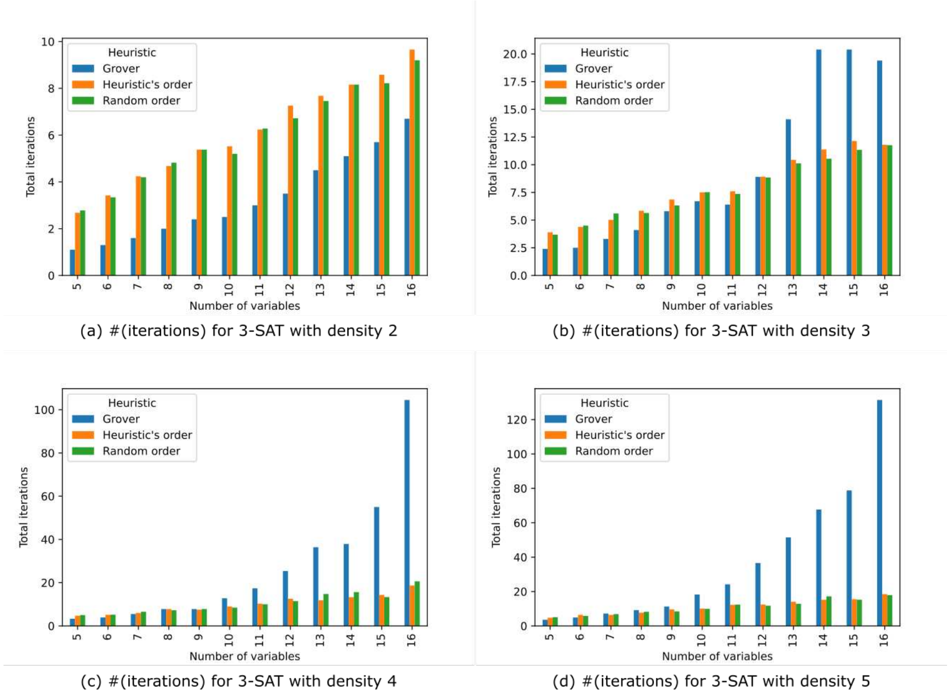
Finally, we conducted a space computational complexity analysis; in the following four graphs, wepresent the results obtained when measuring the upper bound for the number of qubits used along the whole of the proposed algorithm. In our opinion, these are the most representative graphs showing the improvement provided by our proposal (Figure 6).



(a) #(qubits used) for 3-SAT with density 2 (b) #(qubits used) for 3-SAT with density 3 (c) #(qubits used) for 3-SAT with density 4 (d) #(qubits used) for 3-SAT with density 5

**Figure 6.** Comparisons of total number of qubits used stratified by the densities of 3-SAT formulas.

The next four raphs are devoted to capturing issues within the temporal computational complexity. In particular, we measured the number of Grover's iterations. The following results show that our algorithm requires both a high number of variables as well as high-density formulas in order to improve the results of the original Grover's algorithm (Figure 7).



(a) #(iterations) for 3-SAT with density 2 (b) #(iterations) for 3-SAT with density 3 (c) #(iterations) for 3-SAT with density 4 (d) #(iterations) for 3-SAT with density 5

**Figure 7.** Comparisons of number of Grover's iterations for different densities on 3-SAT formulas.

The results provided by our experiments show that, although the space computational complexity of the proposed algorithm is the same than that of Grover´s algorithm, in practice, the number of qubits needed to perform the computations is reduced in a factor that can range between 1.5 and 2.2. This fact, in practice, represents a noticeable improvement since the main drawback to dealing with quantum computing is the number of qubits that are required.

Going beyond this, we found that the extra auxiliary qubits required for each clause under consideration is reduced at least as much as the total number of qubits.

Finally, the biggest advantage pertains to the number of superposed qubits which, as the graphs show, is really important. This last feature is of paramount importance in quantum computing, whether actual or simulated.

Appendix A provides all the raw numbers we used to create the previous 16 maps that summarize the performance in the comparative study we developed. Green numbers highlight advantages, while red numbers highlight disadvantages compared to Grover´s algorithm.

## 5. Conclusions and Further Work

We dealt with the classic problem of deciding whether any truth assignments exist for a Boolean formula. This is a widely studied problem in the related literature in Maths, Physics, Engineering, etc. From the quantum computing perspective, Grover's algorithm has proven to be a useful tool.

We presented an algorithm which, inspired by Grover´s one, has improved the amount of quantum computational resources by a significant ratio. For this algorithm to be designed, we described a sort of incremental strategy over the sets of clauses that conforms to the 3-SAT formula under scope. The aim of this process is, on the one hand, to use the advantages of Grover´s modified algorithm and, on the other hand, to choose an appropriate conforming order (within this incremental scheme) such that the assumptions required for Grover's modified algorithm to be used are as useful as possible. This last issue led us to define an heuristics for the set of clauses to be added to the previously described incremental scheme.

This algorithm solves 3-SAT problems with significantly less quantum computational resources, namely qubits, and fewer iterations than those required for former Grover's algorithm. As the ordinary bottleneck at quantum computing is strongly dependant on the total number of qubits (especially those that are entangled) that are required, this algorithm extends the scope of 3-SAT problems to be solved given a quantum computing machine.

As future work, we plan to further elaborate the heuristics and the process of incorporating clauses within our incremental scheme, mainly by incorporating more than one clause at each step, perhaps under a dynamic criterion.

**Author Contributions:** Individual contributions are as follows. Conceptualization, J.J.P., L.F.L. and F.L.P.; Formal analysis, J.J.P., L.F.L. and F.L.P.; Investigation, J.J.P., L.F.L., H.I.C., M.M., F.C. and F.L.P.; Writing—original draft, J.J.P., L.F.L., H.I.C., M.M., F.C. and F.L.P.; Writing—review and editing, J.J.P., L.F.L. and F.L.P. All authors have read and agreed to the published version of the manuscript

**Funding:** This work has been supported by the Spanish MINECO/FEDER project AwESOMe (PID2021-122215NB-C31), the Region of Madrid project FORTE-CM (S2018/TCS-4314) co-funded by EIE Funds of the European Union and the Qsimov Quantum Computing project 'Ampliación de la plataforma QSIMOV aumentando su versatilidad y conectividad' (220426UCTR).

**Data Availability Statement:** This performance testing study can be fully reproduced (data included) by executing the code of the repository [30].

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A

**Table A1.** 5 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 5.0 | 0.0% | 10.0 | 0.0% |
| 2 | 2 | 0.78 | −4.4% | 3.88 | −61.2% |
| 3 | 2 | 0.76 | −84.8% | 4.46 | −55.4% |
| 1 | 3 | 5.0 | 0.0% | 15.0 | 0.0% |
| 2 | 3 | 1.14 | −77.2% | 8.48 | −43.47% |
| 3 | 3 | 1.48 | −70.4% | 9.88 | −34.13% |
| 1 | 4 | 5.0 | 0.0% | 20.0 | 0.0% |
| 2 | 4 | 2.22 | −55.6% | 14.9 | −25.5% |
| 3 | 4 | 1.84 | −63.2% | 12.78 | −36.1% |
| 1 | 5 | 5.0 | 0.0% | 25.0 | 0.0% |
| 2 | 5 | 1.82 | −63.6% | 15.36 | −38.56% |
| 3 | 5 | 1.68 | −66.4% | 13.14 | −47.44% |
| 1 | 2 | 1.1 | 0.0% | 16.0 | 0.0% |
| 2 | 2 | 1.92 | +74.55% | 7.28 | −54.5% |
| 3 | 2 | 1.98 | +80.0% | 7.94 | −50.37% |
| 1 | 3 | 2.4 | 0.0% | 21.0 | 0.0% |
| 2 | 3 | 2.88 | +20.0% | 13.16 | −37.33% |
| 3 | 3 | 3.14 | +30.83% | 15.28 | −27.24% |
| 1 | 4 | 3.3 | 0.0% | 26.0 | 0.0% |
| 2 | 4 | 4.12 | +24.85% | 20.66 | −20.54% |
| 3 | 4 | 4.18 | +26.67% | 18.24 | −29.85% |
| 1 | 5 | 3.6 | 0.0% | 31.0 | 0.0% |
| 2 | 5 | 3.94 | +9.44% | 20.64 | −33.42% |
| 3 | 5 | 3.7 | +2.78% | 18.78 | −39.42% |

**Table A2.** 6 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 6.0 | 0.0% | 12.0 | 0.0% |
| 2 | 2 | 0.85 | −85.83% | 6.55 | −45.42% |
| 3 | 2 | 0.8 | −86.67% | 5.65 | −52.92% |
| 1 | 3 | 6.0 | 0.0% | 18.0 | 0.0% |
| 2 | 3 | 1.38 | −77.0% | 12.4 | −31.11% |
| 3 | 3 | 1.05 | −82.5% | 9.75 | −45.83% |
| 1 | 4 | 6.0 | 0.0% | 24.0 | 0.0% |
| 2 | 4 | 2.05 | −65.83% | 17.3 | −27.92% |
| 3 | 4 | 1.9 | −68.33% | 14.05 | −41.46% |
| 1 | 5 | 6.0 | 0.0% | 30.0 | 0.0% |
| 2 | 5 | 1.85 | −69.17% | 21.55 | −28.17% |
| 3 | 5 | 2.2 | −63.33% | 21.05 | −29.83% |

**Table A2.** *Cont.*

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 1.3 | 0.0% | 19.0 | 0.0% |
| 2 | 2 | 2.6 | +100.0% | 11.85 | −37.63% |
| 3 | 2 | 2.2 | +69.23% | 10.5 | −44.74% |
| 1 | 3 | 2.5 | 0.0% | 25.0 | 0.0% |
| 2 | 3 | 3.3 | +32.0% | 18.3 | −26.8% |
| 3 | 3 | 3.05 | +22.0% | 15.0 | −40.0% |
| 1 | 4 | 3.9 | 0.0% | 31.0 | 0.0% |
| 2 | 4 | 4.7 | +20.51% | 24.3 | −21.61% |
| 3 | 4 | 3.8 | −2.56% | 20.35 | −34.35% |
| 1 | 5 | 4.9 | 0.0% | 37.0 | 0.0% |
| 2 | 5 | 4.15 | −15.31% | 28.55 | −22.84% |
| 3 | 5 | 4.95 | +1.02% | 27.45 | −25.81% |

**Table A3.** 7 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 7.0 | 0.0% | 14.0 | 0.0% |
| 2 | 2 | 0.92 | −86.86% | 7.62 | −45.57% |
| 3 | 2 | 0.82 | −88.29% | 6.36 | −54.57% |
| 1 | 3 | 7.0 | 0.0% | 21.0 | 0.0% |
| 2 | 3 | 1.58 | −77.43% | 14.76 | −29.71% |
| 3 | 3 | 1.42 | −79.71% | 16.16 | −23.05% |
| 1 | 4 | 7.0 | 0.0% | 28.0 | 0.0% |
| 2 | 4 | 2.36 | −66.29% | 21.42 | −23.5% |
| 3 | 4 | 1.52 | −78.29% | 20.28 | −27.57% |
| 1 | 5 | 7.0 | 0.0% | 35.0 | 0.0% |
| 2 | 5 | 2.78 | −60.29% | 27.7 | −20.86% |
| 3 | 5 | 1.86 | −73.43% | 26.18 | −25.2% |
| 1 | 2 | 1.6 | 0.0% | 22.0 | 0.0% |
| 2 | 2 | 2.9 | +81.25% | 14.02 | −36.27% |
| 3 | 2 | 2.32 | +45.0% | 11.52 | −47.64% |
| 1 | 3 | 3.3 | 0.0% | 29.0 | 0.0% |
| 2 | 3 | 4.16 | +26.06% | 21.98 | −24.21% |
| 3 | 3 | 4.04 | +22.42% | 23.76 | −18.07% |
| 1 | 4 | 5.5 | 0.0% | 36.0 | 0.0% |
| 2 | 4 | 5.72 | +4.0% | 29.36 | −18.44% |
| 3 | 4 | 4.44 | −19.27% | 27.92 | −22.44% |
| 1 | 5 | 7.2 | 0.0% | 43.0 | 0.0% |
| 2 | 5 | 5.4 | −25.0% | 35.7 | −16.98% |
| 3 | 5 | 5.58 | −22.5% | 33.76 | −21.49% |

**Table A4.** 8 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 8.0 | 0.0% | 16.0 | 0.0% |
| 2 | 2 | 0.96 | −88.0% | 8.3 | −48.12% |
| 3 | 2 | 1.22 | −84.75% | 11.68 | −27.0% |
| 1 | 3 | 8.0 | 0.0% | 24.0 | 0.0% |
| 2 | 3 | 1.76 | −78.0% | 18.86 | −21.42% |
| 3 | 3 | 1.74 | −78.25% | 18.08 | −24.67% |
| 1 | 4 | 8.0 | 0.0% | 32.0 | 0.0% |
| 2 | 4 | 2.16 | −73.0% | 21.66 | −32.31% |
| 3 | 4 | 2.46 | −69.25% | 23.52 | −26.5% |
| 1 | 5 | 8.0 | 0.0% | 40.0 | 0.0% |
| 2 | 5 | 2.44 | −69.5% | 30.8 | −23.0% |
| 3 | 5 | 2.32 | −71.0% | 26.78 | −33.05% |
| 1 | 2 | 2.0 | 0.0% | 25.0 | 0.0% |
| 2 | 2 | 3.04 | +52.0% | 15.76 | −36.96% |
| 3 | 2 | 3.32 | +66.0% | 19.72 | −21.12% |
| 1 | 3 | 4.1 | 0.0% | 33.0 | 0.0% |
| 2 | 3 | 4.66 | +13.66% | 27.54 | −16.55% |
| 3 | 3 | 4.58 | +11.71% | 26.5 | −19.7% |
| 1 | 4 | 7.8 | 0.0% | 41.0 | 0.0% |
| 2 | 4 | 5.06 | −35.13% | 30.3 | −26.1% |
| 3 | 4 | 5.44 | −30.26% | 31.9 | −22.2% |
| 1 | 5 | 9.2 | 0.0% | 49.0 | 0.0% |
| 2 | 5 | 6.36 | −30.87% | 39.8 | −18.78% |
| 3 | 5 | 6.52 | −29.13% | 35.02 | −28.53% |

**Table A5.** 9 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 9.0 | 0.0% | 18.0 | 0.0% |
| 2 | 2 | 1.0 | −88.89% | 12.7 | −29.44% |
| 3 | 2 | 0.95 | −89.44% | 9.65 | −46.39% |
| 1 | 3 | 9.0 | 0.0% | 27.0 | 0.0% |
| 2 | 3 | 1.92 | −78.67% | 22.72 | −15.85% |
| 3 | 3 | 1.6 | −82.22% | 18.55 | −31.3% |
| 1 | 4 | 9.0 | 0.0% | 36.0 | 0.0% |
| 2 | 4 | 2.25 | −75.0% | 26.78 | −25.61% |
| 3 | 4 | 1.65 | −81.67% | 26.75 | −25.69% |
| 1 | 5 | 9.0 | 0.0% | 45.0 | 0.0% |
| 2 | 5 | 2.22 | −75.33% | 34.72 | −22.84% |
| 3 | 5 | 2.05 | −77.22% | 31.25 | −30.56% |
| 1 | 2 | 2.4 | 0.0% | 28.0 | 0.0% |
| 2 | 2 | 3.98 | +65.83% | 21.5 | −23.21% |
| 3 | 2 | 3.35 | +39.58% | 17.2 | −38.57% |

**Table A5.** *Cont.*

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 3 | 5.8 | 0.0% | 37.0 | 0.0% |
| 2 | 3 | 4.97 | −14.31% | 32.7 | −11.62% |
| 3 | 3 | 5.15 | −11.21% | 27.1 | −26.76% |
| 1 | 4 | 7.8 | 0.0% | 46.0 | 0.0% |
| 2 | 4 | 6.28 | −19.49% | 36.22 | −21.26% |
| 3 | 4 | 4.5 | −42.31% | 35.9 | −21.96% |
| 1 | 5 | 11.3 | 0.0% | 55.0 | 0.0% |
| 2 | 5 | 7.4 | −34.51% | 44.72 | −18.69% |
| 3 | 5 | 5.95 | −47.35% | 40.55 | −26.27% |

**Table A6.** 10 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 10.0 | 0.0% | 20.0 | 0.0% |
| 2 | 2 | 0.85 | −91.5% | 12.3 | −38.5% |
| 3 | 2 | 0.92 | −90.8% | 8.6 | −57.0% |
| 1 | 3 | 10.0 | 0.0% | 30.0 | 0.0% |
| 2 | 3 | 2.17 | −78.3% | 23.15 | −22.83% |
| 3 | 3 | 1.8 | −82.0% | 20.1 | −33.0% |
| 1 | 4 | 10.0 | 0.0% | 40.0 | 0.0% |
| 2 | 4 | 2.15 | −78.5% | 31.6 | −21.0% |
| 3 | 4 | 2.2 | −78.0% | 27.15 | −32.12% |
| 1 | 5 | 10.0 | 0.0% | 50.0 | 0.0% |
| 2 | 5 | 2.9 | −71.0% | 40.78 | −18.44% |
| 3 | 5 | 2.55 | −74.5% | 36.85 | −26.3% |
| 1 | 2 | 2.5 | 0.0% | 31.0 | 0.0% |
| 2 | 2 | 3.75 | +50.0% | 21.0 | −32.26% |
| 3 | 2 | 2.92 | +16.8% | 15.85 | −48.87% |
| 1 | 3 | 6.7 | 0.0% | 41.0 | 0.0% |
| 2 | 3 | 5.18 | −22.69% | 33.5 | −18.29% |
| 3 | 3 | 5.15 | −23.13% | 29.75 | −27.44% |
| 1 | 4 | 12.8 | 0.0% | 51.0 | 0.0% |
| 2 | 4 | 6.25 | −51.17% | 42.6 | −16.47% |
| 3 | 4 | 5.75 | −55.08% | 36.9 | −27.65% |
| 1 | 5 | 18.3 | 0.0% | 61.0 | 0.0% |
| 2 | 5 | 8.07 | −55.9% | 51.78 | −15.11% |
| 3 | 5 | 7.45 | −59.29% | 47.35 | −22.38% |

**Table A7.** 11 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 11.0 | 0.0% | 22.0 | 0.0% |
| 2 | 2 | 1.0 | −90.91% | 13.85 | −37.05% |
| 3 | 2 | 1.0 | −90.91% | 12.15 | −44.77% |
| 1 | 3 | 11.0 | 0.0% | 33.0 | 0.0% |
| 2 | 3 | 1.4 | −87.27% | 24.65 | −25.3% |
| 3 | 3 | 1.75 | −84.09% | 21.55 | −34.7% |
| 1 | 4 | 11.0 | 0.0% | 44.0 | 0.0% |
| 2 | 4 | 2.88 | −73.82% | 38.38 | −12.77% |
| 3 | 4 | 2.45 | −77.73% | 34.5 | −21.59% |
| 1 | 5 | 11.0 | 0.0% | 55.0 | 0.0% |
| 2 | 5 | 3.48 | −68.36% | 44.38 | −19.31% |
| 3 | 5 | 3.15 | −71.36% | 44.25 | −19.55% |
| 1 | 2 | 3.0 | 0.0% | 34.0 | 0.0% |
| 2 | 2 | 3.68 | +22.67% | 24.22 | −28.76% |
| 3 | 2 | 3.1 | +3.33% | 21.25 | −37.5% |
| 1 | 3 | 6.4 | 0.0% | 45.0 | 0.0% |
| 2 | 3 | 5.42 | −15.31% | 36.48 | −18.93% |
| 3 | 3 | 4.4 | −31.25% | 31.95 | −29.0% |
| 1 | 4 | 17.4 | 0.0% | 56.0 | 0.0% |
| 2 | 4 | 8.28 | −52.41% | 50.38 | −10.04% |
| 3 | 4 | 7.1 | −59.2% | 46.15 | −17.59% |
| 1 | 5 | 24.2 | 0.0% | 67.0 | 0.0% |
| 2 | 5 | 10.15 | −58.06% | 56.38 | −15.85% |
| 3 | 5 | 8.85 | −63.43% | 55.75 | −16.79% |

**Table A8.** 12 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 12.0 | 0.0% | 24.0 | 0.0% |
| 2 | 2 | 1.3 | −89.17% | 17.23 | −28.21% |
| 3 | 2 | 1.15 | −90.42% | 14.28 | −40.5% |
| 1 | 3 | 12.0 | 0.0% | 36.0 | 0.0% |
| 2 | 3 | 1.75 | −85.42% | 27.0 | −25.0% |
| 3 | 3 | 1.4 | −88.33% | 28.1 | −21.94% |
| 1 | 4 | 12.0 | 0.0% | 48.0 | 0.0% |
| 2 | 4 | 3.2 | −73.33% | 41.22 | −14.12% |
| 3 | 4 | 3.05 | −74.58% | 39.4 | −17.92% |
| 1 | 5 | 12.0 | 0.0% | 60.0 | 0.0% |
| 2 | 5 | 3.95 | −67.08% | 50.15 | −16.42% |
| 3 | 5 | 3.45 | −71.25% | 45.2 | −24.67% |

**Table A8.** *Cont.*

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 3.5 | 0.0% | 37.0 | 0.0% |
| 2 | 2 | 5.15 | +47.14% | 29.08 | −21.41% |
| 3 | 2 | 4.38 | +25.14% | 24.25 | −34.46% |
| 1 | 3 | 8.9 | 0.0% | 49.0 | 0.0% |
| 2 | 3 | 5.95 | −33.15% | 39.55 | −19.29% |
| 3 | 3 | 5.3 | −40.45% | 40.5 | −17.35% |
| 1 | 4 | 25.4 | 0.0% | 61.0 | 0.0% |
| 2 | 4 | 8.98 | −64.65% | 54.22 | −11.11% |
| 3 | 4 | 8.65 | −65.94% | 52.0 | −14.75% |
| 1 | 5 | 36.6 | 0.0% | 73.0 | 0.0% |
| 2 | 5 | 9.98 | −72.73% | 63.15 | −13.49% |
| 3 | 5 | 10.3 | −71.86% | 57.3 | −21.51% |

**Table A9.** 13 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 13.0 | 0.0% | 26.0 | 0.0% |
| 2 | 2 | 1.14 | −91.23% | 19.42 | −25.31% |
| 3 | 2 | 1.02 | −92.15% | 15.48 | −40.46% |
| 1 | 3 | 13.0 | 0.0% | 39.0 | 0.0% |
| 2 | 3 | 2.2 | −83.08% | 34.36 | −11.9% |
| 3 | 3 | 2.04 | −84.31% | 31.44 | −19.38% |
| 1 | 4 | 13.0 | 0.0% | 52.0 | 0.0% |
| 2 | 4 | 3.72 | −71.38% | 45.94 | −11.65% |
| 3 | 4 | 2.42 | −81.38% | 34.82 | −33.04% |
| 1 | 5 | 13.0 | 0.0% | 65.0 | 0.0% |
| 2 | 5 | 3.3 | −74.62% | 57.42 | −11.66% |
| 3 | 5 | 3.42 | −73.69% | 56.32 | −13.35% |
| 1 | 2 | 4.5 | 0.0% | 40.0 | 0.0% |
| 2 | 2 | 3.92 | −12.89% | 31.96 | −20.1% |
| 3 | 2 | 3.58 | −20.44% | 26.3 | −34.25% |
| 1 | 3 | 14.1 | 0.0% | 53.0 | 0.0% |
| 2 | 3 | 7.04 | −50.07% | 48.16 | −9.13% |
| 3 | 3 | 5.86 | −58.44% | 44.24 | −16.53% |
| 1 | 4 | 36.4 | 0.0% | 66.0 | 0.0% |
| 2 | 4 | 12.32 | −66.15% | 59.94 | −9.18% |
| 3 | 4 | 7.34 | −79.84% | 47.16 | −28.55% |
| 1 | 5 | 51.5 | 0.0% | 79.0 | 0.0% |
| 2 | 5 | 11.12 | −78.41% | 71.42 | −9.59% |
| 3 | 5 | 8.6 | −83.3% | 70.24 | −11.09% |

**Table A10.** 14 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 14.0 | 0.0% | 28.0 | 0.0% |
| 2 | 2 | 1.24 | −91.14% | 18.88 | −32.57% |
| 3 | 2 | 1.24 | −91.14% | 17.86 | −36.21% |
| 1 | 3 | 14.0 | 0.0% | 42.0 | 0.0% |
| 2 | 3 | 2.22 | −84.14% | 36.32 | −13.52% |
| 3 | 3 | 2.16 | −84.57% | 35.72 | −14.95% |
| 1 | 4 | 14.0 | 0.0% | 56.0 | 0.0% |
| 2 | 4 | 3.02 | −78.43% | 46.88 | −16.29% |
| 3 | 4 | 3.05 | −78.21% | 46.0 | −17.86% |
| 1 | 5 | 14.0 | 0.0% | 70.0 | 0.0% |
| 2 | 5 | 5.25 | −62.5% | 58.82 | −15.97% |
| 3 | 5 | 3.45 | −75.36% | 59.55 | −14.93% |
| 1 | 2 | 5.1 | 0.0% | 43.0 | 0.0% |
| 2 | 2 | 4.7 | −7.84% | 32.32 | −24.84% |
| 3 | 2 | 4.36 | −14.51% | 29.8 | −30.7% |
| 1 | 3 | 20.4 | 0.0% | 57.0 | 0.0% |
| 2 | 3 | 7.7 | −62.25% | 51.1 | −10.35% |
| 3 | 3 | 7.14 | −65.0% | 50.02 | −12.25% |
| 1 | 4 | 37.9 | 0.0% | 71.0 | 0.0% |
| 2 | 4 | 9.1 | −75.99% | 61.85 | −12.89% |
| 3 | 4 | 9.5 | −74.93% | 60.5 | −14.79% |
| 1 | 5 | 67.7 | 0.0% | 85.0 | 0.0% |
| 2 | 5 | 17.02 | −74.86% | 73.82 | −13.15% |
| 3 | 5 | 11.08 | −83.63% | 74.0 | −12.94% |

**Table A11.** 15 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 15.0 | 0.0% | 30.0 | 0.0% |
| 2 | 2 | 1.35 | −91.0% | 22.95 | −23.5% |
| 3 | 2 | 1.1 | −92.67% | 21.62 | −27.93% |
| 1 | 3 | 15.0 | 0.0% | 45.0 | 0.0% |
| 2 | 3 | 2.12 | −85.87% | 36.58 | −18.71% |
| 3 | 3 | 2.28 | −84.8% | 35.65 | −20.78% |
| 1 | 4 | 15.0 | 0.0% | 60.0 | 0.0% |
| 2 | 4 | 3.12 | −79.2% | 52.75 | −12.08% |
| 3 | 4 | 2.75 | −81.67% | 52.65 | −12.25% |
| 1 | 5 | 15.0 | 0.0% | 75.0 | 0.0% |
| 2 | 5 | 4.2 | −72.0% | 59.85 | −20.2% |
| 3 | 5 | 3.38 | −77.47% | 55.05 | −26.6% |

**Table A11.** *Cont.*

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 5.7 | 0.0% | 46.0 | 0.0% |
| 2 | 2 | 5.9 | +3.51% | 38.25 | −16.85% |
| 3 | 2 | 5.18 | −9.12% | 35.72 | −22.35% |
| 1 | 3 | 20.4 | 0.0% | 61.0 | 0.0% |
| 2 | 3 | 7.0 | −65.69% | 52.48 | −13.97% |
| 3 | 3 | 6.98 | −65.78% | 50.25 | −17.62% |
| 1 | 4 | 55.0 | 0.0% | 76.0 | 0.0% |
| 2 | 4 | 11.12 | −79.78% | 68.75 | −9.54% |
| 3 | 4 | 10.22 | −81.42% | 68.45 | −9.93% |
| 1 | 5 | 78.8 | 0.0% | 91.0 | 0.0% |
| 2 | 5 | 12.02 | −84.75% | 75.85 | −16.65% |
| 3 | 5 | 10.35 | −86.87% | 69.6 | −23.52% |

**Table A12.** 16 Variables.

| Scenario | Density | Max Qubits Superposed | Comparison with Grover's Algoritm | Max Clauses in Oracle | Comparison with Grover's Algorithm |
|---|---|---|---|---|---|
| 1 | 2 | 16.0 | 0.0% | 32.0 | 0.0% |
| 2 | 2 | 1.42 | −91.12% | 25.55 | −20.16% |
| 3 | 2 | 1.25 | −92.19% | 21.95 | −31.41% |
| 1 | 3 | 16.0 | 0.0% | 48.0 | 0.0% |
| 2 | 3 | 2.33 | −85.44% | 41.9 | −12.71% |
| 3 | 3 | 2.15 | −86.56% | 41.65 | −13.23% |
| 1 | 4 | 16.0 | 0.0% | 64.0 | 0.0% |
| 2 | 4 | 4.8 | −70.0% | 56.9 | −11.09% |
| 3 | 4 | 4.35 | −72.81% | 56.55 | −11.64% |
| 1 | 5 | 16.0 | 0.0% | 80.0 | 0.0% |
| 2 | 5 | 4.0 | −75.0% | 64.28 | −19.65% |
| 3 | 5 | 4.4 | −72.5% | 67.5 | −15.62% |
| 1 | 2 | 6.7 | 0.0% | 49.0 | 0.0% |
| 2 | 2 | 5.68 | −15.22% | 42.2 | −13.88% |
| 3 | 2 | 4.9 | −26.87% | 36.4 | −25.71% |
| 1 | 3 | 19.4 | 0.0% | 65.0 | 0.0% |
| 2 | 3 | 9.65 | −50.26% | 58.9 | −9.38% |
| 3 | 3 | 8.2 | −57.73% | 58.0 | −10.77% |
| 1 | 4 | 104.5 | 0.0% | 81.0 | 0.0% |
| 2 | 4 | 18.1 | −82.68% | 73.85 | −8.83% |
| 3 | 4 | 13.25 | −87.32% | 73.15 | −9.69% |
| 1 | 5 | 131.4 | 0.0% | 97.0 | 0.0% |
| 2 | 5 | 13.68 | −89.59% | 81.28 | −16.21% |
| 3 | 5 | 13.85 | −89.46% | 84.0 | −13.4% |

## References

1. Cook, S.A. The Complexity of Theorem-Proving Procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, OH, USA, 3–5 May 1971; Association for Computing Machinery: New York, NY, USA, 1971; pp. 151–158. [CrossRef]
2. Mironov, I.; Lintao, Z. Applications of SAT Solvers to Cryptanalysis of Hash Functions. *IACR Cryptol. ePrint Arch.* **2006**, *2006*, 102–115.
3. Ganai, M.; Wang, C. SAT-Based Verification Methods and Applications in Hardware Verification. In Proceedings of the 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006, Bertinoro, Italy, 22–27 May 2006. [CrossRef]
4. Kautz, H.; Selman, B. Planning as Satisfiability. In Proceedings of the 10th European Conference on Artificial Intelligence, Vienna, Austria, 3–7 August 1992.
5. Lin, P.C.; Khatri, S. Efficient cancer therapy using Boolean networks and Max-SAT-based ATPG. In Proceedings of the International Workshop on Genomic Signal Processing and Statistics (GENSIPS), San Antonio, TX, USA, 4–6 December 2011; pp. 87–90. [CrossRef]
6. Davis, M.D.; Logemann, G.; Loveland, D.W. A machine program for theorem-proving. *Commun. ACM* **1962**, *5*, 394–397. [CrossRef]
7. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by Simulated Annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
8. Cerf, N.J.; Grover, L.K.; Williams, C.P. Nested quantum search and structured problems. *Phys. Rev. A* **2000**, *61*, 032303. [CrossRef]
9. Mandrà, S.; Guerreschi, G.G.; Aspuru-Guzik, A. Faster than Classical Quantum Algorithm for dense Formulas of Exact Satisfiability and Occupation Problems. *arXiv* **2015**, arXiv:1512.00859.
10. Ambainis, A. Quantum Search Algorithms. *SIGACT News* **2004**, *35*, 22–35. [CrossRef]
11. Hamadi, Y.; Jabbour, S.; Sais, L. ManySAT: A parallel SAT solver. *JSAT* **2009**, *6*, 245–262. [CrossRef]
12. Moskewicz, M.W.; Madigan, C.F.; Zhao, Y.; Zhang, L.; Malik, S. Chaff: Engineering an efficient SAT solver. In Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232), Las Vegas, NV, USA, 18–19 April 2001; pp. 530–535. [CrossRef]
13. Yu, L.; Zuo, Y.I.; Li, C.; He, A. A DPLL-based High-Concurrent SAT Solver with FPGA. *DEStech Transactions on Computer Science and Engineering*; DesTech: Toronto, ON, Canada, 2017. [CrossRef]
14. Mao, W. Solving satisfiability problems by the ground-state quantum computer. *Phys. Rev. A* **2005**, *72*, 52316. [CrossRef]
15. Leporati, A.; Felloni, S. Three quantum algorithms to solve 3-SAT. *Theor. Comput. Sci.* **2007**, *372*, 218–241. [CrossRef]
16. Gabor, T.; Zielinski, S.; Feld, S.; Roch, C.; Seidel, C.; Neukart, F.; Galter, I.; Mauerer, W.; Linnhoff-Popien, C. Assessing Solution Quality of 3SAT on a Quantum Annealing Platform. *arXiv* **2019**, arXiv:1902.04703.
17. Jeong, S.; Kim, M.; Hhan, M.; Ahn, J. Quantum Programming of the Satisfiability Problem with Rydberg Atom Graphs. *arXiv* **2023**, arXiv:2302.14369.
18. Meuli, G.; Soeken, M.; Micheli, G. SAT-based CNOT, T Quantum Circuit Synthesis. In Proceedings of the 10th International Conference, RC 2018, Leicester, UK, 12–14 September 2018; pp. 175–188. [CrossRef]
19. Cheng, S.T.; Tao, M.H. Quantum cooperative search algorithm for 3-SAT. *J. Comput. Syst. Sci.* **2007**, *73*, 123–136. [CrossRef]
20. Zhang, R.; Chen, J.; Zhao, H. Procedure of Solving 3-SAT Problem by Combining Quantum Search Algorithm and DPLL Algorithm. *Computing* **2020**, *4*, 14–24.
21. Silva, J.P.M.; Sakallah, K.A. GRASP—A New Search Algorithm for Satisfiability. In Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, USA, 10–14 November 1997; pp. 220–227.
22. Eén, N.; Sörensson, N. *An Extensible SAT-Solver*; Springer: Berlin/Heidelberg, Germany, 2003.
23. Mitchell, D.; Selman, B.; Levesque, H. Hard and Easy Distributions of SAT Problems. In Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, CA, USA, 12–16 July 1992.
24. Mouhoub, M.; Sadaoui, S.; Feng, X. A New Branch and Bound Method for Incremental Satisfiability Problem. *Informs J. Comput.* **2004**, *10*, 424–427.
25. Portilheiro, V. Applying Grover's Algorithm to Unique-k-SAT. 2018. Available online: https://vportilheiro.github.io/assets/writeups/quantum-sat.pdf (accessed on 13 February 2023).
26. Grover, L.K. A Fast Quantum Mechanical Algorithm for Database Search. In Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, Philadelphia, PA, USA, 22–24 May 1996; pp. 212–219. [CrossRef]
27. Grover, L.K. Quantum Computers Can Search Rapidly by Using Almost Any Transformation. *Phys. Rev. Lett.* **1998**, *80*, 4329–4332. [CrossRef]
28. Long, G.; Sun, Y. Efficient Scheme for Initializing a Quantum Register with an Arbitrary Superposed State. *Phys. Rev. A* **2001**, *64*, 014303. [CrossRef]
29. Ventura, D.; Martinez, T. Initializing the Amplitude Distribution of a Quantum State. *Found. Phys. Lett.* **1970**, *12*, 547–559. [CrossRef]
30. Paulet, J.J. Quantum Incremental SAT. 2021. Available online: https://github.com/Paulet02/quantum-incremental-SAT (accessed on 13 February 2023).
31. Lauria, M.; Elffers, J.; Nordström, J.; Vinyals, M. *CNFgen: A Generator of Crafted Benchmarks*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 464–473. [CrossRef]

32. de la Cruz Calvo H. I. QSimov. 2021. Available online: https://github.com/Mowstyl/QSimov (accessed on 13 February 2023).
33. Wichert, A. *Principles of Quantum Artificial Intelligence*; World Scientific: Singapore, 2013. [CrossRef]