NExt ApplicationS of Quantum Computing

D6.8: State of the art of SAT and PSA solvers in the light of quantum computing

Document Properties

Contract Number	951821
Contractual Deadline	30/09/2022
Dissemination Level	Public
Nature	Report
Editors	Mohamed Hibti, EDF R&D Ahmed Zaiou, EDF R&D and LaMSN, LIPN
Authors	Mohamed Hibti, EDF R&D Ahmed Zaiou, EDF R&D and LaMSN, LIPN Younès Bennani, LaMSN, LIPN Basarab Matei, LaMSN, LIPN
Reviewers	Vicente Moret Bonillo , UDC Gonzalo Ferro, CESGA
Date	September 30, 2022
Keywords	SAT solvers, benchmark, quantum SAT, quantum PSA
Status	Reviewed
Release	V 1.2

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 951821

History of Changes

Version	Date	Author, Organization	Description of Changes
1.0	12/09/2022	Hibti M., Zaiou A., Bennani Y. and Matei B. (EDF R&D, LIPN)	First Draft, Deliveravble 6.8, WP-6, Task 6.8
1.1	12/09/2022	Hibti M., Zaiou A., Bennani Y. and Matei B. (EDF R&D, LIPN)	First Draft, Deliveravble 6.8, WP-6, Task 6.8
1.2	12/09/2022	Hibti M., Zaiou A., Bennani Y. and Matei B. (EDF R&D, LIPN)	First Draft, Deliveravble 6.8, WP-6, Task 6.8

Table of Contents

1.Executive Summary

The main objective of this report is to understand the main factors that may help to solve fault tree analysis problems using quantum algorithms. It turns out that fault tree analysis can be considered an extended variant of Boolean Satisfiability Problem (SAT) which has attracted for decades a lot of research efforts from people and communities with a strong background on computer science and, particularly, in computational complexity related disciplines.

Many solvers exist for SAT and an annual competition is organized every year to push forward the performances of the algorithms to solve SAT instances.

In this report, an overview of last SAT competition winners is provided. Main review objectives were getting an idea of the different strategies and heuristics for solving SAT problems, understanding where the complexity of these algorithms is located and how non chronological search algorithms succeed to solve complex and big instances. The ultimate goal is to see to what extent these strategies can be embedded to enhance the design of hybrid quantum algorithms. Indeed, up to now the quantum algorithm that were already designed to solve either SAT or Probabilistic Safety Assessment (PSA) are limited by the hardware limitation regarding the number of available qubits and the decoherence time.

After a brief presentation of the main SAT-solvers, a synthesis of the experiments and results of the last SAT-competitions is presented, with an indication of the minimal and maximal sizes of the different instances that were solved. This confirms that these instances, can be generally at least as big as the industrial instances one can find in the reliability problems (or fault tree analysis) for complex industrial systems. Regarding the instances's complexity, we should note that the nature of the industrial instances has many particularities that one take can consider for simplification. Indeed, aside from redundancies (which may be the most complexity related aspect), there are many symmetries and modules that could be simplified for better performances.

Different quantum algorithms are presented to deal with SAT, they fall in different classes of quantum algorithms:

- Quantum walks,
- Harrow-Hassidim-Lloyd (HHL) based algorithms,
- Adiabatic algorithms,
- Chaotic dynamics,
- Ground state quantum computer algorithms,
- Parallel quantum algorithm,
- Cooperative search algorithm,
- Divide and Quantum.

For the PSA problem, in addition to many of the SAT algorithms that can be also used, we present other more specific algorithms

- Direct implementation with reversible game pebbling or ZX directed optimization,
- a vertex separator quantum algorithm,
- quantum unconstrained binary optimization
- a counting/grover algorithm

We present some tests on different small instances of fault trees and show that scaling remains an important question and therefore requires a focus on efficient hybrid approaches that combine performant classical solvers with a relevant use of quantum superposition when it matters.

2.Introduction

In the static framework of PSA (Probabilistic Safety Assessment) problems, where the problem is a reliability one stated in terms of Boolean logic, there is no consideration of the dynamics of the systems and the kinetic elements even if some of these aspects are considered at the early phase of elaboration of the success criteria [1].

The problem is then to evaluate a system, as a "sequence or consequence master fault tree", which is a Boolean formula for which we want to obtain all the cut sets (combinations of failures that make the formulae *True*). The set of cutsets is "quantified" to have a probability or a frequency of obtaining the realization of the Boolean formula. This correspond to the realization of the master fault tree top gate.

Quantified to have a probability or a frequency of obtaining the realization of the Boolean formulas that correspond to the event. To find it, we have to find all possible solutions that realize a Boolean formula, which is close to the Boolean Satisfiability Problem (SAT)[Cook, 1971]. SAT is NP-complete and is very much studied in the computer science field. This report aims to evaluate the contribution of the state of the art of recently proposed algorithms and heuristics to the efficiency of the solvers and to compare them with each other. The main question studied is whether the new heuristics do, indeed, improve the efficiency of solvers on a wide range of Benchmarks. Another objective is to understand the strengths and weaknesses of these heuristics.

The annual SAT competition[2] is a good observation platform for the most effective SAT solvers. In recent years, *Conflict-Driven Clause Learning* (CDCL) algorithms attracted more attention. These algorithms were originally based on *Backtrack Search Algorithms* à la DPLL (for *Davis–Putnam–Logemann–Loveland*).

In DPLL, at each step of the algorithm, a variable and a propositional value are selected for branching purposes. With each branching step, two values can be assigned to a variable, either 0 or 1 and their effects propagated. Whenever a conflict (dead end) is reached, backtracking is executed (undoing branching steps until an unflipped branch is reached). When both values have been assigned to the selected variable at a branching step, backtracking will undo this branching step. If for the first branching step both values have been considered, and backtracking undoes this first branching step, then the Conjunctive Normal Form (CNF) formula can be declared unsatisfiable ([Marques-Silva et al., 2021]).

The backtracking process in DPLL is called *Chronogical Backtracking*. The problem with chronological backtracking is that many of the withdrawn choices may have nothing to do with why the dead end (conflict) is a dead end. Thus, chronological backtracking can be inefficient. In real problems, it can be impractical ([Udovičić, 2006]).

As its name indicates, CDCL considers a conflict-driven learning, and thus unlike chronological backtracking, faulty plan is to withdraw the choices on which the dead end depends [Udovičić, 2006]. Moreover, it involves many other techniques [Marques-Silva et al., 2021].

- *Exploiting structure of the conflicts during clause learning and learning new clauses from conflicts during backtrack search which allows pruning large portions of the search space [Marques Silva and Sakallah, 1996].*
- *Using lazy data structures for the representation of formulae, and using branching heuristics that receive feedback from backtrack search [Moskewicz et al., 2001].*
- *Periodically restarting backtrack search and introducing randomization in the branching variable selection process [Gomes et al., 1998].*
- *Additional techniques include deletion policies for learnt clauses [Goldberg and Novikov, 2002], the actual implementation of lazy data structures [Ryan, 2004], the organization of unit propagation [Lewis et al., 2005], among others.*

In [Kochemazov et al., 2020] Kochemazov S. proposed a state of the art on modern SAT CDCL solvers, which are programmatic implementations of Boolean satisfiability algorithms. Essentially, they are tools intended to be used in practice to solve difficult problems, which cannot be solved easily with existing approaches.

[1] The success criteria are conditions that when met are sufficient for the success of a system, instrumentation and control mission or human action in a specific context regarding the dynamic of some accident sequence triggered by an initiator. Generally, the success criteria are elaborated regarding the "worst case" situation which is considered as an envelope of all the potential scenarios.
[2] http://www.satcompetition.org/

The progress of SAT CDCL solvers is mainly achieved through annual SAT competitions where the winners are considered with great interest. These competitions use benchmark sets composed of instances from various application domains to test new heuristics and implementations of the SAT algorithms.

In [Kochemazov et al., 2020], Kochemazov focused on several promising heuristics proposed in recent years. A summary and description of the heuristics studied are presented below.

- ***LCM:*** *Learnt Clause Minimization the heuristic proposed in [a. M. Luo, 2017], which is aimed at improving the quality of learnt clauses by applying to them a special procedure that makes it possible to remove the redundant literals from a clause.*
- ***DISTANCE:*** *the distance heuristic [Chang et al., 2018] approach initializes the values of the branching heuristic by directing the search at the beginning. It is relatively expensive, so in most implementations the heuristic works for at most 100 000 conflicts.*
- ***CB:*** *Chronological Backtracking is described in [Nadel and Ryvchin, 2018] implements an attempt to make a partial return to the roots of SAT solving, in particular, to the DPLL algorithm. It allows the solver in certain conditions to ignore the non-chronological backtracking, which became one of the trademarks of CDCL solvers, in favor of a chronological one.*
- ***DL:*** *Duplicate Learnts heuristic [Kochemazov et al., 2020] attempts to use simple data mining methods to discover the learnt clauses that are repeatedly derived during the search and store them permanently.*
- ***SLS:*** *the Stochastic Local Search component augmented with Rephasing technique has been appearing in the growing number of solvers. While the idea of combining CDCL and SLS is far from novel, the implementation of the SLS component in Relaxed_LCMDCBDL_newTech [Zhang and Cai, 2020] has several specific features that have not been used before.*

In addition to this classical SAT benchmarks, we do some tests with quantum algorithms to solve the extended version of SAT proposed in NEASQC Deliverable D6.4 ([Rennela et al., 2021]). The problem of finding all the prime implicants is close to the SAT problem, in the sense that we are searching not only a satisfiable assignment, but all the assignments. In practice, the event combinations of low probability/frequency (lower than some threshold) are neglected and that is another pruning possibility of parts of the search tree.

Therefore, the fundamental difference, is that, for industrial cases, the Boolean formulae representing the master fault trees for which prime implicants are searched are not only satisfiable but may have a huge number of solutions.

The question then is to each extent these SAT algorithms may be of interest to the original PSA problem.

The straightforward procedure to use SAT algorithms to solve the PSA problem by recursively adding a negated form of the solution is not efficient nor practical.

The rest of is document is organized as follows. In section 3, we present a number of SAT solvers that have shown good performances in the recent SAT competitions. Section 4 is dedicated to present the results of such solvers on the considered instances while in section 5, we present the results of the last SAT competitions. In section 6, we present a state of the art of quantum algorithms to solve SAT. We finally present a number of quantum algorithms to solve fault trees in section 7 and present the benchmarks of these algorithms in section 8.

3.SAT solvers

This chapter is dedicated to the presentation of the main SAT solvers, Figure 1, that showed the best performances in recent SAT competitions ([Balyo et al., 2017], [Heule et al., 2019] [Bal, 2020], [Froleyks et al., 2021] and [Kochemazov, 2021]) and the associated heuristics. In figure 1, a map show the relation between these heuristics.

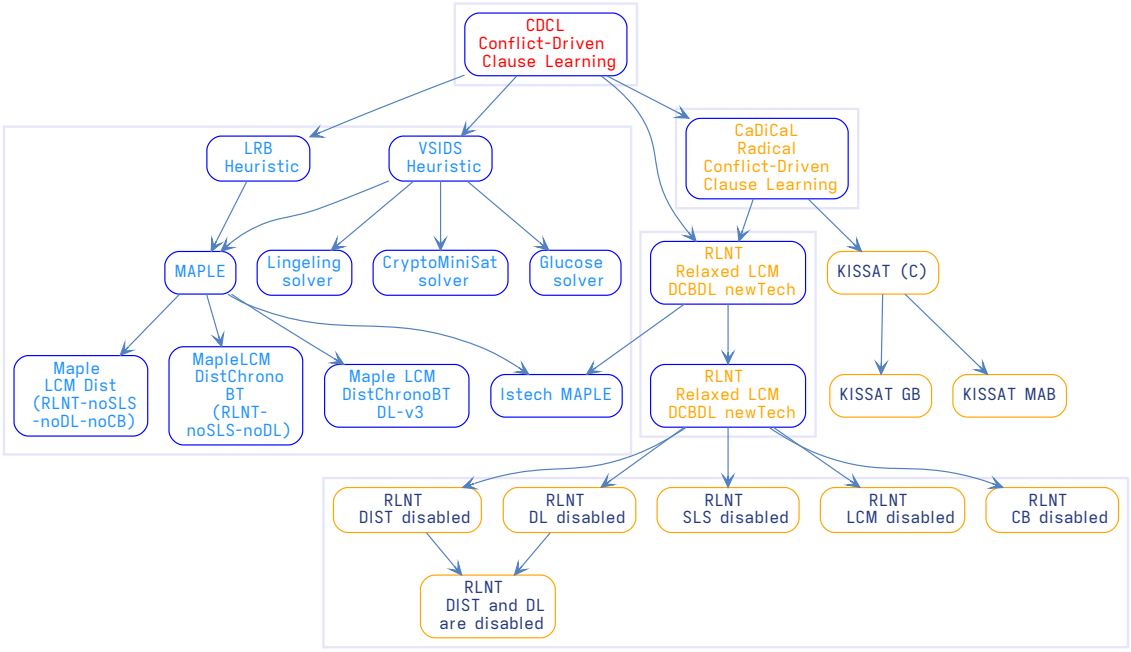


***Figure 1**: Map of the main sat solvers cited in this report*

3.1. DPLL

The basic backtracking algorithm (see Figure 2) runs by choosing a literal, assigning a truth value to it, simplifying the formula and then recursively checking if the simplified formula is satisfiable; if this is the case, the original formula is satisfiable; otherwise, the same recursive check is done assuming the opposite truth value.

In Davis, Putnam, Logemann and Loveland's algorithm (DPLL) (see figure 2), if a clause contains only a single unassigned literal, this clause can only be satisfied by assigning the necessary value to make this literal true. By removing every clause containing a unit clause's literal and discarding the complement of a unit clause's literal from every clause containing that complement, the algorithm leads to cascades of units eliminations, thus avoiding a large part of the naive search space.

Moreover, if a propositional variable occurs with only one polarity in the formula, it is called pure. Assign it to `True` or `False` do not have any impact on the other clauses and cannot constrain the search space anymore. It can then be deleted. This is *Pure literal elimination* .

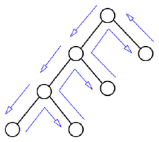


Figure 2: Original backtracking algorithm

3.2. CDCL

CDCL considers a conflict-driven learning, and unlike chronological backtracking, faulty plan is to withdraw the choices on which the dead end depends [Udovičić, 2006]. Moreover, it involves many other techniques [Marques-Silva et al., 2021]. It works as follows[1]:

1. Decision state: Select a variable and assign True or False and remember the assignment.
2. Apply Boolean constraint propagation (unit propagation[2]).
3. Build the *implication graph*[3]
4. If there is any conflict
   a) Find the cut in the implication graph that led to the conflict
   b) Derive a new clause which is the negation of the assignments that led to the conflict
   c) Non-chronologically backtrack ("back jump") to the appropriate decision level, where the first-assigned variable involved in the conflict was assigned
5. Otherwise, continue from step 1 until all variable values are assigned.

A comprehensive example of these steps of CDCL by Tommi Junttila can be found in `https://users.aalto.fi/~tjunttil/2020-DP-AUT/notes-sat/cdcl.html`.

3.3. VSIDS

Variable State Independent Decaying Sum (VSIDS) refer to a family of branching heuristics widely used in modern SAT solvers that pick variables of Boolean formula following some non-chronological order during the run of the search procedure. In VSIDS the following holds (cf. [Moskewicz et al., 2001]):

1. *Each variable in each polarity has a counter, initialized to 0.*
2. *When a clause is added to the database, the counter associated with each literal in the clause is incremented.*
3. *The(unassigned) variable and polarity with the highest counter is chosen at each decision.*
4. *Ties are broken randomly by default, although this is configurable*
5. *Periodically, all the counters are divided by a constant.*

A list of the unassigned variables sorted by counter value is stored during Boolean Constraint Propagation (BCP) and conflict analysis (using an Standard Template Library (STL) set) so that a quick decision can be made by choosing the highest counter variable. The strategy tend to satisfy (recent) conflict clauses and has a particular impact on hard instances that generate many conflicts[Moskewicz et al., 2001]. It is also characterized by its low overhead since the statistics are updated only when a conflict is encountered.

3.4. CaDiCaL

The name of the solver has its roots in “radical(ly)” and “CDCL” ([Marques-Silva et al., 2021] cited in [Biere., 2019]). The main search loop of Radical CDCL Solver (CaDiCaL) combines in-processing

[1] wikipedia
[2] Unit propagation can be naturally associated with an implication graph that captures all possible ways of deriving all implied literals from decision literals [Beanie et al., 2003], which is then used for clause learning.
[3] See appendix A.1

[Järvisalo et al., 2012] and CDCL [Marques-Silva et al., 2021] search. The in-processing part consists of three individually scheduled in-processing methods: probing, subsumption, and (bounded) variable elimination (CDCL is described in section 3.2).

3.5. LRB

Assume we are faced repeatedly with a choice among n different actions. Each choice leads to a monetary reward chosen from a hidden stationary probability distribution that depends on the selected action. The Multi-Armed Bandit (MAB) problem is to maximize the cumulative reward after some period combining greedy[4] and exploratory[5] choices. An approach to solve it is to use a technique called exponential recency weighted average (ERWA) to estimate a moving average incrementally giving more weight to the more recent outcomes.

Learning Rate Branching heuristic (LRB) is based on ERWA and is designed to maximize the Learning Rate (LR) of the solver. Branching is then considered as an optimization problem, where the degree of contribution from an assigned variable to the progress of the solver must be maximized. The degree in question is considered as the propension of a variable to generate learnt clauses. A more formal definition can be found in [Liang et al., 2016].

3.6. MAPLE

The Maple series of SAT solvers is a family of conflict-driven clause-learning SAT solvers outfitted with machine learning-based heuristics.

Maple (Maple) uses LRB heuristic, a departure from the VSIDS branching heuristic that has been the status quo for many years of SAT solving[6].

LRB is based on the idea that online variable selection in SAT solvers is viewed as an optimization problem, where the learning rate (LR) must be maximized (LR being a numerical characterization of a variable's propensity to generate learnt clauses) [Liang et al., 2016].

3.7. Lstech MAPLE

Local Search Tech Maple is an improved version of Relaxed LCM DCBDL newTech (lstech MAPLE) based on **R**elaxed **L**CMDCBDL **N**ew**T**ech (RNLT). Where in the last, the relaxed method is applied to the backtracking process for protecting promising partial assignment, here where the CDCL process meets some conditions, the algorithm will enter a non-backtracking stage until it gets a full assignment α. Once it gets α, a local search SAT solver is called immediately [Zhang et al., 2021].

The differences between Local Search Tech (LSTech) and Relaxed Conflict-Driven Clause Learning (Relaxed) lie in the non-backtracking stage entrance conditions and the local search process entrance conditions. Inspired by the updating method of target phase, LSTech enters the non-backtracking phase to construct a full assignment each time the CDCL process reaches a higher trail. And LSTech enters the local search process according to the number of restarts, rather than after each non-backtracking stage.

3.8. KISSAT (C)

The Kissat SAT solver is a condensed and improved reimplementation of CaDiCaL in C. It has improved data structures, better scheduling of inprocessing, optimized algorithms and implementation. (KISSAT) solver is a condensed and improved reimplementation of CaDiCaL in C. It has improved data structures, better scheduling of inprocessing, optimized algorithms and implementation[7].

[4] A choice based of the past rewards distribution and that exploit what we know up to then.
[5] An exploratory choice which consists on betting on new choices, and thus getting more information of feedback.
[6] https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/
[7] http://fmv.jku.at/kissat/

In its last version of 2021, KISSAT uses target phases during focused mode and usually works better for satisfiable instances [Biere Armin and Heisinger, 2021].

3.9. KISSAT GB

This solver implements the Glue Bumping (GB) method on top of KISSAT. KISSAT employs two branching heuristics: VSIDS and *Variable Move to Front* Variable Move to Front Strategy (VMTF). In Kissat with Glue Bumping strategy (KISSAT GB), the GB scheme is kept active only when VSIDS is active.

3.10. KISSAT MAB

Based on KISSAT, with Multi-Armed Bandit (MAB) framework which combines VSIDS and the Conflict-History Based (CHB) branching heuristics by adaptively choosing a relevant heuristic at each restart using the Upper Confidence Bound (UCB) strategy.

3.11. KISSAT CRVR GB

The solver Kissat with CRVR and GB (KISSAT CRVR GB) implements the GB and Common Reason decision Variable score Reduction (CRVR) method on top of KISSAT SAT. In KISSAT CRVR GB, the GB and CRVR schemes are kept active only when VSIDS is active.

3.12. KISSAT CMS EXP VGBL

The solver GB method on top of LRB, and replace VSIDS with expVSIDS. (CMS EXP VGBL): The baseline CryptoMiniSat solver (CryptoMiniSat) 5.8.0 employs a combination of three branching heuristics: LRB, VSIDS and VMTF. This system extends this baseline by implementing the GB method on top of LRB, and by replacing VSIDS with VSDIS with expScore (expVSIDS).

3.13. CaDiCaL Hack Gb

GB on the top of CaDiCaL1.4.0 when VSIDS is active in the baseline system (CaDiCaL Hack GB) implements the GB method on the top of CaDiCaL1.4.0, only when VSIDS is active in the baseline system. CaDiCaL Hack GB is submitted to the CaDiCaL hack track of the SAT competition-2021.

3.14. Relaxed Backtracking with Rephasing

One of the best SAT solvers is `Relaxed LCMDCBDL newTech` we will name it `rlnt`, which is based on two main processes. Relaxed CDCL and probabilistic based phase saving.

3.14.1. Relaxed CDCL Approach

The idea is to relax the backtracking process for protecting promising partial assignment, where a promising assignment is defined according to its consistency (no conflict) and length. When the CDCL process reaches a node with some conditions, the algorithm enters a non-backtracking phase until it gets a full assignment β.

Then Local search process is called to seek for a model near β.

If the local search fails to find a model within certain limits, then the algorithm goes back to the normal CDCL search from the node where it was interrupted.

For a given CNF with V variables, $|V|$ denotes the number of variables. And for a partial assignment α in CDCL process without conflicts, $|\alpha|$ is the number of assigned variables in α, then we name the max number of $|\alpha|$ in CDCL history as max trail. Here we control the entrance of local search process by p, q and

c, where p, q presents $|\alpha|/|V|$ and $|\alpha|/\max$ trail. And c presents the in-processing times between two local search processes.

3.14.2. Probability Based Phase Saving

Phase saving is a technique that saves the assignment of variables when traceback and uses the assignment when variables are selected as decision variables. Like the rephase technique in CaDiCaL [Biere., 2019], this approach uses vectors to save different phases, the difference is that it uses probability to select which phase to use after each restart.

4.SAT benchmarks

4.1. Benchmarks

To accurately assess the performance of CDCL SAT solvers it is necessary to use a diverse set of benchmarks that come from different applications. The benchmark sets used in SAT Competitions appear to satisfy this criterion, thus in this report, they were used in the experiments. In particular, the test instances from the main tracks of SAT Competitions 2016[1] and 2020[2], and also SAT Race 2019[3] were used. They are freely available from the corresponding websites. In total there were 1300 instances.

4.2. Evaluation criteria

Remind, that when a SAT solver is launched with a fixed time limit on some SAT instance, there are three possible outcomes.

1. It can find a satisfying assignment, meaning that a SAT instance is satisfiable.
2. It can prove that there are no satisfying assignments and thus the input formula is unsatisfiable.
3. If the solver terminates due to the time limit, then the status of the instance is unknown.

Overall, taking into account the fact that a formula can have many satisfying assignments, it may seem that satisfiable instances are significantly easier to tackle for a SAT solver, compared to unsatisfiable instances, for which it is required to construct a specific proof. Empirical observations show that it is indeed true but to a limited extent. Both hard and simple cases of satisfiable and unsatisfiable instances are often encountered in practice.

In the experiments, each considered solver was launched once on each instance from the benchmark set. The outcome of the launch, together with the runtime of a solver was recorded. The statistics presented below include the count of solved instances, divided into counts for solved satisfiable and unsatisfiable ones. Similar to the SAT competitions, these numbers are accompanied by the PAR-2 score (Penalized Average Runtime with factor 2) which is computed as follows. First, a sum of terms is computed, where for each instance from the benchmark set the term is formed as a runtime of a solver if it solved this instance, or as 2 times the time limit if the solver did not. Then this sum is divided by the number of terms. Given the two solvers that solved the equal number of instances, the one with the lower value of PAR-2 is faster on average.

4.3. Computing platform

Following the standard SAT Competition procedure, all solvers were launched with a time limit of 5000 seconds. As a computing platform the PCs with 16-core AMD Ryzen 3950x CPUs and 32Gb RAM were used, operating under Ubuntu 20.10. The solvers were launched in 16 simultaneous threads without restrictions on memory usage.

4.4. Computational experiments

The results of the experiments are summarized in Table 1. The so-called cactus plots, that present the performance of the solvers on satisfiable instances (satis) and unsatisfiable instances (unsatis), are showed in Figures 3 and 4. The plotline for each solver displays the runtimes of this solver over all benchmarks, ordered in ascending order. It means that on the cactus plot, the further the plotline is to the right the larger number of instances were successfully tackled by the corresponding solver within the time limit, and the closer the line is to the bottom the smaller is the average runtime of a solver over all benchmarks.

[1] https://baldur.iti.kit.edu/sat-competition-2016/index.php?cat=downloads
[2] https://satcompetition.github.io/2020/downloads.html
[3] http://sat-race-2019.ciirc.cvut.cz/index.php?cat=downloads

Solver	Solution Count Ranking	Numbers of solved satisfiable benchmarks	Numbers of solved unsatisfi-able benchmarks	Penalized average runtime
RLNT-noDL-noDIST	726	405	321	4841.34
RLNT-noDIST	721	400	321	4875.14
RLNT-noDL	721	403	318	4883.67
RLNT	715	399	316	4904.70
RLNT-noCB	697	378	319	5024.10
RLNT-noSLS-noDL	694	344	350	5179.37
RLNT-noLCM	684	387	297	5116.70
MapleLCMDistChronoBT-DL-v3	682	331	351	5252.24
RLNT-noSLS-noDL-noCB	682	337	345	5270.08
RLNT-noSLS	671	331	340	5300
RLNT-noSLS-noDL-noCB-noDIST-noLCM	670	340	330	5341.44
MapleLCMDistChronoBT	665	312	353	5413.83
MapleLCMDist	658	303	355	5483.10

***Table 1**: The results of computational evaluation of different solvers on the joint set of benchmarks from SAT Competitions 2016 and 2020 and SAT Race 2019 (1300 tests in total).*

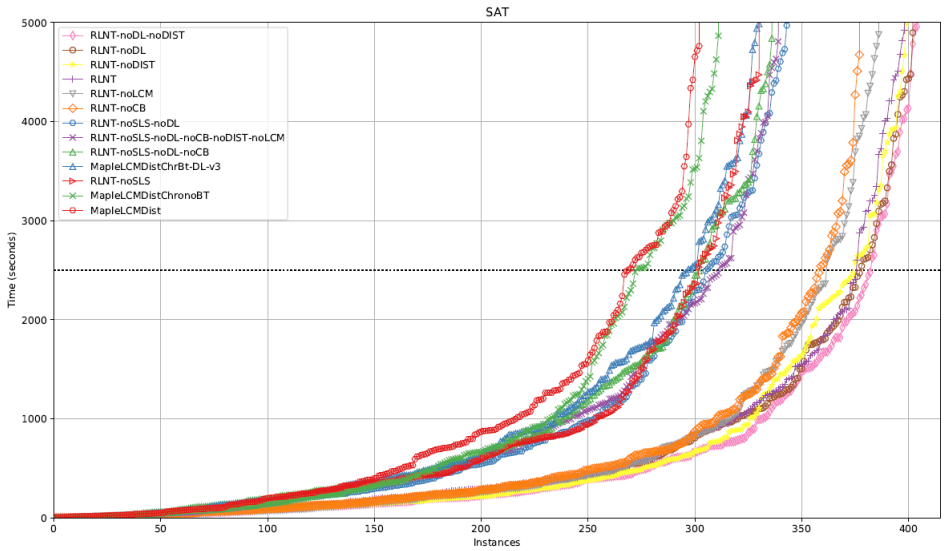


***Figure 3**: Cactus plot for considered solvers over satis*

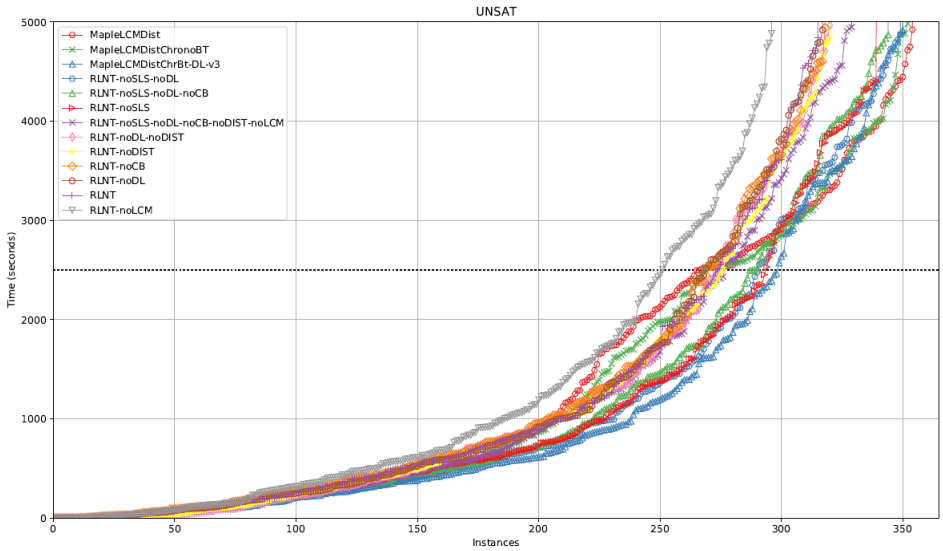


***Figure 4**: Cactus plot for considered solvers over unsatis*

4.5. Discussion

From the results in Table 1, and those presented in Figures 3 and 4, several observations can be made. Firstly, on the benchmark set used at least, the heuristics do not produce equal gain. In the first three rows of the table, we see that dropping certain heuristics increases the performance of the solver. Specifically, it appeared that disabling DL or DIST allows RLNT to solve a larger number of satisfying and unsatisfying instances. Additionally, deactivating them together is even more beneficial. Therefore, it is necessary to re-evaluate the combination of heuristics for each solver if the goal is to obtain the best possible performance. That means that when incorporating a new heuristic into a solver, it is best to study how it fits into the overall array. However, this makes it much more difficult to make direct and fair comparisons.

From Figure 3, the SLS heuristic appears to give the solver the ability to cope with at least 50 more satisfiable test instances than the competition for the set of benchmarks considered, which is quite amazing. It is the SLS heuristic that is responsible for the large gap between the two groups of lines in Figure 3. From Table 1, we can see that dropping SLS results in a significant gain (about 30) in the number of resolved unsatisfiable instances, which is, however, dwarfed by the decrease in the number of resolved satisfiable trials. The second-best heuristic seems to be LCM which mainly targets unsatisfiable instances. The CB heuristic, which helps the solver a lot with satisfiable instances, is in third position. Another observation is that the overall architecture of SAT solvers has indeed improved slightly in recent years, since, for example, RLNT-noSLS-noDL shows better performance compared to MapleLCMDistChronoBT which uses the same set of heuristics.

Comparing the graphs in Figures 3 and 4, we can conclude that since 2017, solvers have begun to concentrate on satisfiable instances, often at the expense of unsatisfiable instances. In particular, the 2017 MapleLCMDist solver successfully solved the largest number of unsatisfiable instances and the smallest number of satisfiable instances among all solvers tested. However, this change in focus is accompanied by an overall improvement that results in a decrease in the average execution times of the solvers.

5.SAT competition recent results

In this section we present the different results of the recent successive session since 2019, with a focus on sequential SAT solvers and their evaluation on structured, non-random benchmarks coming from various application areas. In this competition, solvers needed to output certificates for both the satisfiable and the unsatisfiable instances.

5.1. Session 2019

In the SAT Race 2019, 55 solvers participated with 200 benchmarks, in addition to 200 instances (not solvable by MiniSAT in 600 seconds) from prior competitive events.

In this competition the top 3 solvers of the Main Track SAT were.

1. CaDiCaLsat (3176.29)
   CaDiCaLdefault (3322.23) by Armin Biere2.
2. MapleLCMDistChronoBT-DLv2.1 (3436.32)
   MapleLCMDistChronoBT-DLv2.2 (3441.61)
   MapleLCMDiscChronoBT-DLv3 (3448.87) by Stepan Kochemazov, Oleg Zaikin, Victor Kondratiev, and Alexander Semenov
3. SmallSATdefault (3505.78) by Jingchao Chen

5.2. Session 2020

For the main sequential track 50 solvers were competing on the basis of 400 benchmarks, a combination of "application" and "crafted". In the competition the solvers are required to solve instances within a limit of 5,000 sec, where 40,000 sec is the limit for proof checking Solvers run on a single core. Moreover, UNSAT proof logging is required.

In this competition the top 3 solvers of the Main Track SAT are ([Bal, 2020]):

1. **Relaxed LCMDCBDL newTech** by Xindi Zhang and Shaowei Cai.
2. **Kissat-sc2020-sat** (PAR-2: 3128, 146 solved) by Armin Biere (https://github.com/arminbiere/kissat)
3. **Cryptominisat-ccnr-lsids** (PAR-2: 3263, 144 solved)
   Cryptominisat-ccnr (PAR-2: 3317, 145 solved) by Mate Soos, Shaowei Cai, Jo Devriendt, Stephan Gocht, Arijit Shaw, and Kuldeep Meel.

Figure 5 shows the top 10 winners of the main track competition for solving SAT instances, while Figure 6 shows the performance of the top 10 solvers in the main track unsatisfiable instances.

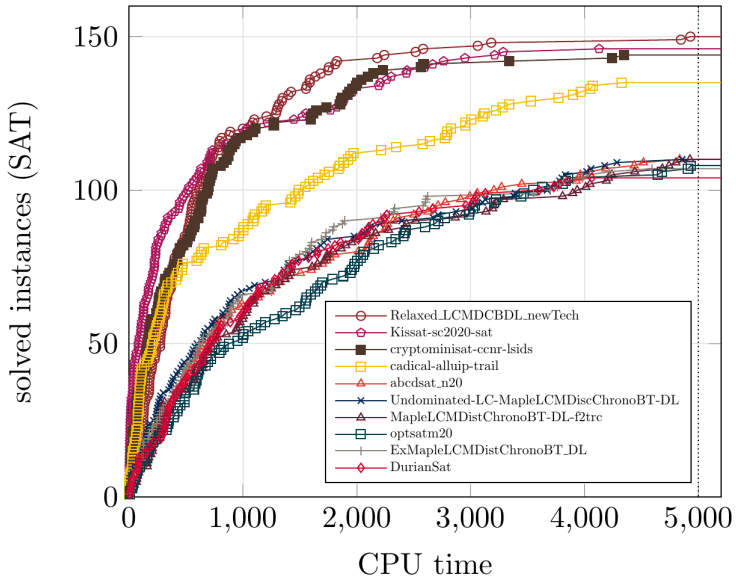


Figure 5: Top 10 Main track (src [Froleyks et al., 2021])

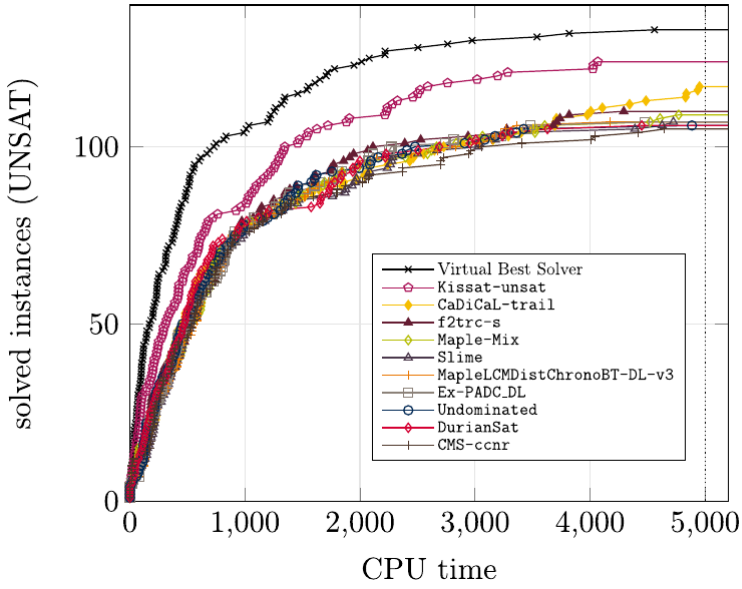


Figure 6: Top 10 Main track (src [Froleyks et al., 2021])

5.3. Session 2021

The Top 3 solvers of the Main Track SAT are:

1. Kissat MAB (PAR-2: 2222, 148 solved) by Mohamed Sami Cherif, Djamal Habet and Cyril Terrioux
2. lstech maple (PAR-2: 2358, 144 solved) by Xindi Zhang, Shaowei Cai, and Zhihan Chen
3. kissat gb (PAR-2: 2430, 143 solved) by Md Solimul Chowdhury, Martin Muller and Jia-Huai You

Figure 7 shows the top 10 winners of the main track competition for solving SAT instances, while Figure 8 shows the performance of the top 10 solvers in the main track unsatisfiable instances.

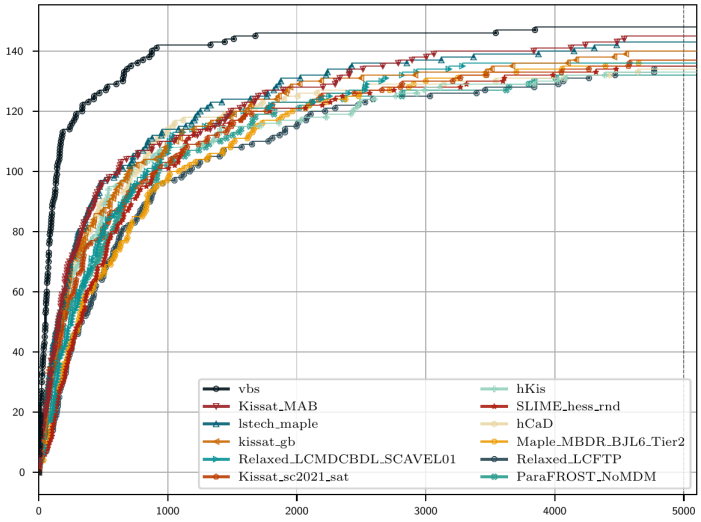


***Figure 7**: Top 10 Main track (solved SAT instances according to CPU time) (src [Froleyks et al., 2021])*

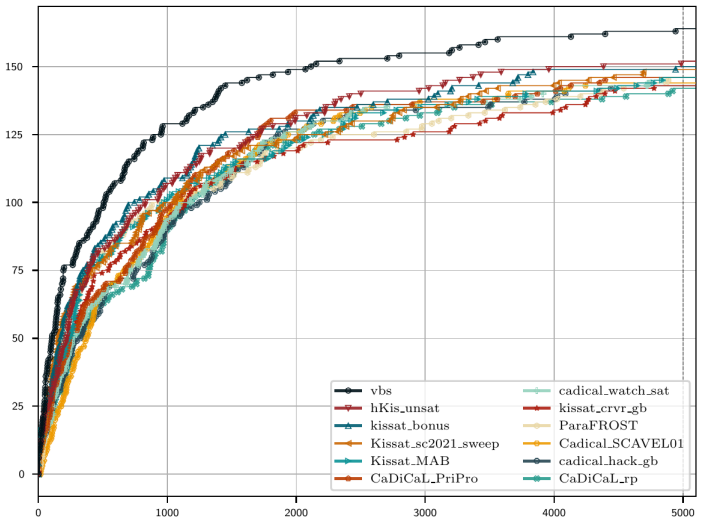


***Figure 8**: Top 10 Main track (solved unsatisfiable instances according to CPU time) (src [Froleyks et al., 2021])*

5.4. Instances Sizes

We should note that the instances solved in these competitions are from small to huge intances. In the table 2, we present the minimal and maximal size of each competition data set. This is just to keep in mind the scalability issues we will see in section 6.

***Table 2**: Table of the max and min sizes of the instances and their connections*

Year	Min of variables	Max of variables	Min of connections	Max of connections
2019	97	9582411	97	103690720
2020	54	8043699	54	129333040
2021	44	3416996	44	67009046

In Figure 9, we see the number of connections for the different sets of instances, while in Figure 10, we show the number of variables of the different datasets.

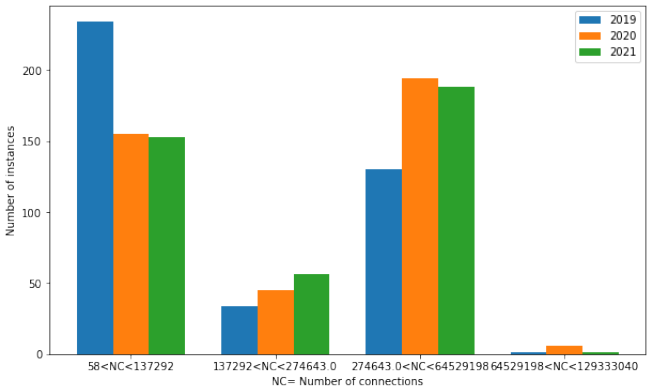


***Figure 9**: Number of connections of different sets of instances*

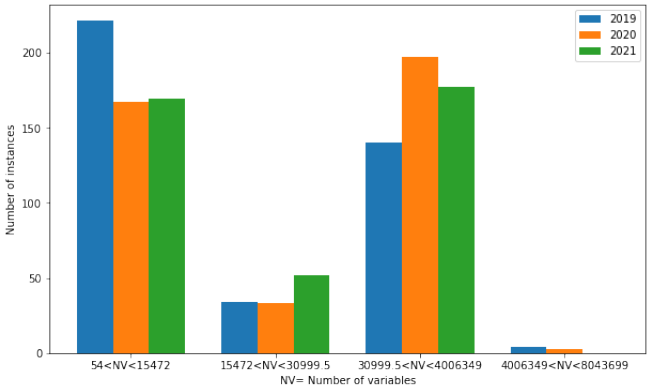


***Figure 10**: Number of variables of different sets of instances*

6. Quantum SAT

In this section, we give an overview of the main quantum algorithms for SAT, mainly quantum walks based algorithms (cf. section 6.1), HHL based algorithms (cf. 6.2), adiabatic comutation based models (cf. 6.3), chaotic dynamics based algorihms (cf. 6.4) and groud state quantum computing (cf. 6.5).

6.1. Quantum Walks

Many algorithms were tried using quantum walks to solve SAT problem. In this section, we will give a very brief overview of a short list of them ([Montanaro, 2019], [Campos et al., 2021], [Martiel and Remaud, 2020]). In [Montanaro, 2019], there is a more complete list of works using quantum walks to speed up backtracking algorithms.

6.1.1. Speedup of Backtracking Algorithms

A constraint satisfaction problem (Constraint Satisfaction Problem (CSP)) P is defined by a set $C = \{C_1, \ldots, C_m\}$ of constraints over a set of variables $X = \{X_1, \ldots, X_n\}$ having their values in a domain D.[1] The problem is to find an assignement of the variables X_i to satisfy the constraints C_i. Therefore, SAT can be considered of such a problem where the constraints are the different predicates composing the Boolean formulae and the domain is the set $0, 1$. In [Montanaro, 2019], SAT is considered as a CSP problem. Backtracking algorithms have proven their performance on such problems, and in [Montanaro, 2019] it is shown that these algorithms run substancially faster than their classical counterparts using quantum walks in trees.

Indeed, given a backtracking algorithm A, for any $0 < \delta < 1$, there is a quantum algorithm which, given T (an upper bound on the number of vertices in the tree explored by A), makes $O(Tn^{3/2}log(n)log(1/\delta))$ evaluations of each of P and h (the heuristic considered in the backtracking algorithm[2]), and outputs x such that $P(x)$ is true, or "not found" if no such x exists (cf. [Montanaro, 2019]).

The algorithm uses $poly(n)$ space, $O(1)$ auxiliary operations per use of P and h, and fails with probability at most δ.

In section 5, we have seen different performance achievement of the heuristics used to solve SAT using backtracking algorithms. The question is how these heuristics may behave in the quantum framework. It turns out that such comparison was carried out by Campbell et al. in [Campbell et al., 2019], in the case of k-SAT[3], this speedup can solve an instance by the quantum algorithm in one day, while the classical algorithm, run on a standard computer, would require more than 10^5 days.

6.1.2. Quantum tunneling and quantum walks as algorithmic resources to solve hard K-SAT instances

K-SAT is a special case of SAT where instances are in CNF of disjunctions of k literals. In [Campos et al., 2021], a quantum heuristic algorithm based on quantum walks and quantum tunneling is proposed to solve it. The algorithm uses a Hamiltonian constructed to solve K-SAT instances. The evolution of this Hamiltonian is done through a quantum walk where each iteration consists in the evolution of an evolving state following a hypercube graph. The idea of following a hypercube graph is to reduce the Hamming distance between the evolving state and a satisfying assignment of the K-SAT instance.

6.1.3. Quantum query algorithms

The `NAND` is known as a universal classical gate and any boolean formulae can be writter using only `NAND` gates. In [Ambainis et al., 2007], Ambainis et al. showed that evaluating a `NAND` formula of size N could be done in

[1] It can be a specific domain for each variable $D = \{D_1, \ldots, D_n\}$.

[2] In backtracking algorithms, there are many different heuristics that are used to explore the search graph in different ways. This is related to the strategy of choosing the next node to branch on a partial assignment.

[3] In the most optimistic regime where measurement time is of 0.5 ns, 2-qubit gate time is of 0.3 ns, cycle time of 2 ns and a gate error rate of 10^{-5} [Campbell et al., 2019].

time $N^{\frac{1}{2}+o(1)}$ which is close to $\sqrt{N}$ on a Quantum Computer using a query model[4]. In their algorithm, they used a search algorithm in top of a quantum walk, which is a clear improvment of the previous approaches for this problem (cf. [Farhi and Gutmann, 1997], [Høyer et al., 2003], [Buhrman et al., 1998], ...). A history of these approaches can be found in [Ambainis et al., 2007].

Jeffery S. et Kimmel S. have obtained similar results through s-t connectivity (cf. [Jeffery and Kimmel, 2017]). The evaluation of a Boolean formula is reduced to the problem of deciding in a graph if there is a path between a source s and a target t for which there is an algorithm with $O(\sqrt{(N)})$ complexity.

Andrew M. Childs et al. also found a similar result (cf. [Childs et al., 2009]) using the concept of random walk and a model of queries to an oracle (the term oracle designates a kind of algorithm portion in the form of a black box).

In this section we give a short description of [Ambainis, 2010].

We therefore assume a tree built only with NAND gates and whose leaves are x_i variables. We consider a model of requests to oracle. The entries $x_1, \ldots, x_N$ are accessible via requests in O to a black box.

To define O, we represent the base states as $|i, z\rangle$ where $i \in 0, 1, \ldots, N$. The transformation query O_x (where $x = (x_1, \ldots, x_N)$) transforms $|0, z\rangle$ into $|0, z\rangle$ and for $i \neq 0$ in $(-1)^{x_i} |i, z\rangle$.

The algorithm consists of a succession of queries O_x and of arbitrary transformations **not dependent on** x_i, and its goal is to calculate the value of the tree using the fewest queries possible.

6.2. HHL based Algorithms

Harrow-Hassidim-Lloyd (HHL) quantum algorithm can solve Linear System Problems (LSP) with exponential speed-up over "classical computing" approaches under certain conditions (sparsity and well-conditionning). Indeed, for a linear system $A|x\rangle = |b\rangle$, A has to be sparse and well conditionned. A $N \times N$ matrix A is called d-sparse if it has at most d non-zero entries in any row and column. We call it sparse if d scales in $poly(logN)$. It is called well-conditioned if its condition number[5] scales in $poly(logN)$, where the condition number of a matrix is the ratio of the largest to the smallest singular value, and undefined when the smallest singular value of A is 0 (i.e., when A is not invertible)[Childs et al., 2017]. In [Guan et al., 2021], it is noted that, in practice, systems of linear equations with a polylogarithmic condition number are quite rare and highlighted the result of Ambainis relaxing this condition to obtain at least a polynomial speedup.

In this section, we present two approaches that use HHL to solve SAT.

6.2.1. Quantum Algorithms for Boolean Equation Solving

In [Chen and Gao, 2022], an algorithm was proposed to solve Boolean equations through solving the Macaulay linear system using a modified version of the HHL algorithm and to obtain the Boolean solutions based on the properties of quantum solution state.

It is shown that for a set $\mathcal{F} = f_1, ..., f_r$ of Boolean polynomials in variables $X = x_1, \ldots, x_n$ and for $\epsilon \in (0, 1)$, there is a quantum algorithm which decides whether $F = 0$ has a solution and computes one if $F = 0$ does have solutions, with probability at least $1 - \epsilon$ and runtime complexity of $\tilde{O}((n^{3.5} + T^{3.5})\kappa^2 log1/\epsilon)$, where $\tilde{O}$ suppresses more slowly-growing logarithm terms and κ is the condition number of the Boolean polynomial system $\mathcal{F}$, and T the total sparseness of $\mathcal{F}$ (i.e. $T = \sum_{i=1}^{r} \#f_i$, and $\#f_i$ the number of terms in f_i).

The algorithm in question passes through three main steps: computing pseudo-solutions of the system via the HHL algorithm, and then compute Boolean solutions from the pseudo-Boolean solutions with high

[4]While in gate model of Quantum Computing, we have a set of qubits, an ordered set of gates applied to induvidual or multiple qubits and a number of measurements on some qubits, the Quantum query model adds the access to a black box (oracle), that *queries* some function that we want to learn about.

[5]The condition number associated with the linear equation $Ax = b$ gives a bound on how inaccurate the solution x will be after approximation. Conditioning is a property of the matrix and represent the rate at which the solution x will change with respect to a change in b.

probability. The problem of solving Boolean equation system is reduced to the computation of Boolean solutions of a 6-sparse polynomial system over $\mathbb{C}$.

6.2.2. An HHL-Based Algorithm for Computing Hitting Probabilities of Quantum Walks

HHL algorithm is used in [Guan et al., 2021], to compute hitting probabilities of quantum walks into absorbing boundaries. This is done via a reduction of the problem of computing hitting probabilities to a problem of inverting a matrix, for which HHL is proven to be efficient even with an exponential speedup under some conditions as it is shown in the introduction of section 6.2.

6.3. Adiabatic algorithms

Adiabatic Quantum Computation (AQC) is an universal model of computation that uses quantum mechanical processes operating under adiabatic conditions[6] [Grant and Humble, 2020]. It is universal in the sense that any computation in the gate model can be recasted in the AQC model, and it is based on the fact that a quantum system will stay near its instantaneous ground state if the Hamiltonian that governs its evolution varies slowly enough [Farhi et al., 2001].

AQC involves ground state quantum computation.

Van Dam et al. in [Van Dam et al., 2001] showed lower bounds for solving instances from some NP-Complete problems including SAT with formulae in CNF with each clause containing up to 3 literals (3-SAT) which is a good indication for the incapacity of the approach to solve efficiently 3-SAT and more generally SAT instances. We however present some algorithms that deal with this approach for the sake of generality. Recall that the main problem, we are dealing with, is Fault Tree Analysis (FTA) where, in practice, there are always not only one but so many solutions[7].

In this section, we present algorithms that use AQC to solve SAT.

6.3.1. Adiabatic quantum computing for random satisfiability problems

The evolution of a quantum system in time follows the Schrödinger equation

$$i\frac{d}{dt}|\psi(t)\rangle = H(t)|\psi(t)\rangle$$

where $|\psi(t)\rangle$ is the state vector and $H(t)$ the hamiltonian operator. The idea of an algorithm is to specify an initial state $|\psi(0)\rangle$ and a Hamiltonian such that at some time T the state $|\psi(T)\rangle$ encodes the solution we are looking for. In the adiabatic case [Farhi et al., 2001], the Hamiltonian evolves slowly enough to get a gound state $|\psi_g(t)\rangle$ that is close to the ground state $|\psi_g(0)\rangle$ of the operator at $t = 0$.

Therefore, we choose the Hamiltonian in such a way that the ground state at time $t = 0$ is known and the ground state $|\psi_g(T)\rangle$ encodes the solution. Meanwhile, the Hamiltonian $H(t)$ interpolate between $H(0)$ and $H(T)$ (cf. [Farhi et al., 2001])

$$H(t) = (1 - \frac{1}{T})H(0) + \frac{t}{T}H(T)$$

Then starting from the well known ground state $H(0)$ the evolution leads to a final state which is close to $|\psi_g(T)\rangle$.

In [Hogg, 2003], the algorithm continuously evolves the state of the quantum computer using

$$H(f) = (1 - f)H^{(0)} + fH^{(c)}$$

[6] Where the energy transfer to the surroundings is done only as work.
[7] These solutions represent the combinations of the elementary failures that cause the global failure of the system. There are always many!

a Hamiltonian with minimal-cost assignments as ground states is introduced

$$H^c_{r,s} = c(s)\delta_{r,s}$$

for assignments r and s, the cost $c(s)$ being the number of clauses s do not satisfy, and $\delta_{r,s}$ is 1 if $r = s$ and 0 otherwise.

This Hamiltonian introduces a phase factor in the amplitude of assignment s depending on its associated cost $c(s)$ [Hogg, 2003]. For $H^{(0)}$, they introduce for each variable i a non-negative weight w_i with $\sum_i w_i = \omega$.

$$H^0_{r,s} = \begin{cases} \omega/2 & \text{if } r = s, \\ -\omega/2 & \text{if } r \text{ and } s \text{ differ only on variable } i, \\ O & \text{otherwise} \end{cases}$$

and suggest it use the Walsh-Hadamard Transform (WHT) W with $W_{r,s} = 2^{-2/n}(-1)^{r.s}$[8].

For small instances, the algorithm performs in a number of the steps which is growing as a cube of the number of variables and gives solution probabilities close to 1. But it is not clear if a scaling of such algorithm is possible since the minimum energy gaps of most instances are large. Moreover, the resulting search costs are much higher than for other methods.

6.3.2. An algorithm based on an explicit modeling of Boolean mapping using Hatfield's rules or Fourier expansion

In [Bourreau et al., 2022], an adiabatic quantum algorithm is proposed to solve SAT where, each SAT formula is modeled with a Hamiltonian. For each clause a Hamiltonian connected with controlled-unitary operator that computes the function value in a qubit register. Boolean functions are modeled by a linear combination of Pauli Z operators via Hadfield's construction rules (cf. [Hadfield, 2021]), thus providing an explicit manner to define a problem mapping for Boolean functions. The following table represents the equivalent of each boolean operator with a combination of Pauli Z:

$f(x)$	H_f
x	$\frac{1}{2}I - \frac{1}{2}Z$
$\bar{x}$	$\frac{1}{2}I + \frac{1}{2}Z$
$x_1 \oplus x_2$	$\frac{1}{2}I - \frac{1}{2}Z_1Z_2$
$\bigoplus_{j=1}^{k} x_j$	$\frac{1}{2}I - \frac{1}{2}Z_1Z_2 \ldots Z_k$
$x_1 \wedge x_2$	$\frac{1}{4}I - \frac{1}{2}Z_1 + Z_2 - Z_1Z_2$
$\bigwedge_{j=1}^{k} x_j$	$\frac{1}{2^k}\prod_j(I - Z_j)$
$x_1 \vee x_2$	$\frac{3}{4}I - \frac{1}{4}Z_1 + Z_2 + Z_1Z_2$
$\bigvee_{j=1}^{k} x_j$	$I - \frac{1}{2^k}\prod_j(I + Z_j)$
$\overline{x_1x_2}$	$\frac{3}{4}I - \frac{1}{4}Z_1 + Z_2 - Z_1Z_2$

The Z_i operator is defined as

$$Z_i = I \otimes \ldots \otimes I \otimes \underset{i\text{th position}}{Z} \otimes I \otimes \ldots \otimes I$$

An experimentation of this approach was performed by Bourreau et al. with different values of T which leads to solutions with 94% of the distribution (with Qiskit [et al. ANIS, 2021] and MyQLM), and also experiments(on the 27 qubits IBM `ibmq_montreal`[9]).

[8] r and s are treated as vector of bits to count the variables assigned to 1 in both assignments.

[9] `ibmq_montreal` is composed of 27 qubits and has an average CNOT error about 1.2 10^{-2} and an average readout error about 1.8 10^{-2} with gates time about 426 ns.

	Percentage T=10	Percentage T=100	Percentage T=1000
$\|111\rangle$	12.7%	16.7%	16.6%
$\|100\rangle$	11.8%	17.3%	15.8%
$\|101\rangle$	62.8%	50.5%	51.8%
$\|001\rangle$	11.6%	15.5%	15.8%

***Table 3**: Experiment of the adiabatic algorithm by Bourreau et al. with 3 simulation times in [Bourreau et al., 2022]*

The results of Table 3 prove the capacity of the `ibmq_montreal` to solve this 4-clauses and 3 variables SAT problem.

	Number of shots	percentage
$\|111\rangle$	383	18.7%
$\|100\rangle$	411	20.1%
$\|101\rangle$	459	22.4%
$\|001\rangle$	314	15.3%
$\|010\rangle$	113	5.5%
$\|000\rangle$	107	5.2%
$\|110\rangle$	107	5.2%
$\|011\rangle$	154	7.5%

***Table 4**: Experiment of the adiabatic algorithm by Bourreau et al. with T=20 and 2048 shots in [Bourreau et al., 2022]*

6.3.3. Quantum Annealing

Quantum annealing (QA) is an optimization process, on special[10] quantum devices, for finding the global minimum of a given objective function over a given set of candidate solutions, by a process using quantum fluctuations (or temporary random change in the amount of energy in a point in space, as prescribed by Werner Heisenberg's uncertainty principle). The idea is to start from a quantum-mechanical superposition of all possible states (candidate states with equal weights), that continue to change realizing a quantum parallelism, which causes quantum tunneling between states. If the evolution of this system is slow enough, the system stays close to the ground state of the instantaneous Hamiltonian.

Quantum annealing is used for instance for combinatorial optimization problems where it is difficult to find some global minima (e.g. Traveling Salesman Problem (TSP), Traffic Flow Optimization (TFO), SAT, ...). The main process can be resumed by the following steps [Krüger and Mauerer, 2020]:

- Reduction to a Quadratic Unconstrained Binary Optimization (QUBO)
- Hardware embedding
- Hardware programming
- Execution
- Post-processing

In [Krüger and Mauerer, 2020], two reduction approaches were considered to reduce SAT with formulae in CNF with each clause containing up to 3 literals (k-SAT); one is Choi's standard reduction (cf. [Choi, 2011]) and the other, Krüger-Mauerer Blackbone reduction that showed different performances.

$$min[\vec{x}](-\sum_{l_{ij}} w l_{ij} + \sum_{l_{ij}, l_{ij'} \in C_i} \delta l_{ij} l_{ij'} + \sum_{lij=l\vec{i}'j} \delta l_{ij} l_{i'j}) \qquad Choi$$

[10] Quantum computer technology that is based on quantum mechanics and qubits, but with characteristics and performance levels intermediate between traditional supercomputers and general-purpose gate-based quantum computers. D-Wave is the main commercial player in this category [Ezratty, 2021].

$$q(\vec{x}) = \omega(\sum_{l_{ij},l_{ij'}} l_{ij}l_{ij'} + \sum_{l_{ij}=x_j} -l_{ij}x_j + \sum_{lij\neq x_j} -l_{ij} + l_{ij}x_j) \qquad \textit{Krüger-Mauerer}$$

See a good discussion in [Krüger and Mauerer, 2020], on the quality of the reduction in question, regarding the solution quality but also its scalability. There is an influence of the embedding method on the probability of finding correct satisfying assignments for randomly generated 3-SAT instances with varying ratios α of clauses to variables. The tests were performed using a dataset containing 250 random 3-SAT instances with 42 clauses each with samples sizes from 5 to 100.

In section 7.4, there is an approach to solve FTA by doing a similar reduction. It could be interesting to see to what extent the conclusions of [Krüger and Mauerer, 2020] can help better solving FTA.

6.4. Chaotic dynamics and quantum algorithms

In [Ohya and Volovich, 2003], a combination of quantum computer with a chaotic dynamics amplifier is proposed to solve a formulation of SAT. This quantum chaos computer is a new model of computation going beyond usual scheme of quantum computation. The amplification considered here is claimed to occur in a polynomial time.

The SAT problem can be regarded as determining whether a formula in *Product Of Sums (POS)* form is satisfiable. The following analytical formulation of SAT problem is useful. We define a family of Boolean polynomials f_A, where A is a set

$$A = S_1, ..., S_N N, T_1, ..., T_N$$

where $S_i, T_i \subseteq \{1, ..., n\}$ and f_A is defined as

$$f_a(x_1, \ldots, x_n) = \prod_{i=1}^{n}(1 + \prod_{a\in S_i}^{N}(1 - x_a)) \prod_{b\in T_i}^{N}(x_b)$$

The SAT problem now is to determine whether there exists a value of $x = (x_1, \ldots, x_n)$ such that $f_A(x) = 1$.

The quantum version of the function $f(x) := fA(x)$ is given by the unitary operator $U_f |\mathbf{x}, y\rangle = |\mathbf{x}, y + f(x)\rangle$. We assume that the unitary matrix U_f can be build in the polynomial time. Now let's use the usual quantum algorithm:

(i) By using the Fourier transform produce from $|0, 0\rangle$ the superposition

$$|x\rangle := \frac{1}{\sqrt{2^n}} \sum_x |x, 0\rangle .$$

(ii) Use the unitary matrix U_f to calculate $f(x)$:

$$|v_f\rangle = U_f |x\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x, f(x)\rangle .$$

After applying the projector $P = I \otimes |1\rangle \langle 1|$ to the state $|v_f\rangle$, the result $f(x)$ may occur with a probability of $\| P |v_f\rangle \|^2 = r/2^n$ where r is the number of roots of the equation $f(x) = 1$[11].

6.5. Ground state quantum computer

Ground state quantum computing is an approach that circumvents the problem of decoherence [Mizel et al., 2001], and works with only quantum mechanical ground states without the need of a time-dependent control of a system. Indeed, *the qubits do not change in time; they are fixed in their ground states. The*

[11] For the reliability problem, where we look for all the solutions of $f(x)$, we know that r is high enough.

steps in the computation correspond, not to evolution between time points, but rather to development of the ground state between connected parts of the Hilbert space.

In [Mao, 2005], a new quantum algorithm is proposed to solve SAT by taking advantage of non-unitary transformation in ground state quantum computer. The energy gap scale of the ground state quantum computer is analyzed for 3-bit Exact Cover problems. Under some conditions, the algorithm solves SAT in a polynomial time whenever a specific criterion holds. Indeed, there is a ratio S_j/S_{j+1}, with S_j being the number of solutions when the j^{th} clause is applied, and S_{j+1} the number of solutions when the $(j+1)^{th}$ clause is applied, that conditions the success of the algorithm.

Remark. There is an important difference to note between simulation and ground state energy computation. Simulation aims to predict the behavior of a system while the objective of ground state energy computation is to optimize a global property of a system.

6.6. A Parallel Quantum Algorithm for the Satisfiability Problem

In [Liu et al., 2008], a classical parallel quantum algorithm for the satisfiability problem is proposed, and it is based on a combination of a counting algorithm to get the number of solutions of a formulae, and Long algorithm to find solutions. Indeed, Long algorithm is used instead of Grover algorithm for its successful rate of always 100%.

Similar to the Grover algorithm, each iteration in the Long algorithm consists of four steps:

(i) apply the oracle O to the whole $n_1 + n_2$ argument register and on condition the item satisfies the oracle, rotate the phase of the item by angle ϕ;

(ii) apply the Hadamard transform to register 2;

(iii) Perform a phase rotation ϕ on $|00\ldots0\rangle$ state;

(iv) apply the Hadamard transform on the n 2 argument register.

After J_1 iteration of the Long algorithm, make a multiple-qubit controlled NOT gate on the auxiliary qubit so that the auxiliary qubit changes from 0 to 1 when the oracle is satisfied.

6.7. Cooperative search algorithm

In [Cheng and Tao, 2007], an interesting approach, called Cooperative Quantum Search (CQS), is proposed to deal with 3-SAT problems. Cooperative Quantum Search (CQS) combines the evolutionary algorithm framework with a Grover search algorithm.

The idea is to use combined registers of quantum and classical variables to limit the action of the quantum search algorithm to few quantum variables by the use of what Cheng and Tao call *auxiliaries* [Cheng and Tao, 2007]. Auxiliaries are dedicated to classically prepare candidate assignments to search a solution of the complete formula using a Grover search algorithm.

The process of selection of candidate assignments is done within Evolutionary Algorithms (EA) framework with different strategies Genetic Huill-Climbing Algorithms for Satisfiability (GenSAT) and a modified version of GenSAT algorithm (GSAT), while the selection process of the variables to be considered quantumly is driven by the appearance number of the variables. Indeed, it is demonstrated in [Cheng and Tao, 2007] that selecting the variables having smaller appearance as qubit-variables has higher success probability on finding a solution than selecting the variables having larger appearance. [12] The Figure 11 shows the general circuit of the algorithm.

In this work, different strategies were followed to prepare the auxiliaries. The experimentation showed that the CQS algorithm with GenSAT has the best performance in terms of query complexity. Moreover, the optimal configuration (the best number of quantum bits regarding the number of classical variables) for the CQS algorithm is suggested by mathematical analysis.

[12] If two variables are equal in appearance, selecting the one which has larger value of |positive appearance–negative appearance| as qubit-variable will have higher success probability on finding a solution.[Cheng and Tao, 2007].

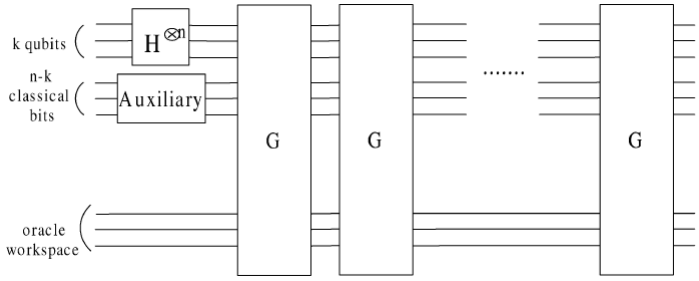


***Figure 11**: Quantum circuit with combined quantum and classical bits. The classical bits are picked at random following one of the GenSAT strategies and the quantum bit selection driven by the appearance number.*

6.8. Divide and Quantum with DPLL

While in [Cheng and Tao, 2007], scaling is allowed by the use of auxiliaries, by introducing combination of classical and quantum bits. In [Zhang et al., 2020], a divide and quantum approach in the head of DPLL is proposed to solve 3-SAT. The method divide a CNF into small parts and solve them separately by utilizing the structure of the DPLL algorithm (cf. section 3.1). Indeed, there is a conditioning that is done following a selection process like that involved in the cooperative approach [Cheng and Tao, 2007] (see section 6.7). But, here instead of choosing the variables dedicated to be considered quantumly, all the variables are considered as such, but the formulae are reduced by the conditioning into small fragments that can be quantumly solved separately.

The procedure used is to span the search tree and, starting from some level, solve the resulting Boolean formulae using Grover. Of course, pruning is performed when applicable to remove useless branches. The global solutions are built using the partial ones and the prior assignments to the node of the restricted formula.

In [Zhang et al., 2020], there is an estimation of the optimal[13] number of variables m that should be solved in the restricted formula (using Grover Algorithm) given an initial one of n variables:

$$m = \frac{2}{3}n + 2 + \frac{1}{3}log_2\pi^2 \tag{6.1}$$

6.9. Hybrid divide and conquer method

As pointed out in Neasqc D6.4 report (cf. [Rennela et al., 2021]), the hybrid divide-and-conquer method was used in [Dunjko et al., 2018b], [Ge and Dunjko, 2020] and [Dunjko et al., 2018a] to solve solve 3-satisfiability problems using a de-randomized version of the algorithm of Schöning, and for an algorithm for finding Hamilton cycles in degree-3 graphs. In [Rennela et al., 2021], the focus was to identify criteria when speed-ups, in the sense of provable asymptotic run-times of the (hybrid) algorithms, are possible in the framework of tree search algorithms.

As in section 6.8 (cf. [Cheng and Tao, 2007]), this work demonstrates speed-ups for the tree search subroutines of Paturi, Pudlák, Saks, and Zane Algorithm (PPSZ) – which is the core of the fastest exact Boolean satisfiability solver – for certain classes of formulae ([Hertli, 2011]).

In addition, this work highlighted the conditions of speed-ups and discussed query and space complexity of the hybrid algorithms in question. Indeed, Rennela et al. ([Rennela et al., 2021] Theorem 5.1) precised the factors that should hold to ensure clean polynomial speed-ups:

[13] To minimize the sum of the number of nodes of the spanned tree and the number of iterations that grows according to the number of qubits.
If this number is $f(m)$, then

$$f(m) = \frac{\pi}{4}2^{\frac{m}{2}} + 2^{n-m}$$

whose derivative equals 0 at m of equation 6.1 (cf. [Zhang et al., 2020]).

- The space complexity must yield a tree search decomposition where many of the subtrees which will be delegated to the quantum computer are large enough for a substantial advantage to be even in principle possible,
- and the quantum algorithm must actually realize such a polynomial advantage.
- The time complexities of the subroutines realizing one query in both classical and quantum routines are polynomial.

7.Quantum Fault tree analysis

By static approach, it is meant PSA framework where the problem is a reliability one stated in terms of Boolean logic. There is no consideration of the dynamics of the systems and the kinetic elements even if some of these aspects are considered at the early phase of elaboration of the success criteria[1].

The problem is then to evaluate a system, a "sequence or consequence master fault tree", which is a Boolean formula for which we want to obtain all the cutsets (combinations of failures that make the formula `true`). The sum of products[2] obtained (the list of cutsets or prime implicants) is then quantified to have a probability or a frequency of obtaining the realization of the Boolean formula which correspond to the realization of the master fault tree top gate.

Different classes of quantum algorithms were identified for the resolution of this problem. Recall that it is a generalization of a well-known SAT problem. Where we look for an assignment that satisfies a Boolean formula, while for the PSA case **all the assignments** have to be considered.

In the classical case, two main approaches are generally considered. The first is based on MOCUS like algorithms [J. B. Fussell and Marshall, 1974] (e.g [Relcon AB., 2003], [SAIC and EPRI, 1989], [Rauzy A., 2012]) and involves a number of approximation routines (for filtering cutsets with frequencies under some threshold, dealing with success events, ...). The second is based mainly on (Z)BDD construction ([Arboost Technologies, 2004], [EPRI, 2008], [Jung, 2009]).

In the quantum case, there are different approaches that were identified. The first is a direct implementation of Boolean gates to simulate a Boolean formula (Master fault tree) and then compute the probability of the top gate (section 7.1.1). The second is a combination of Grover search procedure in addition to a counting algorithm for specifying the number of steps in the rotation phase (section 7.1.3). Finally, another algorithm to calculate the minimum cut sets by tracing the fault trees back to the sequences and systems that were used to generate the master fault tree. The problem is then reduced to a problem of vertex separation in an s-t network [Balas and Souza, 2005] (section 7.3).

7.1. Boolean reduction

7.1.1. Direct implementation of logical gates

The very simple way to solve the reliability problem using a quantum circuit is to encode logical Boolean formulae using a rewriting of logical connector using quantum gates.

Given a fault tree or a Boolean formula $f(x_1, ..., x_n)$ where x_i are binary variables, the problem is to find the probability $P(f(x_1, .., x_n) = true)$, which is the sum of products of all the assignments that make $f(x_1, .., x_n)$ `true`.

If we consider the following example, Figure 7.1.1:

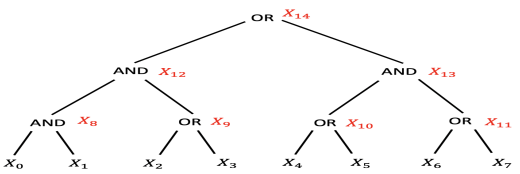


***Figure 12**: Example of a Boolean formula*

we can write the gates `AND` and `OR` using the following transcription in the form of a quantum gate (see Figure13).

[1] The success criteria are conditions that when met are sufficient for the success of a system/I&C mission or human action in a specific context regarding the dynamic of some accident sequence triggered by an initiator. Generally, the success criteria are elaborated regarding the "worst case" situation which is considered as an envelope of all the possible scenarios.

[2] The list of cut sets is called a sum of products in the sense that we have a big `OR` on a set of cut sets. A cutset is a combination of events which can be seen as a product of events when evaluating the probabilities.

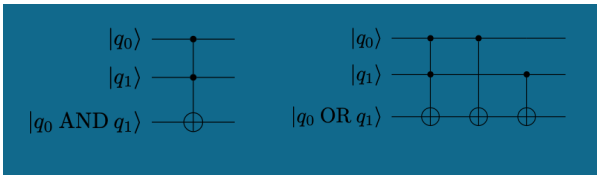


***Figure 13**: Quantum circuit for the AND and OR logic gates*

We can write a circuit that solves the formula using a number of qubits corresponding to the initial variables of the fault tree and a number of working qubits to store intermediate results of the logical connectors. The quantum circuit 14, represents the fault tree .

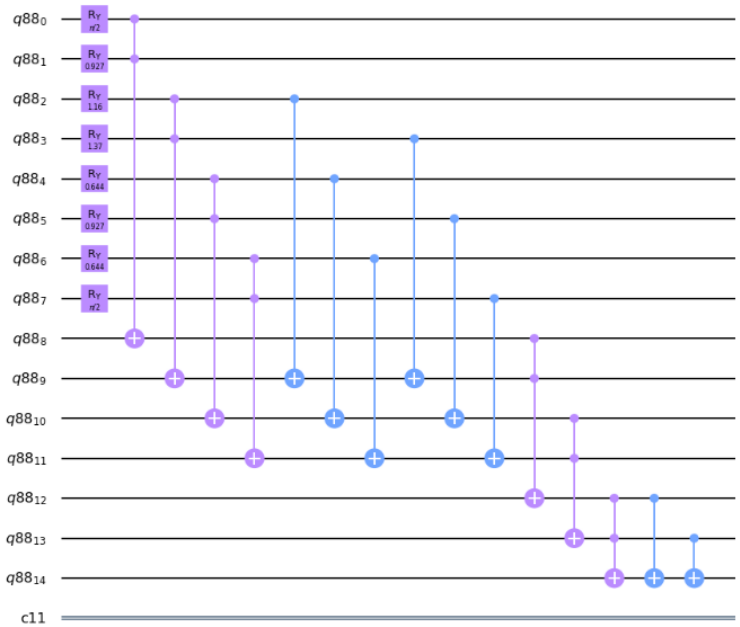


***Figure 14**: The quantum circuit for the fault tree 7.1.1*

The problem in this approach is the number of additional gates and qubits that have to be added at each logical gate.

7.1.2. Direct implementation and circuit optimization

The problem of the direct approach is that one has to consider replacing all the logical gates with their quantum counterpart. This turns out to be quite expensive in terms of additional gates and working qubits. We should note that the problem of optimizing quantum circuit can be handled in many ways. We can cite two approaches of interest, one is to consider reversible pebbling games strategies (cf. [Meuli et al., 2019b]) which is based on finding optimal uncomputing strategies that allow to clean up the circuit. Recall that quantum computing is reversible and thus one has to clean up (reset the qubits to their initial values) the input qubits using different strategies. The other is to consider ZX-Calculus ([Munson et al., 2019], [Duncan et al., 2020] and [van de Wetering, 2020]).

7.1.3. Reversible pebbling games and memory management

In the example proposed in [Meuli et al., 2019b], a quantum algorithm should do the following transformation:

$$|x_1\rangle\,|x_2\rangle\,|x_3\rangle\,|x_4\rangle\,|0\rangle\,|0\rangle \rightarrow |x_1\rangle\,|x_2\rangle\,|x_3\rangle\,|x_4\rangle\,|y_1\rangle\,|y_2\rangle \tag{7.1}$$

where

$$z_1 = A(x_2, x_3) \qquad z_2 = C(z_1, x_3) \qquad z_3 = B(x_3, x_4) \tag{7.2}$$
$$z_4 = D(z_3, x_3) \qquad y_1 = E(z_2, z_4) \qquad y_2 = F(x_1, z_1) \tag{7.3}$$

with A, B, C, D, E, F being some generic 2-input Boolean operations and z_1, z_2, z_3, z_4 the intermediate results. The transformation (7.1) could be represented by the *directed acyclic graph* (DAG) shown in Figure 15.

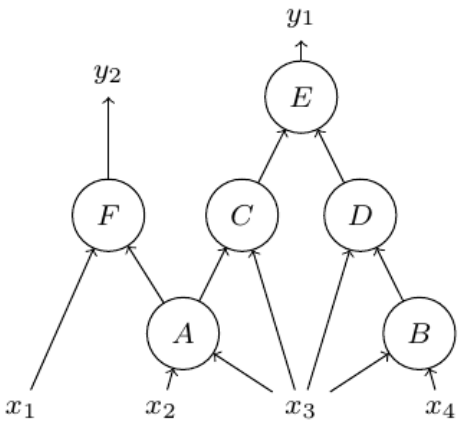


***Figure 15**: Transformation in the form of a DAG ([Meuli et al., 2019b])*

The question is to know which is the best strategy to design an optimal circuit to perform such transformation. There are two main parameters that can' t be reduced at the same time: the depth of the circuit and the number of necessary qubits.

Different strategies can be followed for uncomputing (cf. [Meuli et al., 2019b]). Bennett's strategy [Bennett, 1989] (Figure 16), which consists of computing all the operations in a bottom-up order, and then uncomputing the intermediate results in the reverse order, so that all the nodes have their inputs available. In this strategy, there is a reduction of the number of gates while the number of ancillary qubits is maximized.

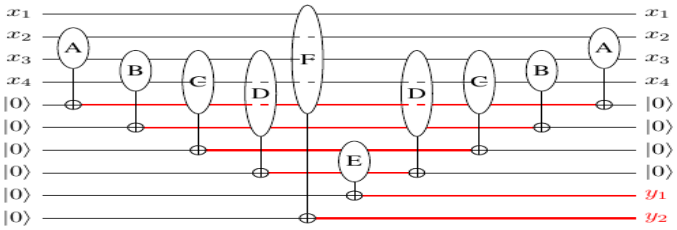


***Figure 16**: Bennett strategy (cf. [Meuli et al., 2019a])*

Another strategy is based on space-optimization by reordering the gates without any modification of the global operator (the order here is about the time when some gate is applied on the qubit line, cf. Figure 17).

Another strategy is based on space-optimization by increasing the number of gates (cf. Figure 18).

The problem of finding an optimal computing strategy is shown equivalent to the problem of reversible pebbling game. The latter is encoded as a SAT instance. The algorithm proposed in [Meuli et al., 2019a] consists of considering k pebbles and then trying if a pebbling strategy is possible with k pebbles[3]. If not, k is increased until reaching the optimal unpebbling strategy.

7.1.4. ZX-Calculus and circuit optimization

The ZX-calculus [Duncan et al., 2020] is a graphical language for reasoning about linear maps between qubits. The graphical objects of this language, called ZX-diagrams, consist of a set of generators or spiders

[3] A SAT solver is used in this step.

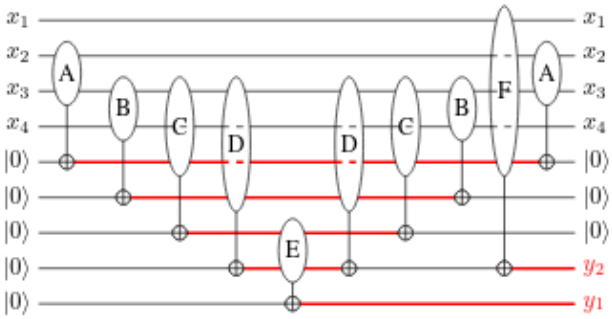


***Figure 17**: space-optimization by reordering the gates (cf. [Meuli et al., 2019a])*

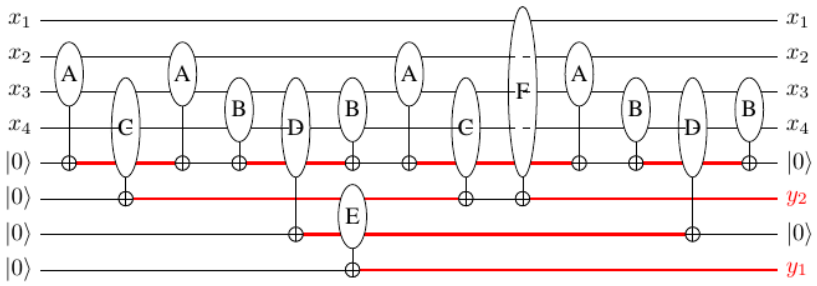


***Figure 18**: space-optimization by increasing the number of gates (cf. [Meuli et al., 2019a])*

that represent specific tensors. The topology of these graphic objects can be transformed according to some rules ([Duncan et al., 2020] or [Backens et al., 2017]) without changing the linear maps that they represent. ZX-diagrams can be seen as a generalization of quantum circuit and have been successfully used to optimize quantum circuits reducing their depth and avoiding useless quantum gates ([Munson et al., 2019], [Duncan et al., 2020] and [van de Wetering, 2020]).

In [Munson et al., 2019], ZX-calculus is used to optimize circuits that contain phase gadgets, which are families of multi-quantum gates, that occur naturally in many interesting quantum circuits. Indeed, this is done using pattern replacement which consists of recognizing a subcircuit of specific form and replacing it with an equivalent one (e.g. merging adjacent rotation gates acting on the same basis, cancelling operation-inverse pairs, and applying commutation rules). In addition to the fact that any sequence of single-qubit operations may be fused into a single unitary, for which an Euler decomposition can be computed[4].

1. Some simplification rules

   Spider-fusion This rule corresponds to the fact that the Z-spider (resp X-spider) represents an orthonormal basis, the computational basis. When two Z-spiders (X-spiders) touch, then can fuse together, and their phases add.

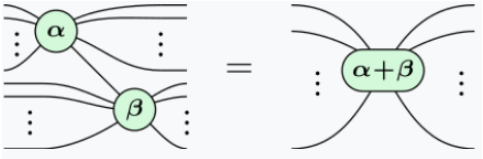


***Figure 19**: Z-Spider Fusion*

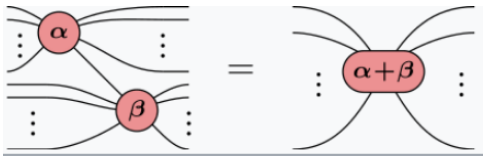


***Figure 20**: X-spider Fusion*

   Identity removal A phaseless arity 2 Z- or X-spider is equal to the identity. This rule states that the Bell-state is the same whether expressed in the computational basis ($|0\rangle$, $|1\rangle$) or the Hadamard-transformed basis ($\{|+\rangle, |-\rangle\}$) (See Figure 21).

[4]The idea is to use the Euler decomposition for the operator obtained as a product of the sequence. Each unitary operator on a single qubit can be written as $U = R_z(\alpha)R_y(\beta)R_z(\gamma)$, where α, β, and γ are the Euler angles (see Z-Y decomposition for single qubit Theorem 4.1 [Nielsen and Chuang, 2011]).

***Figure 21**: Identity removal*

Pivoting This property is equivalent to the possibility to express (one of) the Pauli matrices in terms of the Hadamard ones ([Duncan and Perdrix, 2014]). It is a local transformation of graphs. Given a graph G with an edge uv, $G \wedge uv$, the graph obtained by pivoting according to uv, consists in exchanging the two vertices u and v and in complementing the tripartite subgraph formed by (i) the common neighbors of u and v; (ii) the exclusive neighbours of u; and (iii) the exclusive neighbors of v (See Figure 22).

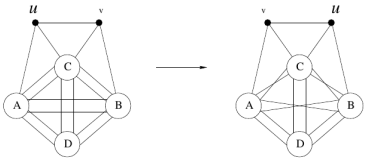


***Figure 22**: Pivoting ([Duncan and Perdrix, 2014])*

Local complementation Let G be a graph and let u be a vertex of G. The local complementation of G according to u, written as $G \star u$, is a graph which has the same vertices as G, but all the neighbors v, w of u are connected in $G \star u$ if and only if they are not connected in G. All other edges are unchanged (cf. [Duncan et al., 2020]) (See Figure 23).

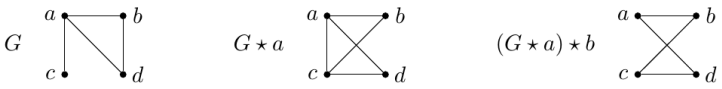


***Figure 23**: Local complementation (cf. [Duncan et al., 2020]).*

2. Example with Boolean formula

   The objective with the ZX-calculus here is the reduction of the T-count of a given circuit, that is of the number of T-gates it uses, these beeing costly to implement (see below for a definition of these gates). We designed a program to generate circuits corresponding to boolean formulae. These circuits can be used to obtain minimal cut sets which is the most complex task in resolving master fault trees. The remaining quantification can be done a posteriori as a sum of products. For this, we can either use the well-known classical approximation algorithms based on Sylvestre-Poincaré [Knuth D. E.,] or the Min Cut Upper Bound [Jung, 2015] when appropriate or even other approaches like rewriting cut sets in the form of binary decision diagrams [R., 1986] and then use Shannon decomposition [Bäckström and Ying, 2008].

   The circuits we obtain are then transformed into $qasm$ format to deal with pyzx transformations. Other more detailed tests are needed in this direction to deal with a variety of structure functions for boolean master trees. Note that this experiment is not optimal in the sense that the $R_y(t)$ is not implemented in the $pyzx$ library, but it was replaced with $SR_x(t)SZ$.

   In the following example, we show how this is used to optimize a circuit corresponding to a Boolean formula.

   Consider a boolean formula of the following form.

$$f = ((a \wedge b) \vee (c \wedge d)) \wedge ((t \vee f) \wedge (a \vee b)) \tag{7.4}$$

   Clifford gates are the elements of the Clifford group, a set of mathematical transformations which effect permutations of the Pauli operators. The notion was introduced by Daniel Gottesman and is named after the mathematician William Kingdon Clifford. (cf. wikipedia entry Clifford gates).

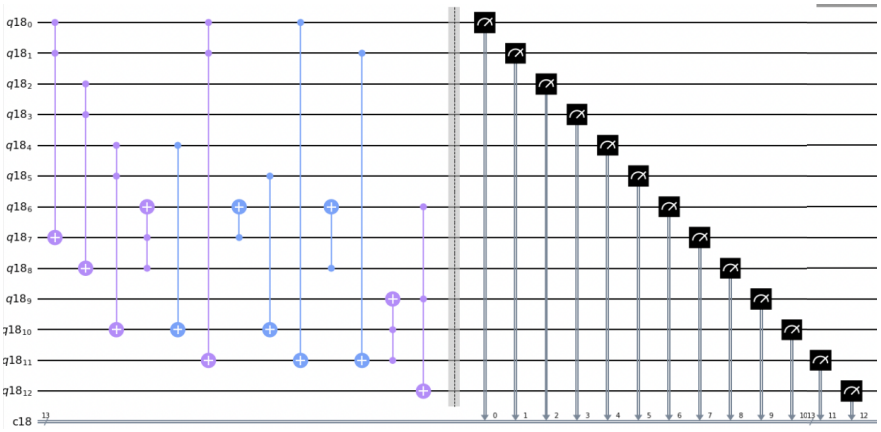


***Figure 24**: Direct implementation of the boolean formula 7.4*

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \text{ and } Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

T is an additional gate that makes the $Clifford + T$ set universal.

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$$

The circuit can be transformed in a ZX-diagram of the form of Figure (25) in ZH notation (cf. [Backens and Kissinger, 2019]). If we first convert the circuit to basic gates (i.e. Clifford + T), it can be seen as a pure ZX-diagram as in Figure 26. This transformation was done using the pyzx library ([Kissinger and van de Wetering, 2020]).

We consider the Boolean formula in 7.4, the corresponding zx-diagram is as follows:

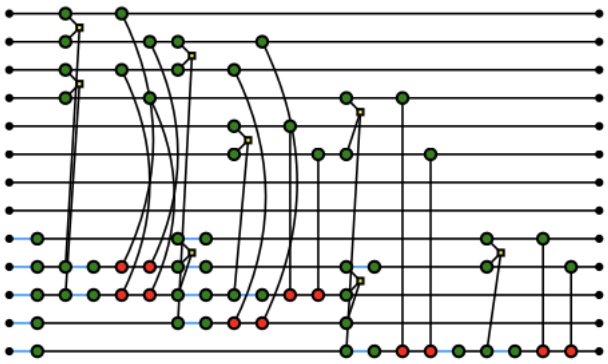


***Figure 25**: The zx-diagram of the boolean formula 7.4*

By default, the CCZ gates are drawn in ZH notation [Backens and Kissinger, 2018]. A conversion to a pure ZX-diagram can be obtained by converting the circuit to basic (i.e. $Clifford + T$) gates (as showed in Figure 26):

The circuit is composed of 13 qubits with 20 gates.

- 56 is the T-count
- 12 Cliffords among which
- 12 2-qubit gates (12 CNOT, 0 other) and
- 0 Hadamard gates.

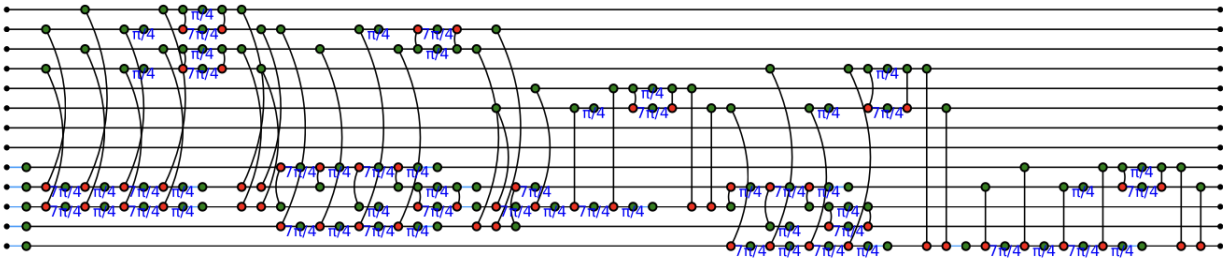


Figure 26: The pure zx-diagram after the transformation

There are 8 gates of a different type, and we can write it in terms of basic gates to get more accurate $Clifford + T$ statistics:

Indedd it is composed of 13 qubits with 132 basic Clifford+T gates.

- 56 is the T-count
- 76 Cliffords among which
- 60 2-qubit gates (60 CNOT, 0 other) and
- 16 Hadamard gates.

3. Optimizing ZX-diagrams

   PyZX contains many functions for optimizing circuits and ZX-diagrams.

   This above circuit has a T-count 56.

   The most basic simplification routine for ZX-graphs is $interior_clifford_simp$ which uses the simplification rules based on **spider-fusion**, **identity removal**, **pivoting and local complementation** until they cannot be applied anymore. There may be many iterations.

   $spider\_simp$: 24. 17. 13. 7. 4. 1. 6 iterations

   id_simp: 14. 1 iterations

   $spider\_simp$: 6. 1. 2 iterations

   $pivot\_simp$: 4. 1 iterations

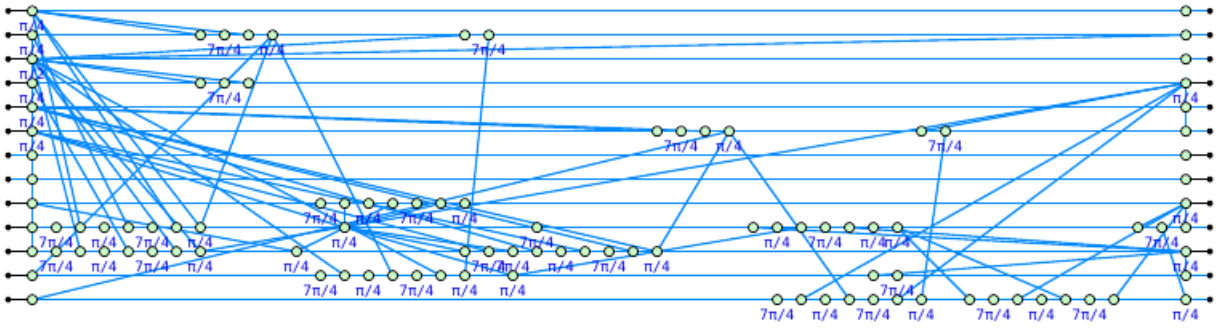


Figure 27: The zx-diagram after the iteration above

The optimized one has a T-count of 54 (See Figure 28).

Now the optimized T-count is 46.

The next step is to turn this graph back into a circuit, Figure 29.

Many other optimisation routines can be used, for instance to reduce the amount of Hadamard gates or to transform adjacent CNOT gates into SWAPs. These routines may depend heavily on the circuit and its structure.

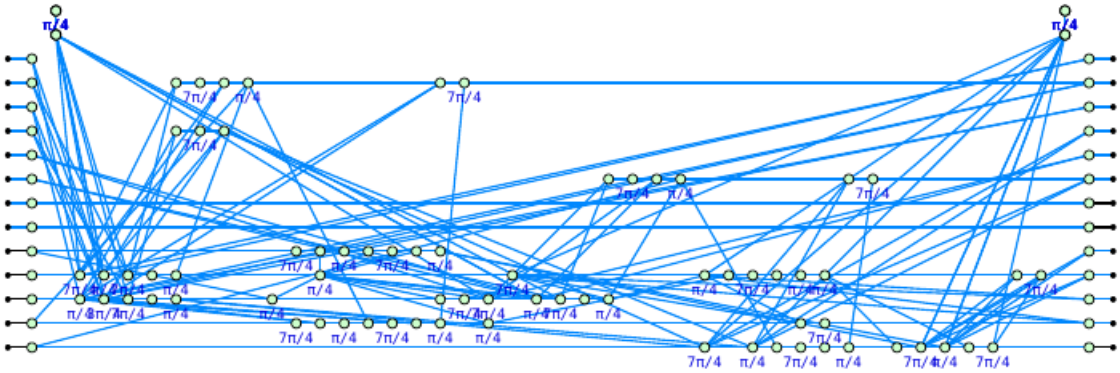


Figure 28: Optimisation intermediate step

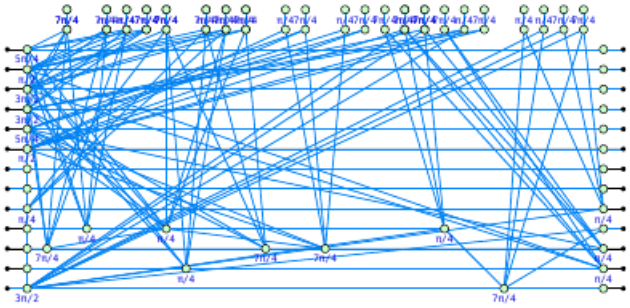


Figure 29: Optimized zx-diagram step

Figure 30 shows the obtained circuit from the optimization proccess.

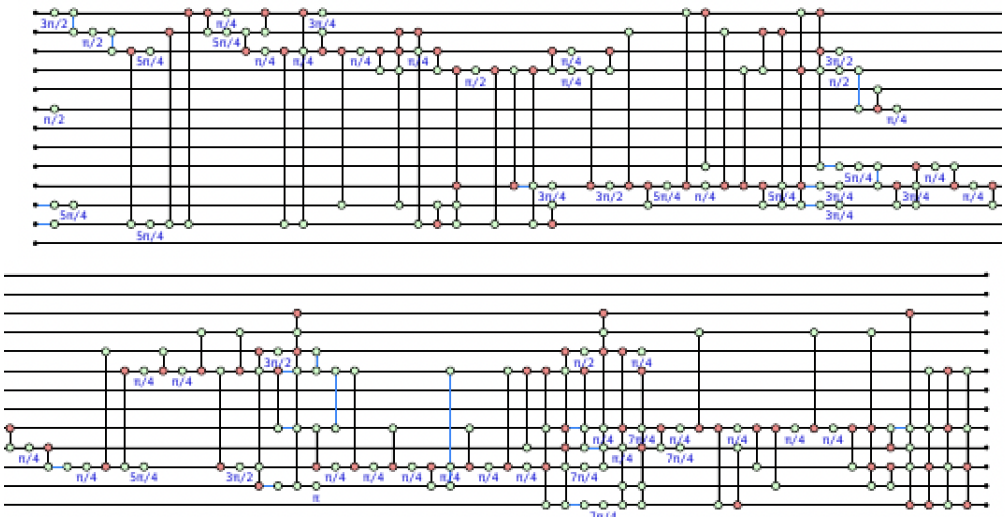


Figure 30: Complete Circuit

We have seen that there is a reduction of the T-count of the circuit. It may be of interest to do this exercise on a bench of examples from real life problems including instances of medium size with a high redundancy level[5].

[5]In terms of computational complexity this is representative of the most difficult instances we can encounter in rel life PSA models.

7.1.5. Evaluating Patterns

In this section we consider evaluating Boolean fault trees with special forms that can be obtained through different rewriting techniques or using an expert system for generating real-life system fault trees. We hope that using some logical pattern may help to reduce the need of additional quantum gates.

There are many questions:

1. How can we use the techniques of reverse game pebbling (section 7.1.4) and ZX-Calculus (section 7.1.3) to optimize corresponding circuits to Boolean formula in a direct optimization?
2. What method could be relevant for the problem at hand?
3. Are these methods equivalent ? Reverse game pebbling techniques may have impacts on both depth and gates number, while ZX-calculus seems elegant and could be much more efficient in reducing the gates number.
4. On the other side, regarding Boolean formulae or fault tree structure, the question is about which form of patterns we may find suitable for such circuits. The objective is then to generate fault trees following adequate patterns.

7.2. A Counting algorithm

In [Siciliano, 2018], it was shown that given an `AND-OR` Boolean formula, one can find all the assignments that satisfy it in $O(\sqrt{N})$, with N being the number of solutions. However, there is a precise number of queries to obtain the solution with optimal precision which is not known a priori.

A solution is to use *Quantum counting* algorithm (cf. [Brassard et al., 2000]) which is a quantum algorithm for efficiently counting the number of solutions for a given search problem.

It is based on the quantum phase estimation algorithm and on Grover's search algorithm. These two algorithms have been proved to provide a significant speedup over classical algorithms (exponential for quantum phase estimation and quadratic for Grover search).

In this section we present a joint work with ATOS [Hereramarti, 2020].

7.2.1. Quantum Fourier transform

Fourier transform has been very powerful in many areas of science; in addition to be an elegant tool, it often transformed a difficult problem into an easier one.

Similarly, to its classical version, the quantum Fourier transform acts on a quantum state $\sum_{i=0}^{N-1} x_i |i\rangle$ and maps it to the quantum state $\sum_{i=0}^{N-1} y_i |i\rangle$ where

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=1}^{N-1} x_j \omega_N^j k$$

which is the discrete Fourier transform of the amplitude x_k. This transformation is unitary and can be implemented as the dynamics for a quantum computer (cf. [Nielsen and Chuang, 2011] p. 217). Moreover, it maps the computing basis states $|k\rangle_{k\in\mathbb{Z}_d}$ to another basis as follows:

$$|w_k\rangle = \mathcal{F}\,|k\rangle = \frac{1}{\sqrt{d}} \sum_{l=0}^{d-1} e^{2\pi i k l/d}\,|l\rangle$$

The new basis $|w_k\rangle_{k\in\mathbb{Z}_d}$ is called the quantum Fourier basis.

7.2.2. Quantum phase estimation

Quantum phase estimation is a one of the important routines of quantum computing as it is convened in many quantum computing algorithms (e.g. state separation [Zhao et al., 2019], accelerating variational quantum eigen solver [Wang et al., 2019], tensor principal component analysis [Hastings, 2020], . . . etc).

Given a unitary operator U with $|\psi\rangle$ an eigenvector and $e^{2\pi i\theta}$ its corresponding eigenvalue ($U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$), since θ is not known a priori, the quantum phase estimation algorithm will serve to estimate it.

The quantum phase estimation procedure uses two registers. The first register contains t qubits[6] initially in the state $|0\rangle$ we call this the counting register. The second register will serve to store the eigenvector.

In the computational basis, we store numbers in binary form using the states $|0\rangle$ and $|1\rangle$, but in the Fourier Transform basis, we store numbers using different rotations around the Z-axis. This is used by the quantum phase estimation algorithm to write the phase of U (in the Fourier basis) to the t qubits in the counting register, then again using the inverse quantum Fourier transform to go back in the computational basis to be measured.

Recall that to represent a number in binary form we use the following scheme:

$$a_0 \times 2^0 + a_1 \times 2^1 + a_2 \times 2^2 + a_3 \times 2^3 + a_4 \times 2^4 + ... + a_n \times 2^n$$

Where coefficients a_0, a_1, a_2 are bits and, the power i corresponding to a bit a_i is called its weight.

For instance, we can encode 5 as in Figure 31.

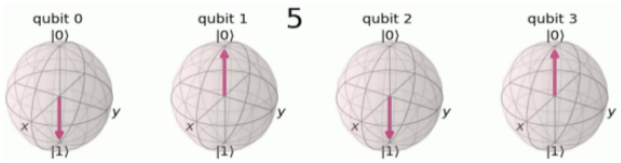


***Figure 31**: 5 as a binary string*

There is another way of coding numbers using "phases". For instance, we can encode 5 as in figure 32.

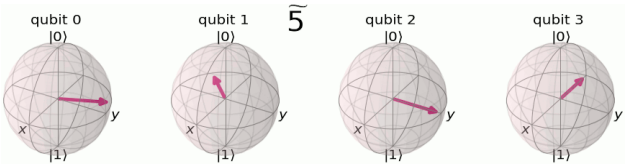


***Figure 32**: 5 as a phase superposition*

Indeed, to count a number, x between 0 and 2t, we rotate this qubit by $x2t$ around the z-axis. For the next qubit we rotate by $2x2t$, then $4x2t$ for the third qubit.

Therefore, in the Fourier basis, the topmost qubit completes one full rotation when counting between 0 and 2t.

When we use a qubit to control the U-gate, the qubit will turn (due to kickback) proportionally to the phase $e^{2i\pi\theta}$. We can use successive CU-gates to repeat this rotation an appropriate number of times until we have encoded the phase θ as a number between 0 and 2t in the Fourier basis.

Then we use $QFT^\dagger$ to convert this into the computational basis before measurement (Cf. Figure 33).

[6] The choice of t depends on the number of digits of accuracy we wish to have in our estimate for θ, and with what probability we wish the phase estimation procedure to be successful (cf. [Nielsen and Chuang, 2011]).

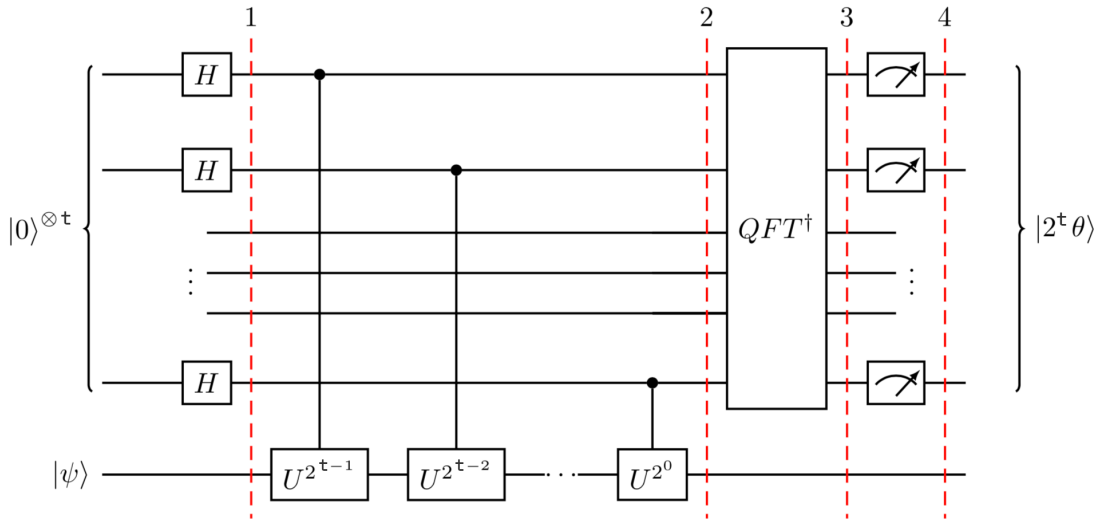


Figure 33: Quantum Phase Estimation Circuit (src Qiskit documentation)

7.2.3. Grover search

Grover's algorithm allows searching an element in a set of unstructured data. In Figure 34 it is considered to begin the algorithm in the starting state $|\alpha\rangle$ and the objective is reaching the final state $|\beta\rangle$.

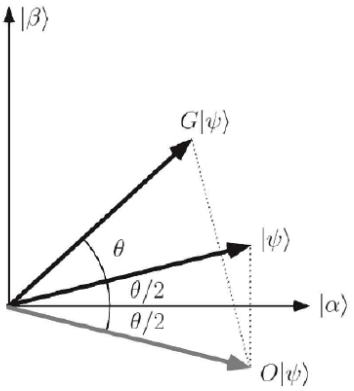


Figure 34: Geometrical representation of the effect of Grover search. The state $|\beta\rangle$ is assumed to be the solution of a given problem.

Each step of the algorithm consists in 2 rotations, which practically is implemented by working on the state's amplitudes. In particular, the aim is the maximization of the "good" state's amplitude. The method, generalized in [Brassard et al., 2000] and called quantum amplitude amplification, maximizes such amplitude, first a rotation around the initial state is done by a given angle $\frac{\theta}{2}$ and then a second rotation, that will be performed by the so called *Grover operator* which will rotate around the new state $|\psi\rangle$ by a full angle θ as can be seen in Figure 34. The key parameter here is the angle of rotation θ: it is needed it to be the largest in order to reach with the smallest number of operations the final state but *without* exceeding it. The problem is indeed that an excessive rotation would lead the amplitude to decrease, thus decreasing the probability of having a good result.

If we consider $|\alpha\rangle$ as the superposition of vectors that are not solutions of the problem we deal with, and $|\beta\rangle$ the superposition of the vectors that are solutions of our problem. A Grover iteration is the space spanned by $|\alpha\rangle$ and $|\beta\rangle$ is a rotation of an angle θ (cf. Figure 34). This rotation can be described as a matrix with eigenvalues $e^{i\theta}$ and $e^{i(2\pi-\theta)}$.

Definition 1. *The Quantum Oracle is a black box used to estimate a function using qubits. It allows transforming a*

system from a quantum state $|x\rangle$ to a state $|f(x)\rangle$, by the evolution of quantum states.

$$O|x\rangle = |f(x)\rangle$$

As in the case of quantum phase estimation, Grover search is based on the possibility of building oracles to implement the U^{2x} (or at least polynomial time subroutines to solve some yes/no questions). There are many ways to implement the oracle effects [Nielsen and Chuang, 2011]. In the *Quantum Learning Machine* of Atos *QLM* the "phase oracle" that changes the vector signs is considered.

The relationship between the angle θ and the number of solutions M is determined by the formula $sin^2(\theta/2) = M/N$. (N being the number of all potential candidates).

7.2.4. Quantum Counting

Once the number of solutions M given, the Grover algorithm can be successful in finding all the solutions of the PSA/reliability problem.

The corresponding circuit could be as in Figure 35 (cf. [Hereramarti, 2020]).

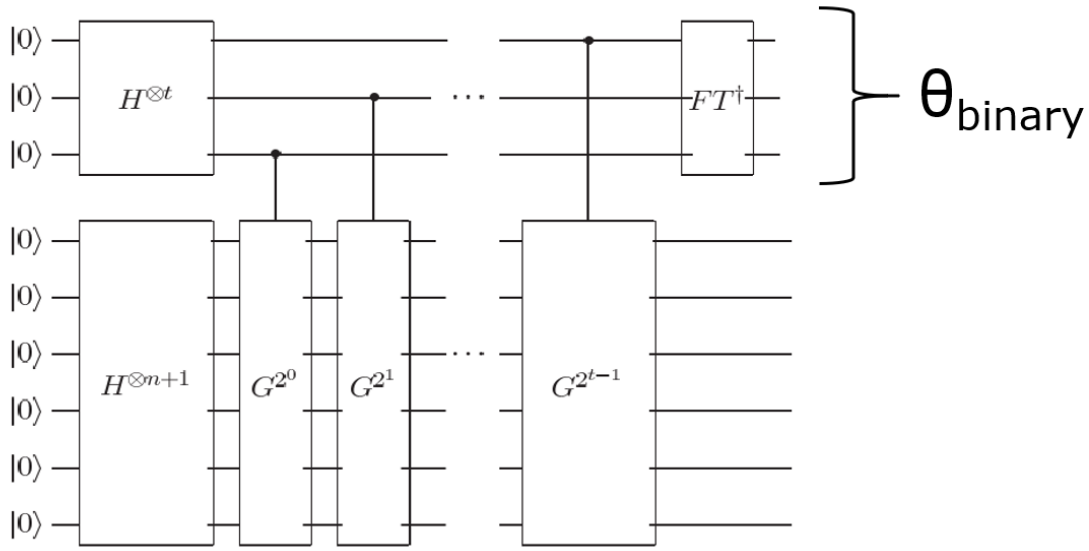


***Figure 35**: Grover algorithm to find all the solutions of the PSA/reliability problem*

An experimentation using a truth tables as oracles on some instances (`BSU`, `LIFT` and `THREE_MOTOR` from the OPEN PSA website) showed interesting results. It particularly pointed out the fact that for instance while Grover algorithm needed 50 iterations to maximize the probability of finding the solutions, the classical approach needed 8000 evaluations (for more details see [Hereramarti, 2020]).

To go further we need to find an efficient way to implement the oracle without any prior knowledge of truth tables.

Two main alternatives were identified in [Hereramarti, 2020]:

- Define the oracle as a Boolean function instead of a truth table.
- Study the noise effects, and in particular the compromise between the number of bits in the binary precision and the error introduced by the increase of the noisy gates number.

Some solutions could be provided by the QLM machine in a near future.

7.3. A quantum algorithm for s-t network vertex separation

All the calculations made within the framework of the PSA come from the model's event trees representing accident sequence diagrams. These diagrams represent all the possible scenarios initiated by an initiating event. These scenarios are generally represented in the so-called Master Faut Tree. In [Jeffery and Kimmel, 2017] we can see that some Boolean formulae can be solved for satisfiability (SAT problem) using a reduction to a problem of deciding in a graph if there is a path between a source s and a target t for which there is an algorithm with $O(\sqrt{(N)})$ complexity.

The approach we adopt here is to consider upstream modeling, it is to say the structure of the scenarios that we will represent as a graph where the failure or success of the system is represented by graphs representing the topology of the systems that are used in the different parades (cf. [Hibti, 2013]).

For example, in the small pumping system in Figure 36, it can be represented in the form shown in Figure 37.

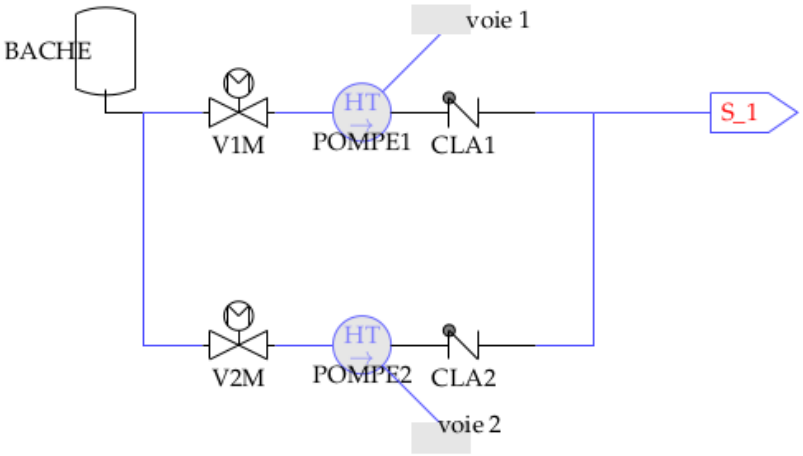


***Figure 36**: Small pumping system*

If we consider the traditional failure modes for the components in question, the problem of getting cut sets corresponding to the undesired event Top (that represents the failure to feed the node S_1) can be solved by identifying the set of paths that can join two specific nodes *origin* and *destination* in a special graph representing different macro-components of the system in question. The failure of the feed S_1 is detected by finding directly all the combinations of events in the graph that can stop the flux to S_1 or by finding the inputs (cuts) that can realize the Boolean formula of the master fault tree of the system:

$$f = BACHE \vee ((V1M \wedge POMPE1 \wedge CLA1) \vee (V2M \wedge POMPE2 \wedge CLA2))$$

For simplification reasons, we consider that there is no electrical or cooling supply of the different components. Moreover, we do not consider Common Cause Failures (CCF).

The *generic cut-sets* represent all the sets of edges or vertices that if removed may lead to cut the corresponding graph in tow disconnected sub-graphs with the sink (s) in the first and target (t) in the second.

Indeed, the generic cut-sets are:

- the failure of the tank `BACHE`
- the failure of the macro component `C1` and the macro component `C2`

Therefore, the minimal cut-sets can be obtained as the set of all products $B \times F1 \times F2$ where B is the tank failure, F1 is the set of failure modes of the components of C1 and F2 is any failure mode of the components of C2.

The problem is then reduced to a problem of finding cut-sets in an s-t graph.

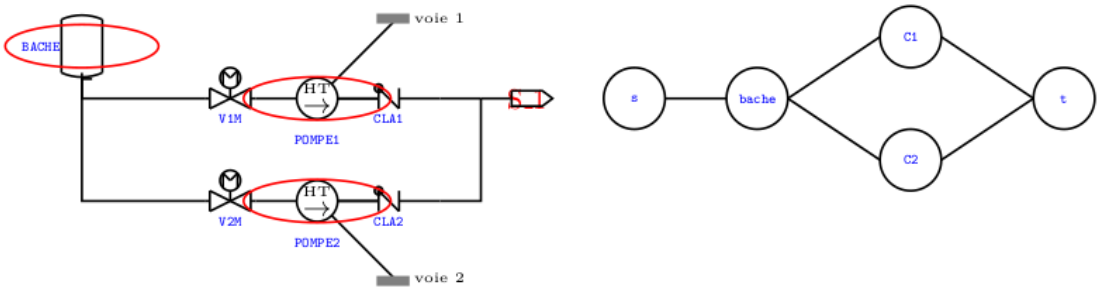


Figure 37: Representation in the form of a graph

We propose an algorithm based on a movement strategy where it uses movement oracles to build a quantum superposition containing all these minimal cuts (see the section 7.3.4 for more details).

The first question that comes up here is how can we represent all the vertex sets with quantum qubits? On the other hand, for a graph of n vertices, we will find 2^n different subsets of these vertices. In the quantum setting, with n qubits, we can represent 2^n possible states. In both cases with n elements we have 2^n possibilities, so we represent the subsets by the quantum states of these n qubits. To do this, we use for each vertex of the graph a qubit, and each state of these qubits represents a subset of the vertices.

7.3.1. Oracle of movement

Consider the graph of Figure 38.

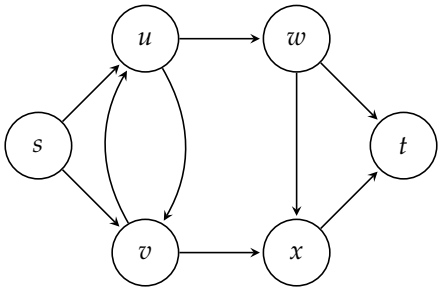


Figure 38: Directed graph

The movement of a vertex is given by the set of the successors starting from the vertex of the movement. The movement of a vertex v is defined by this general formula:

$$Mov(v) = Succ(v)$$

with $Succ(v)$ represents the set of the successors of the vertex v.

If we have a set of vertices S and we need to make a movement by one vertex v of these vertices, we will simply replace the vertex v by its successors in the set S to obtain the result of the movement $Mov_S(v)$. The movement of a vertex $v \in S$ is defined by this general formula:

$$Mov_S(v) = \{S \setminus v\} \cup Succ(v)$$

As a first example, based on the graph in Figure 38 and assume that the starting set S contains just the source vertex s, $S = \{s\}$. The movement of s it is the replacement of the vertex s by their successors $\{u, v\}$ in the starting set S, therefore:

$$Mov_S(s) = Succ(s) = \{u, v\}$$

We consider another example using the same graph in Figure 38 we take the current set $S = \{u, v\}$, in order to obtain the movement of $u \in S$ we simply replace u by its successors set $\{v, w\}$. So the new set after the

movement is :

$$Mov_{\{u,v\}}(u) = \{\{u,v\} \setminus u\} \cup Succ(u) = \{v\} \cup \{v,w\} = \{v,w\}$$

In the quantum context, to apply these movements we use oracles of movements. Which takes the current subset as an input quantum state and provides the movement subset in the output.

For some fixed vertex v and some subset of vertices S, such as $v \in S$, firstly, we represent the subset S with the quantum state $|\psi_S\rangle$. To apply the movement of v we give the quantum state $|\psi_S\rangle$ to the oracle as an input, and in the output of the oracle we will find a quantum state representing the subset $(S \setminus v) \cup Succ(v)$.

More precisely, let us consider the current set of vertices defined as the union of two subsets $S = S_1 \cup S_2$, this union can be represented by the following quantum superposition $|\psi_S\rangle = \alpha_1 |S_1\rangle + \alpha_2 |S_2\rangle$ where the state $|S_1\rangle$ represent the subset S_1 and the state $|S_2\rangle$ represent the subset S_2. We consider some fixed vertex $w \in S_1$ and define the movement of $w \in S$ as follows:

$$|\psi_{out}\rangle = \frac{\alpha_1}{\sqrt{2}} |S_1\rangle + \alpha_2 |S_2\rangle + \frac{\alpha_1}{\sqrt{2}} |S_3\rangle ,$$

which combines the two input subsets S_1, S_2, and the new subset of the movement S_3. And, S_3 is defined as follows:

$$S_3 = Mov_{S_1}(w) = \{S_1 \setminus w\} \cup Succ(w)$$

If the vertex of the movement is not in the input subset, $w \notin S_1 \cup S_2$, the oracle will not change the input superposition.

In order to write the general formula of an oracle of movement O, let us consider the input quantum state $|\psi_S\rangle$ which represents some subset of vertices S.

To provide the output quantum state of the movement $|\psi'_S\rangle$, we use this general formula :

$$|\psi'_S\rangle = O |\psi_S\rangle \text{ and } |\psi'_S\rangle = \sum_{e \in S} \alpha_e f(e) |e\rangle$$

where

$$f(e) = \begin{cases} 1 & \text{if } e \text{ is the start state} \\ 1 & \text{if } e \text{ is the movement state} \\ 0 & \text{otherwise} \end{cases}$$

And

$$\begin{cases} \alpha_e = 0 & \text{if } f(e) = 0 \\ \sum_e \alpha_e = 1 \end{cases}$$

The general circuit of oracles is as follows in Figure 39:

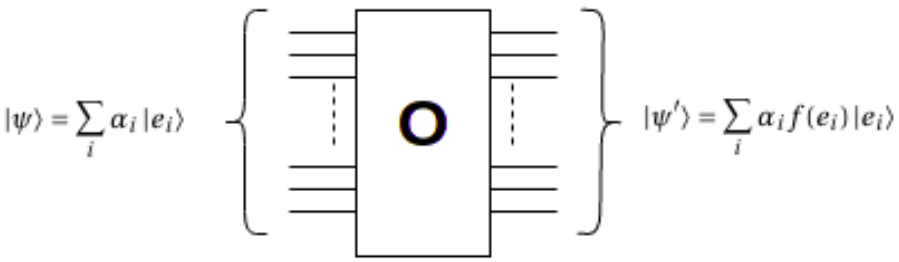


***Figure 39**: General circuit of oracles*

To represent an oracle with a simple quantum circuit, we should add two more control qubits $|c_0\rangle$ and $|c_1\rangle$.

The first control qubit $|c_0\rangle$ is used to check if the vertex of the movement exists in the input set. The qubit $|c_0\rangle$ will be in state $|1\rangle$, if the qubit corresponding to the vertex of the movement is in state $|1\rangle$, and in $|0\rangle$ if

not. For this, we use the C-X gate with the qubit corresponding to the vertex of the movement as a control and the qubit $|c_0\rangle$ as a target.

If the vertex is in the input set, we add another set to the set of cuts. Or, we can say if the qubit of the vertex is in the state $|1\rangle$ we add a new state to the input superposition. To do this, we use the C-H gate with the qubit $|c_0\rangle$ as a control and the target qubit is the qubit of the vertex of the movement. After that, we use a C-X gate to apply the movement on the new set.

To add all the successors of the movement vertex to the new set, we use the sub-circuit in the Figure 40 which allows inverting the qubit of the successor in the state $|1\rangle$ if it is in the state $|0\rangle$ and to do nothing if it is in the state $|1\rangle$.

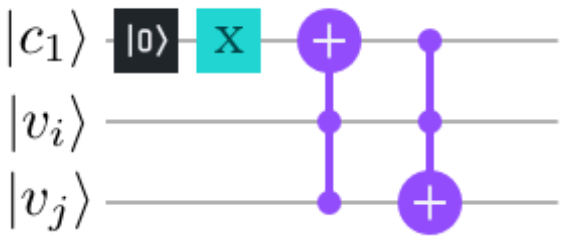


***Figure 40**: Sub-circuit of movement from v_i to v_j, this sub-circuit has three qubits as an input: $|c_1\rangle$ for the control. $|v_i\rangle$ represent the vertex of the movement and $|v_j\rangle$ represent the successor vertex.*

7.3.2. Algorithm description

Let $G = (V, E)$ be a directed graph, where V is the set of vertices such as $|V| = n$ and E the set of edges, the source of the graph is the vertex S and the sink is the vertex T. The first step of the algorithm, is preparing the necessary number of qubits to represent all possible subset of vertices of the graph. The size of the set V is n, so we use n quits.

At the beginning all the n qubits are initialised in the state $|0\rangle$, so the quantum state it initialised to $|\psi\rangle = |0 \dots 0\rangle$. With the first qubit corresponding to the source vertex S, the second qubit for the second vertex, until the last qubit for the last vertex (the sink vertex T). There are only zeros in the state $|\psi\rangle$, which means all the vertices are absent in the cut represented by $|\psi\rangle$.

$$|\psi\rangle = |tv_n \dots v_1 s\rangle = |00 \dots 00\rangle \iff \psi = \{\}$$

Therefore, to start with a set ψ containing only the input vertex S, we apply the not gate X on the first qubit, which gives us $|\psi\rangle = |0 \dots 01\rangle$ as a result, with the qubit $|s\rangle$ is in the state $|1\rangle$.

In the second step, for each vertex, we have an oracle of movement, so we call all these oracles of movement.

The first oracle corresponding to the movement of the vertex S to its successors:

$$|\psi_1\rangle = O_1 |\psi\rangle = \alpha_1 |\psi\rangle + \alpha_2 |Mov_\psi(S)\rangle$$
$$|\psi_1\rangle = \alpha_1 |\psi\rangle + \alpha_2 |Succ(S)\rangle$$

After this iteration, the oracle O_1 adds to the superposition the state $|Succ(S)\rangle$ which represent the first cut of the graph.

After that, we apply all the remaining oracles:

$$|\psi_{fin}\rangle = O_n O_{n-1} \dots O_2 |\psi\rangle$$

Each one of these oracles adds a number of states to the superposition, which means its adds a number of cuts to the set of cuts represented by the superposition.

$$|\psi_{fin}\rangle = \alpha_1 |cut_1\rangle + \cdots + \alpha_k |cut_k\rangle$$

After the n oracles, in the output superposition $|\psi_{fin}\rangle = \sum_i \alpha_i |Mincut_i\rangle$, we find all the possible cuts represented by the states $|Mincut_i\rangle$. Finally, we use a simple classical filter to eliminate non-minimal cuts.

Algorithm 1: All the possible minimal cuts

Input : Graph $G = (V, E)$, with $n = |V|$ is the number of vertices of the graph, source vertex S, sink vertex T

Output: Min-cuts Cs

Start:

Initialized n qubits,

$$|\psi_0\rangle = |0, \ldots, 0\rangle$$

Apply the not gate X in the first qubit which represents the source S

$$|\psi_1\rangle = |0, \ldots, 0, 1\rangle$$

Make the movement of S we apply the oracle O_s

$$|\psi_2\rangle = O_s |\psi_1\rangle$$

for *each* $v \in V$ *and* v *!=* S *and* T *is not a successor of* v **do**

$$|\psi_{i+1}\rangle = O_v |\psi_i\rangle$$

Cs = measured $|\psi_{n-1}\rangle$ and eliminate non-minimal cuts.

return Cs

Algorithm 1 represents the steps to generate the quantum circuit to find all the possible minimal cuts.

7.3.3. Complexity Analysis

Suppose we have a graph $G = (V, E)$, with V the set of vertices and E the set of edges such as $|V| = n$ and $|E| = m$. In the classical case Kloks and Kratsch propose in [Ton and Dieter, 1998] an algorithm to list all these minimal cuts with a polynomial complexity for each found cut. In the quantum case, our algorithm is able to find all these minimal cuts with linear complexity. It uses n qubits to represent all the possible states of the graph and 2 auxiliary qubits for control. To build the n oracles of movement we need n gates C-H, $2n - 2$C-X gates , $m + 2$ X gates and $2m$CC-X gates.

7.3.4. Test of the algorithm

In this section, we present a test case of our algorithm. Let us take the directed graph $G = (V, E)$ represented in the Figure 41.

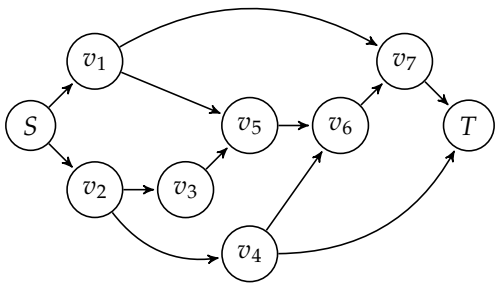


***Figure 41**: Oriented Graph of 9 nodes*

In this graph, we want to find all the minimal cuts that can stop the flux between the source S and the terminal T. The graph has $n = 9$ vertices, which leads us to use 9 qubits to represent all possible combinations of these vertices.

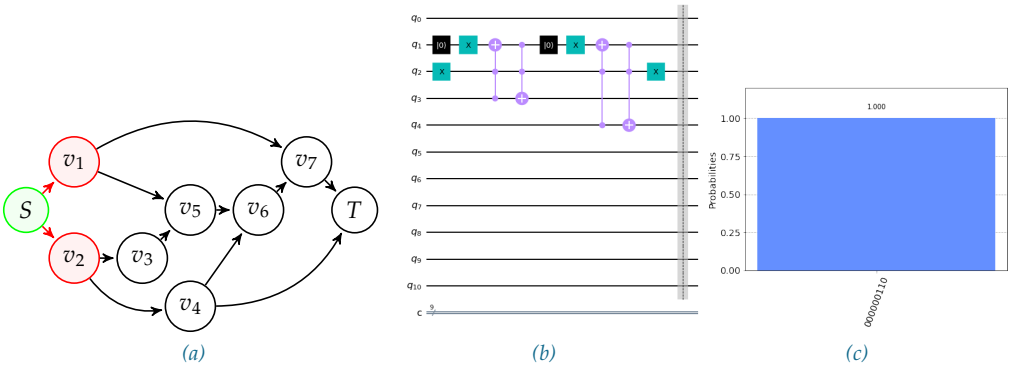
The source S is represented by the first qubit $|v_0\rangle$ and each vertex $v_i, i = 1, \ldots, 7$ is associated to the qubit $|v_i\rangle$, $i = 1, \ldots, 7$ and the terminal T is associated to the last qubit $|v_8\rangle$.

Each state $|C_i\rangle$ of the superposition $|v_8v_7v_6v_5v_4v_3v_2v_1v_0\rangle = \sum_{i=0}^{2^9-1} \alpha_i |C_i\rangle$ represents a combination of the vertices, we say that the vertex v_i belongs to a combination of vertices C_i if the corresponding qubit $|v_i\rangle$ of the state $|C_i\rangle$ is in the state $|1\rangle$. For example, $C = \{v_1, v_2\}$ is represented by $|000000110\rangle$.

To begin with the source of the graph we start with the state $|\psi_0\rangle = |000000001\rangle$.

Now let's suppose that all the successors of the source $S = v_0$ are broken down, then there isn't any way to go to the next vertices of the graph, so the combination of the successors of the vertex $S = v_0$ is a cut, moreover, if one of these successors is in good condition, we will find a way to go to the next vertices. Consequently, the combination of successors of $S = v_0$ is a minimal cut.

Then, to find this first minimal cut, we apply the first oracle of the movement on the source S to its successors v_1 and v_2. That is, we take $|\psi_0\rangle$ as an input and we apply the movement to go to the state $|\psi_1\rangle = |000000110\rangle$. Figure 42 represents the movement, oracle of the movement and the results of the movement of S.

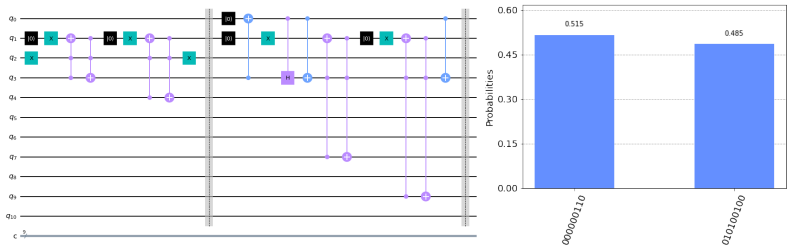


(a) (b) (c)

***Figure 42**: The movement of S towards the two successors v_1 and v_2 represented in (a) can be performed by the oracle (b). (c) The result of the execution gives the state $|000000110\rangle$ which represents the first cut $\{v_1, v_2\}$.*

Here $|\psi_0\rangle$ is X a cut, so we use only the not gate to transform this state to $|\psi_1\rangle$.

Now, suppose that one of the successors v_1 and v_2 is in a good state (is not broken), then there is a way to go to the following vertices. For example, if vertex v_1 is not broken, so we can find a way to the terminal through the successors of v_1. If the successors of v_1 are broken, we can't find a path to the terminal. Therefore, the combination $C_2 = Succ(v_1) \cup C_1 \setminus \{v_1\}$ is a cut, and if one element of C_2 is not broken, we find a path to the terminal, so C_2 is a minimal cut. Then, if we apply the movement of v_1 in the state $|\psi_1\rangle$, we find a new minimal cut containing the successors of v_1 and the vertex v_2.



(a) (b)

***Figure 43**: (a) The second oracle starts after the first barrier in the circuit. (b) The result of the execution gives two states: $|000000110\rangle$ represents the input and $|010100100\rangle$ represents the new cut after the movement. Note that the second oracle in (a) uses the Hadamard gate to add the new state into the superposition.*

Here, the oracle uses the Hadamard gate to keep the first cut and add the new state corresponding to the new cut. Figure 43 shows the circuit of the second oracle and the results of the movement.

In step k, the O_k oracle is applied to the output superposition of the step $k-1$, which adds new states to $|\psi_k\rangle$, if the qubit v_k is in the state $|1\rangle$ for each state of the superposition $|\psi_{k-1}\rangle$.

The circuit for graph in Figure 41 is presented in Figure 44 and the results of the execution of this circuit is presented in Figure 45.

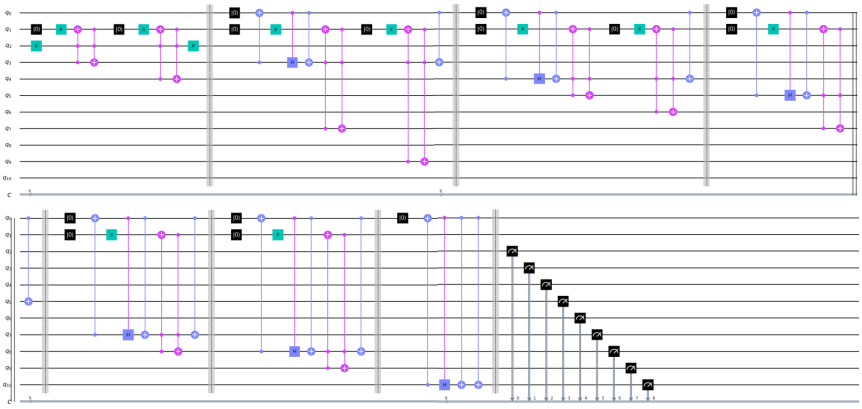


***Figure 44**: The circuit reserves 11 qubits: 9 to represent all possible failure states, 2 for the control. And also it uses 7 oracles of movements separated by vertical separators. Each oracle represents the movement of a vertex. At the end of the circuit, we measure the 9 qubits to find the superposition that represents all the minimal cuts.*

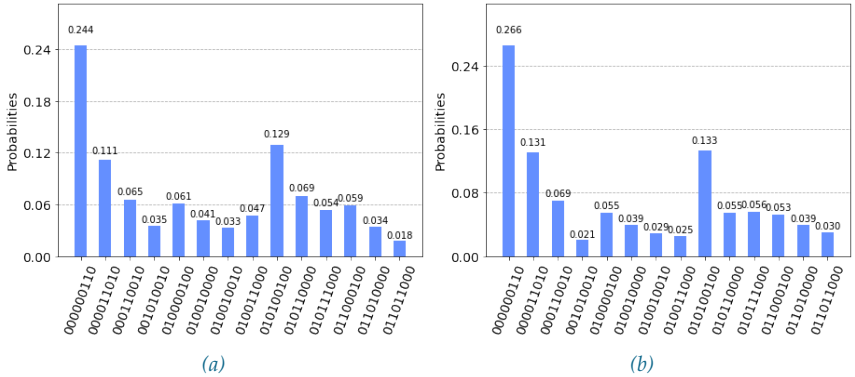


***Figure 45**: The histograms represent the superposition of output $|\psi_{final}\rangle$. (a) The result of the execution in the IBM Qasm simulator. (b) The result of the execution in the IBM Q 16 Melbourne*

Finally, we eliminate the non-minimal cuts. For that, for each (i, j) eliminate the cut C_j if $C_i \subset C_j$.

To check the results we visualized each state of the superposition in figure 45 in an independent graph in figure 46, with red color if the vertex qubit in the state $|1\rangle$ (present in the minimal cut) and black if it is in the state $|0\rangle$.

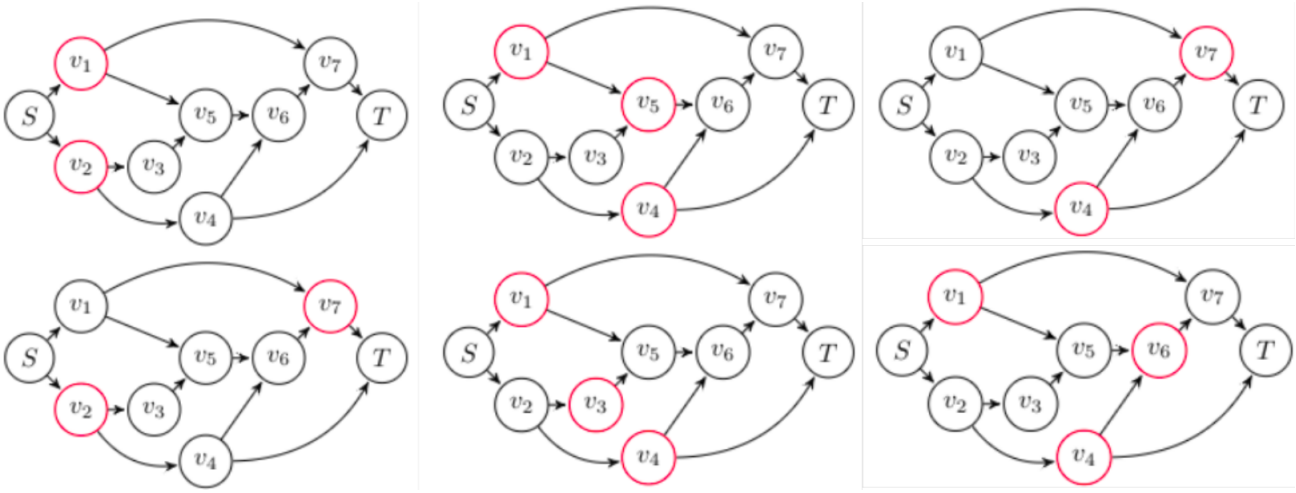


***Figure 46**: Illustration of the results on the graph*

7.4. Quadratic Unconstrained Binary Optimisation

Quadratic Unconstrained Binary Optimization is an optimization problem that aims to find a binary vector x that is minimal with respect to a quadratic polynomial $f_Q(x) = \sum_{i=1}^{n} \sum_{j=1}^{i} q_{ij} x_i x_j$ over binary variables $x_i \in \mathbb{B}$ ($\mathbb{B} = \{0, 1\}$) for $i \in [n]$ and coefficients $q_{ij} \in \mathbb{R}$ [7].

In [Smelyanskiy et al., 2012], an encoding of fault tree evaluation into QUBO is done by encoding the relation of the logical gates in the following manner in Table 5:

Gate	Ising encoding
2-input AND gate	$H_{AND}(y, x_1, x_2) = y + x_1 + x_2 + x_1x_2 - 2yx_1 - 2yx_2$
2-input OR gate	$H_{OR}(y, x_1, x_2) = 3y + x_1x_2 - 2yx_1 - 2yx_2$.
Gates with 3 or more inputs	$H_{OR}(y, x_1, x_2, x_3) = H_{OR}(y, z, x_3)$ where z with encoding $H_{OR}(z, x_1, x_2)$
3-input majority gate	$H_{MAJ}(y, x_1, x_2, x_3) = 3y - 2y(x_1 + x_2 + x_3) + x_1x_2 + x_1x_3 + x_2x_3$

***Table 5**: An encoding of fault tree evaluation into QUBO*

The complete cost function, that will be translated to a problem Hamiltonian that encode ab optimization problem (to implement on Ising graph) may be written in the following form [Smelyanskiy et al., 2012]:

$$H_{fault-tree} = A \sum_{j=1}^{M} H^j_{gate} + B(1 - z_0) + \sum_{i \in \mathbb{B}} w_i z_i$$

where the first sum is over all gates, using an appropriate expression (e.g. HAND , HOR, or HMAJ) with appropriate events z_i substituted for $y, x_1, ..., x_k$.

For unweighed minimum cut we set $w_i = 1$ for all basic events while choosing $A > 3N$ and $B > 3MA$ ensures that in the global minimum $z_{0=1}$ and all gate constraints are satisfied [Smelyanskiy et al., 2012].

[7] $[n] = j \in \mathbb{N}; 1 \leq j \leq n$

8.Quantum Benchmarks

In this section, we evaluate the approaches proposed in NEASQC Deliverable 6.4 and in Section 6.7 of present document. We performed several tests on small problems based on the number of qubits available to us at the time. To perform these tests, we used four small fault trees: Theatre (5 nodes), SmallTree (7 nodes), BSCU (15 nodes) and LIFT(27 nodes) and in addition to that 7 faults tree of more than 27 nodes. The fault trees are represented in the following after that we show the test results and the comparative results between the two approaches, the section will be ended by a discussion.

8.1. Theatre

In this part, we represent a Theater fault tree, which contains 5 nodes and which are linked together as showed in Figure 47:

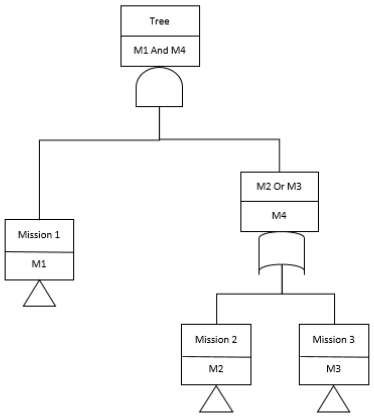


Figure 47: Fault tree

With :

- Tree = Theatre
- Mission 1 = Mains fail
- Mission 2 = Generator fail
- Mission 3 =Relay fail
- Mission 4 = Generator

8.2. SmallTree

In this part, we represent a SmallTree, which contains 7 nodes and which are linked together as showed in Figure 48:

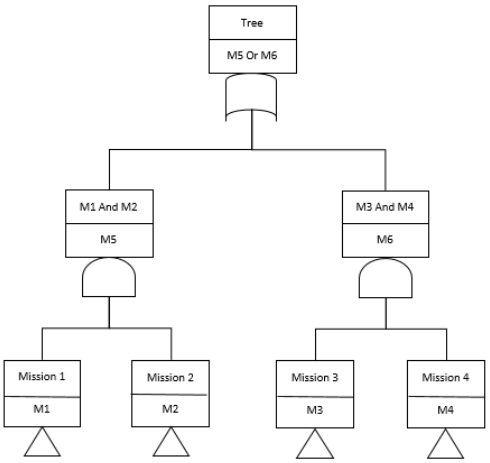


Figure 48: Small Fault tree

8.3. BSCU

In this part, we represent BSCU fault tree, which contains 15 nodes and which are linked together as reference to Figure 49:

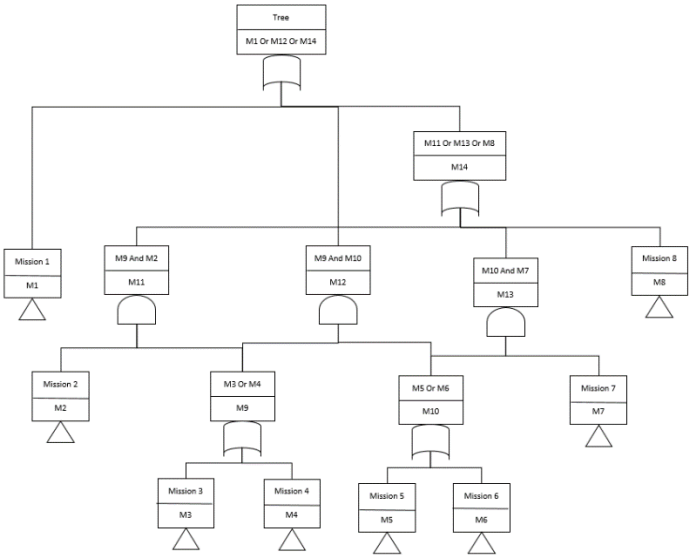


Figure 49: Fault tree BSCU

With :

- Mission 1 = Validity Moniter Failure
- Mission 2 = Switch Stuck In Position 2
- Mission 3 = System 2 Electronic Faillure
- Mission 4 = Loss Of System 2 Power Supply
- Mission 5 = System 1 Electronic Faillure
- Mission 6 = Loss Of System 1 Power Supply
- Mission 7 = Switch Stuck In position 1

- Mission 8 = Switch Stuck In Inter./mediate Position
- Mission 9 = Loss Of System 2
- Mission 10 = Loss Of system 1
- Mission 11 = Fails In Position 2 And System 2 fails
- Mission 12 = System 1 and 2 Do not operate
- Mission 13 = Switch fails in position 1 and System 1 fails
- Mission 14 = Switch Faillure
- Tree = Loss of brking commands

8.4. LIFT

In this part, we represent LIFT fault tree, which contains 27 nodes and which are linked together as reference to Figure 50:

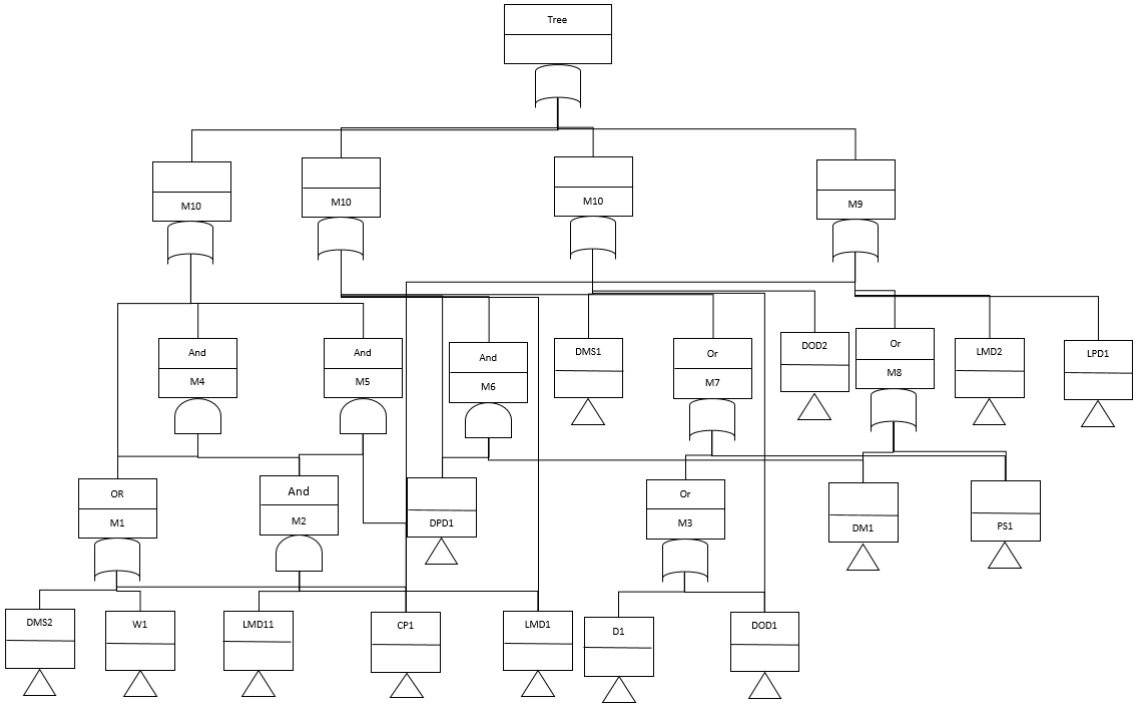


Figure 50: Fault tree LIFT

8.5. Other faults tree

In the table 6, we illustrate 7 faults tree of various sizes:

Fault Tree	Number of nodes	Number of clauses
FaultTree-0	27	30
FaultTree-1	30	31
FaultTree-2	31	30
FaultTree-3	31	31
FaultTree-4	32	32
FaultTree-5	32	33
FaultTree-6	33	34
FaultTree-7	33	34

Table 6: 7 Faults tree

8.6. Results

In this section, we compare the results obtained by the approach proposed in NEASQC Deliverable D6.4 [Rennela et al., 2021] and the "cooperative" approach proposed in section 6.7 according to the number of qubits used and if the problem is solved or not.

The CNF form of the fault trees $FaultTree - i$ can be found in appendix A.2.

Fault Tree	Number of nodes	Number of used qubits	D6.6 approach	Cooperative approach
Theatre	5	7	Solved	Solved
SmallTree	7	10	Solved	Solved
BSCU	15	20	Not Solved	Solved
LIFT	27	20	Not Solved	Solved
FaultTree-0	27	20	Not Solved	Solved
FaultTree-1	30	20	Not Solved	Solved
FaultTree-2	31	20	Not Solved	Solved
FaultTree-3	31	20	Not Solved	Solved
FaultTree-4	32	20	Not Solved	Solved
FaultTree-5	32	20	Not Solved	Solved
FaultTree-6	33	20	Not Solved	Solved
FaultTree-7	33	20	Not Solved	Solved

Table 7: Quantum Results

8.7. Discussion

In Table 7, we can see that in the quantum case, we cannot solve large fault trees, and this goes back to the number of qubits available today in quantum computers.

Almost like all use cases of quantum computing, the problem of the number of qubits are still a major problem to find a solution of our benchmark problem, we have encountered this problem even in the work that we have done in the paper [Zaiou et al., 2022a]. In the latter, we have proposed a quantum approach to search for minimal cuts in a directed graph, where we use movement oracles in it and we find these minimal cuts with a linear number of qubits and oracles. Even though we do a classical filter at the end to eliminate cuts that maybe are not minimal but are in general cuts.

Until today, we are not able to solve real cases or real fault trees in the field of PSA with quantum approaches. To make this feasible, or in order to find a solution of our problem, several approaches are envisioned: of course the first one is to have a quantum computer with an interesting number of qubits, which is the case that is not yet envisaged for the next few years; the second one is to divide the large fault trees on several small trees and to process them in a separate way and to merge the results in order to find the global results at the end.

For the first solution, it is in the hands of the researchers working on the hardwere, but for the second question, it depends on the skills that we can have, so the question here is how can we cut these large fault trees? and what method can we use to do this task?

To answer these two questions, we proposed to use the unsupervised machine learning algorithms, and for avoiding high splitting complexity, we proposed the two quantum approaches [Zaiou et al., 2021] and [Zaiou et al., 2022b]: in the first [Zaiou et al., 2021], we improved the quantum version of the unsupervised learning algorithm Balanced K-means, this algorithm allows to split a database (which can be a graph or a fault tree or any database represented in a given space) on k clusters where the size of the clusters is the same N/k, we chose this approach simply to have divided the fault tree on several small trees (clusters) of the same size N/K, N being the size of the data set, to have well specified the number of components in each small tree according to the number of qubits available in the quantum computer that we use; for the second paper [Zaiou et al., 2022b]: we proposed the quantum version of the Convex Non-negative Matrix Factorization algorithm, to find a first clustering apart from the matrix that represents the tree and to find a representative matrix of the data with a reduced dimension to make easier the steps for the algorithm of the paper [Zaiou et al., 2021].

With these two approaches, we can split the large fault trees into several small trees able to be processed according to the number of qubits available today in quantum computers, but the question that remains open until today, is how can we make a reverse work of this splitting to scale up the results?

8.8. Conclusion

In this document,we presented an overview of the main winners of the annual SAT competition; we compared them in terms of runtime and to see which solvers are able to solve a large number of instances. The benchmarks proposed in the different SAT competitions include several big instances that are comparable to the big industrial instances we may find in the PSA problem.

Our aim was to see to which extent these algorithms can be extended to deal with the PSA problem first classically and then using quantum algorithms. Our attempts to solve industrial instances using variants of some algorithms (for instance Minicard[1] [Liang et al., 2016], HTT-BDD [2][Toda and Soh, 2016]) that deal with the search of all solutions were not successful (which is not reported in this document).

We also have made benchmarks on some small fault trees in the PSA field to test quantum algorithms. By comparing these results, we can see that in the quantum case, we have very limited class of problems to solve with today algorithms (The algorithm proposed in NEASQC Deliverable 6.4 [Rennela et al., 2021] and the cooperation algorithm proposed in [Cheng and Tao, 2007]).

The problem of finding all the prime implicants is close to the SAT problem, in the sense that we are searching not only a satisfiable assignment, but all the assignments. In practice, the event combinations of low probability/frequency (lower than some threshold) are neglected and that is another pruning possibility of parts of the search tree.

With current hardware, scaling remains an important question and therefore requires a focus on efficient hybrid approaches that combine performant classical solvers with a relevant use of quantum superposition when it matters.

[1] https://github.com/liffiton/minicard
[2] http://www.sd.is.uec.ac.jp/toda/htcbdd.html

List of Acronyms

3-SAT SAT with formulae in CNF with each clause containing up to 3 literals . . . 23
AQC Adiabatic Quantum Computation . . . 23
CaDiCaL Radical CDCL Solver . . . 9
BCP Boolean Constraint Propagation . . . 9
CaDiCaL Hack GB GB on the top of CaDiCaL1.4.0 when VSIDS is active in the baseline system . . . 11
CDCL Conflict-Driven Clause Learning . . . 3
CHB Conflict-History Based . . . 11
CMS EXP VGBL GB method on top of LRB, and replace VSIDS with expVSIDS. . . . 11
CNF Conjunctive Normal Form . . . 6
CQS Cooperative Quantum Search . . . 27
CRVR Common Reason decision Variable score Reduction . . . 11
CryptoMiniSat CryptoMiniSat solver . . . 11
CSP Constraint Satisfaction Problem . . . 21
DPLL Davis, Putnam, Logemann and Loveland's algorithm . . . 8
EA Evolutionary Algorithms . . . 27
ERWA exponential recency weighted average . . . 10
MAB Multi-Armed Bandit . . . 10
expVSIDS VSDIS with expScore . . . 11
FTA Fault Tree Analysis . . . 23
TFO Traffic Flow Optimization . . . 25
GB Glue Bumping . . . 11
GenSAT Genetic Huill-Climbing Algorithms for Satisfiability . . . 27
GSAT GenSAT algorithm . . . 27
HHL Harrow-Hassidim-Lloyd . . . 5
k-SAT SAT with formulae in CNF with each clause containing up to 3 literals . . . 25
KISSAT The Kissat SAT solver is a condensed and improved reimplementation of CaDiCaL in C. It has improved data structures, better scheduling of inprocessing, optimized algorithms and implementation. . . . 10
KISSAT GB Kissat with Glue Bumping strategy . . . 11
KISSAT CRVR GB Kissat with CRVR and GB . . . 11
LRB Learning Rate Branching heuristic . . . 10
LR Learning Rate . . . 10
LSP Linear System Problems . . . 22
lstech MAPLE Local Search Tech Maple is an improved version of Relaxed LCM DCBDL newTech . 10
LSTech Local Search Tech . . . 10
MAB Multi-Armed Bandit . . . 10
Maple Maple . . . 10
PSA Probabilistic Safety Assessment . . . 5
POS Product Of Sums . . . 26
PPSZ Paturi, Pudlák, Saks, and Zane Algorithm . . . 28
QA Quantum annealing . . . 25
QUBO Quadratic Unconstrained Binary Optimization . . . 25
Relaxed Relaxed Conflict-Driven Clause Learning . . . 10
RNLT **R**elaxed **L**CMDCBDL **N**ew**T**ech . . . 10
SAT Boolean Satisfiability Problem . . . 5
STL Standard Template Library . . . 9
TSP Traveling Salesman Problem . . . 25
UCB Upper Confidence Bound . . . 11
VSIDS Variable State Independent Decaying Sum . . . 9
VMTF Variable Move to Front Strategy . . . 11
WHT Walsh-Hadamard Transform . . . 24

List of Figures

List of Tables

Bibliography

[Bal, 2020] (2020). SAT COMPETITION 2020 Solver and Benchmark Descriptions. Department of Computer Science, University of Helsinki.

[a. M. Luo, 2017] a. M. Luo, M. (2017). An effective learnt clause minimization approach for CDCL SAT solvers. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 7003–7011.

[Ambainis, 2010] Ambainis, A. (2010). New Developments in Quantum Algorithms. In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, {MFCS} 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, pages 1–11.

[Ambainis et al., 2007] Ambainis, A., Childs, A. M., Reichardt, B. W., Spalek, R., and Zhang, S. (2007). Any AND-OR Formula of Size N can be Evaluated in time $N^{1/2 + o(1)}$ on a Quantum Computer. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 363–372.

[Arboost Technologies, 2004] Arboost Technologies (2004). *Aralia User Manual*. Arboost.

[Backens and Kissinger, 2018] Backens, M. and Kissinger, A. (2018). ZH: A complete graphical calculus for quantum computations involving classical non-linearity. *arXiv preprint arXiv:1805.02175*.

[Backens and Kissinger, 2019] Backens, M. and Kissinger, A. (2019). ZH: A complete graphical calculus for quantum computations involving classical non-linearity. In *Electronic Proceedings in Theoretical Computer Science, EPTCS*.

[Backens et al., 2017] Backens, M., Perdrix, S., and Wang, Q. (2017). A simplified stabilizer ZX-calculus. In *Electronic Proceedings in Theoretical Computer Science, EPTCS*.

[Bäckström and Ying, 2008] Bäckström, O. and Ying, D. (2008). A presentation of the MCS BDD algorithm in the risk spectrum software package. In *9th International Conference on Probabilistic Safety Assessment and Management 2008, PSAM 2008*.

[Balas and Souza, 2005] Balas, E. and Souza, C. D. (2005). *The vertex separator problem: algorithms and computations*. Mathematical Programming.

[Balyo et al., 2017] Balyo, T., Heule, M. J. H., and Järvisalo, M. (2017). SAT COMPETITION 2017 Solver and Benchmark Descriptions. *Proceedings of SAT COMPETITION 2017*.

[Beanie et al., 2003] Beanie, P., Kautz, H., and Sabharwal, A. (2003). Understanding the power of clause learning. In *IJCAI International Joint Conference on Artificial Intelligence*.

[Bennett, 1989] Bennett, C. H. (1989). Time/space trade-offs for reversible computation. *SIAM Journal on Computing*.

[Biere., 2019] Biere., A. (2019). Cadical at the sat race. In *SAT RACE 2019*, page 8.

[Biere Armin and Heisinger, 2021] Biere Armin, F. M. and Heisinger, M. (2021). CADICAL, KISSAT , PARACOOBA Entering the SAT Competition 2021. In Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., and Suda, M. e., editors, *Proceedings of SAT Competition 2021 : Solver and Benchmark Descriptions*, pages 10–12.

[Bourreau et al., 2022] Bourreau, E., Fleury, G., and Lacomme, P. (2022). Adiabatic based Algorithm for SAT: a comprehensive algorithmic description.

[Brassard et al., 2000] Brassard, G., Höyer, P., Mosca, M., and Tapp, A. (2000). Quantum Amplitude Amplification and Estimation.

[Buhrman et al., 1998] Buhrman, H., Cleve, R., and Wigderson, A. (1998). Quantum vs. classical communication and computation. In *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*.

[Campbell et al., 2019] Campbell, E., Khurana, A., and Montanaro, A. (2019). Applying quantum algorithms to constraint satisfaction problems. *Quantum*.

[Campos et al., 2021] Campos, E., Venegas-Andraca, S. E., and Lanzagorta, M. (2021). Quantum tunneling and quantum walks as algorithmic resources to solve hard K-SAT instances. *Scientific Reports*.

[Chang et al., 2018] Chang, W., Xu, Y., and Chen, S. (2018). A new rewarding mechanism for branching heuristic in SAT solvers. *International Journal of Computational Intelligence Systems*.

[Chen and Gao, 2022] Chen, Y. A. and Gao, X. S. (2022). Quantum Algorithm for Boolean Equation Solving and Quantum Algebraic Attack on Cryptosystems. *Journal of Systems Science and Complexity*.

[Cheng and Tao, 2007] Cheng, S. T. and Tao, M. H. (2007). Quantum cooperative search algorithm for 3-SAT. *Journal of Computer and System Sciences*.

[Childs et al., 2017] Childs, A., Kothari, R., and Somma, R. (2017). Quantum Algorithm for Systems of Linear Equations with Exponentially Improved Dependence on Precision. *SIAM Journal on Computing*, 46(6):1920–1950.

[Childs et al., 2009] Childs, A. M., Cleve, R., Jordan, S. P., and Yonge-Mallo, D. L. (2009). Discrete-Query Quantum Algorithm for {NAND} Trees. *Theory of Computing*, 5(1):119–123.

[Choi, 2011] Choi, V. (2011). Different adiabatic quantum optimization algorithms for the NP-complete exact cover and 3SAT problems. *Quantum Information and Computation*.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Annual ACM Symposium on Theory of Computing*.

[Duncan et al., 2020] Duncan, R., Kissinger, A., Perdrix, S., and van de Wetering, J. (2020). Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum*.

[Duncan and Perdrix, 2014] Duncan, R. and Perdrix, S. (2014). Pivoting makes the ZX-calculus complete for real stabilizers. In *Electronic Proceedings in Theoretical Computer Science, EPTCS*.

[Dunjko et al., 2018a] Dunjko, V., Ge, Y., and Cirac, J. I. (2018a). Computational Speedups Using Small Quantum Devices. *Physical Review Letters*.

[Dunjko et al., 2018b] Dunjko, Y., Ge, V., and J., I. C. (2018b). Computational Speedups Using Small Quantum Devices. *PHYSICAL REVIEW LETTERS*, 121(250501).

[EPRI, 2008] EPRI (2008). FTREX User Manual Version 1.4.

[et al. ANIS, 2021] et al. ANIS, M. D. S. (2021). Qiskit: An Open-source Framework for Quantum Computing.

[Ezratty, 2021] Ezratty, O. (2021). Understanding Quantum Technologies.

[Farhi et al., 2001] Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A., and Preda, D. (2001). A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science*.

[Farhi and Gutmann, 1997] Farhi, E. and Gutmann, S. (1997). Quantum Computation and Decision Trees. *Physical Review A*, 58.

[Froleyks et al., 2021] Froleyks, N., Heule, M., Iser, M., Järvisalo, M., and Suda, M. (2021). SAT Competition 2020. *Artificial Intelligence*, 301:103572.

[Ge and Dunjko, 2020] Ge, Y. and Dunjko, V. (2020). A hybrid algorithm framework for small quantum computers with application to finding Hamiltonian cyclesball. *Journal of Mathematical Physics*.

[Goldberg and Novikov, 2002] Goldberg, E. and Novikov, Y. (2002). Berkmin: A fast and robust sat-solver. *Proceedings -Design, Automation and Test in Europe, DATE*.

[Gomes et al., 1998] Gomes, C. P., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the National Conference on Artificial Intelligence*.

[Grant and Humble, 2020] Grant, E. K. and Humble, T. S. (2020). Adiabatic Quantum Computing and Quantum Annealing.

[Guan et al., 2021] Guan, J., Wang, Q., and Ying, M. (2021). An HHL-based algorithm for computing hitting probabilities of quantum walks. *Quantum Information and Computation*.

[Hadfield, 2021] Hadfield, S. (2021). On the Representation of Boolean and Real Functions as Hamiltonians for Quantum Computing. *ACM Transactions on Quantum Computing*.

[Hastings, 2020] Hastings, M. B. (2020). Classical and quantum algorithms for tensor principal component analysis. *Quantum*.

[Hereramarti, 2020] Hereramarti, D. (2020). Quantum Counting Pour Études Probabilistes de Sureté. Technical report, ATOS.

[Hertli, 2011] Hertli, T. (2011). 3-SAT faster and simpler - Unique-SAT bounds for PPSZ hold in general. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*.

[Heule et al., 2019] Heule, M. J. H., Järvisalo, M., and Suda, M. (2019). SAT Competition 2018. *Journal on Satisfiability, Boolean Modeling and Computation*.

[Hibti, 2013] Hibti, M. (2013). What if we revisit evaluation of PSA models with network algorithms ? Ans psa 2013 international topical meeting on probabilistic safety assessment and analysis.

[Hogg, 2003] Hogg, T. (2003). Adiabatic quantum computing for random satisfiability problems. *Physical Review A - Atomic, Molecular, and Optical Physics*.

[Høyer et al., 2003] Høyer, P., Mosca, M., and De Wolf, R. (2003). Quantum search on bounded-error inputs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

[J. B. Fussell and Marshall, 1974] J. B. Fussell, E. B. H. and Marshall, N. H. (1974). {MOCUS} – A Computer program To Obtain Minimal Sets From Fault Trees. Technical report, Aerojet Nuclear Corp.

[Järvisalo et al., 2012] Järvisalo, M., Heule, M. J., and Biere, A. (2012). Inprocessing rules. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

[Jeffery and Kimmel, 2017] Jeffery, S. and Kimmel, S. (2017). Quantum Algorithms for Graph Connectivity and Formula Evaluation. *CoRR*, abs/1704.0.

[Jung, 2009] Jung, W. S. (2009). ZBDD algorithm features for an efficient Probabilistic Safety Assessment. *Nuclear Engineering and Design*.

[Jung, 2015] Jung, W. S. (2015). A method to improve cutset probability calculation in probabilistic safety assessment of nuclear power plants. *Reliability Engineering and System Safety*.

[Kissinger and van de Wetering, 2020] Kissinger, A. and van de Wetering, J. (2020). PyZX: Large scale automated diagrammatic reasoning. In *Electronic Proceedings in Theoretical Computer Science, EPTCS*.

[Knuth D. E.,] Knuth D. E. *The Art of Computer Programming, vol 4, Boolean Basics*. Addison-Wesley.

[Kochemazov, 2021] Kochemazov, S. (2021). Analysis of comparative effectiveness of state-of-the-art heuristics for CDCL SAT solvers. In *CEUR Workshop Proceedings*.

[Kochemazov et al., 2020] Kochemazov, S., Zaikin, O., Semenov, A., and Kondratiev, V. (2020). Speeding up cdcl inference with duplicate learnt clauses. In *Frontiers in Artificial Intelligence and Applications*.

[Krüger and Mauerer, 2020] Krüger, T. and Mauerer, W. (2020). Quantum Annealing-Based Software Components: An Experimental Case Study with SAT Solving. In *Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020*.

[Lewis et al., 2005] Lewis, M. D. T., Schubert, T., and Becker, B. W. (2005). Speedup Techniques Utilized in Modern SAT Solvers.

[Liang et al., 2016] Liang, J. H., Ganesh, V., Poupart, P., and Czarnecki, K. (2016). Learning rate based branching heuristic for SAT solvers. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

[Liu et al., 2008] Liu, W. Z., Zhang, J. F., and Long, G. L. (2008). A parallel quantum algorithm for the satisfiability problem. *Communications in Theoretical Physics*.

[Mao, 2005] Mao, W. (2005). Solving satisfiability problems by the ground-state quantum computer. *Physical Review A - Atomic, Molecular, and Optical Physics*.

[Marques-Silva et al., 2021] Marques-Silva, J., Lynce, I., and Malik, S. (2021). Chapter 4: Conflict-driven clause learning SAT solvers. In *Frontiers in Artificial Intelligence and Applications*.

[Marques Silva and Sakallah, 1996] Marques Silva, J. P. and Sakallah, K. A. (1996). GRASP - a new search algorithm for satisfiability. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers*.

[Martiel and Remaud, 2020] Martiel, S. and Remaud, M. (2020). Practical Implementation of a Quantum Backtracking Algorithm. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

[Meuli et al., 2019a] Meuli, G., Soeken, M., Roetteler, M., Bjorner, N., and De Micheli, G. (2019a). Reversible pebbling game for quantum memory management.

[Meuli et al., 2019b] Meuli, G., Soeken, M., Roetteler, M., Bjorner, N., Micheli, G. D., and De Micheli, G. (2019b). Reversible Pebbling Game for Quantum Memory Management. In *Proceedings of the 2019 Design, Automation and Test in Europe Conference and Exhibition, DATE 2019*.

[Mizel et al., 2001] Mizel, A., Mitchell, M. W., and Cohen, M. L. (2001). Energy barrier to decoherence. *Physical Review A*, 63(4).

[Montanaro, 2019] Montanaro, A. (2019). Quantum-walk speedup of backtracking algorithms. *Theory of Computing*.

[Moskewicz et al., 2001] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Proceedings - Design Automation Conference*.

[Munson et al., 2019] Munson, A., Coecke, B., and Wang, Q. (2019). A note on AND-gates in ZX-calculus: QBC-completeness and phase gadgets.

[Nadel and Ryvchin, 2018] Nadel, A. and Ryvchin, V. (2018). Chronological Backtracking. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

[Nielsen and Chuang, 2011] Nielsen, M. A. and Chuang, I. L. (2011). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition.

[Ohya and Volovich, 2003] Ohya, M. and Volovich, I. V. (2003). New quantum algorithm for studying NP-complete problems. *Reports on Mathematical Physics*.

[R., 1986] R., B. (1986). *Graph Based Algorithms for Boolean Function Manipulation*. IEEE transactions on computers.

[Rauzy A., 2012] Rauzy A. (2012). An Open-PSA Fault Tree engine. *PSAM/ESREL 2011*.

[Relcon AB., 2003] Relcon AB. (2003). *RiskSpectrum Professional, RSAT users Manual*. Relcon AB.

[Rennela et al., 2021] Rennela, M., Brand, S., Laarman, A., and Dunjko, V. (2021). Divide & Quantum Mathematical Foundation. Technical report, NEASQC Project.

[Ryan, 2004] Ryan, L. (2004). Efficient algorithms for clause-learning SAT solvers. *Citeseer*.

[SAIC and EPRI, 1989] SAIC and EPRI (1989). *CAFTA user's manual, version 2.0*. EPRI (Series). The Institute.

[Siciliano, 2018] Siciliano, F. (2018). On the applications of quantum computing to probablistic safety assessment. Technical report, EDF.

[Smelyanskiy et al., 2012] Smelyanskiy, V. N., Rieffel, E. G., Knysh, S. I., Williams, C. P., Johnson, M. W., Thom, M. C., Macready, W. G., and Pudenz, K. L. (2012). A Near-Term Quantum Computing Approach for Hard Computational Problems in Space Exploration.

[Toda and Soh, 2016] Toda, T. and Soh, T. (2016). Implementing efficient all solutions SAT solvers. *ACM Journal of Experimental Algorithmics*.

[Ton and Dieter, 1998] Ton, K. and Dieter, K. (1998). Listing All Minimal Separators of a Graph, SIAM J. *Comput.*, 27:605–613.

[Udovičić, 2006] Udovičić, M. (2006). Chronological and dependency-directed backtracking. In *SISY 2006 - 4th Serbian-Hungarian Joint Symposium on Intelligent Systems*.

[Van Dam et al., 2001] Van Dam, W., Mosca, M., and Vazirani, U. (2001). How powerful is adiabatic quantum computation? *Annual Symposium on Foundations of Computer Science - Proceedings*.

[van de Wetering, 2020] van de Wetering, J. (2020). ZX-calculus for the working quantum computer scientist.

[Wang et al., 2019] Wang, D., Higgott, O., and Brierley, S. (2019). Accelerated variational quantum eigensolver. *Physical Review Letters*.

[Zaiou et al., 2022a] Zaiou, A., Bennani, Y., Hibti, M., and Matei, B. (2022a). Quantum Approach for Vertex Separator Problem in Directed Graphs. *Encours*.

[Zaiou et al., 2021] Zaiou, A., Bennani, Y., Matei, B., and Hibti, M. (2021). Balanced K-means using Quantum annealing. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE.

[Zaiou et al., 2022b] Zaiou, A., Matei, B., Bennani, Y., and Hibti, M. (2022b). Convex Non-negative Matrix Factorization Through Quantum Annealing. *arXiv preprint arXiv:2203.15634*.

[Zhang et al., 2020] Zhang, R., Chen, J., and Zhao, H.-l. (2020). Procedure of Solving 3-SAT Problem by Combining Quantum Search Algorithm and DPLL Algorithm.

[Zhang and Cai, 2020] Zhang, X. and Cai, S. (2020). 2020 Relaxed backtracking with rephasing. In *Proc. of SAT Competition 2020*, pages VOL–B 15–16.

[Zhang et al., 2021] Zhang, X., Cai, S., and Chen, Z. (2021). Improving CDCL via Local Search. In Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., and Suda, M. e., editors, *Proceedings of SAT Competition 2021 : Solver and Benchmark Descriptions*, page 42.

[Zhao et al., 2019] Zhao, J., Wu, Y. C., Guo, G. C., and Guo, G. P. (2019). State preparation based on quantum phase estimation.

A.Appendix

A.1. Implication Graph

A.1.1. Trail

During the resolution of a Boolean formula using CDCL we keep trails to remember why a variable has certain value in the current partial truth assignment. A trail is a sequence of literals annotated either with a special symbol (e.g. `dec` for decision) or with a clause in the current formula (including the original clauses as well as the clauses learned so far).

The trail is maintained as follows:

- When the algorithm decides that the literal l should be true, the trail is appended with l^{dec}
- When the algorithm uses unit propagation on a clause $C = (l_1 \vee \cdots \vee l_k \vee l)$ to imply that l must be true, then the trail is appended with l^C
- When the algorithm backjumps, a number of annotated literals is removed from the back of the trail.

Trails can be seen as directed acyclic graphs called "implication graphs".

A.1.2. Implication Graph

Consider a trail T. A corresponding implication graph G_T is the directed acyclic graph (V, E), where the vertex set is $V = \{l | lr \in T\}$ and the edge set E has an edge $(\neg l_i, l)$ for each $1 \leq i \leq k$ with $l^C \in T$, $C = (l_1 \vee \cdots \vee l_k \vee l)$.

A.2. CNF format of the fult trees used in the benchmark

$$FaultTree-0 : (\neg v_{19} \vee v_3 \vee \neg v_1) \wedge (v_{17} \vee v_{22} \vee v_7) \wedge (v_{30} \vee \neg v_{18} \vee \neg v_{28}) \wedge (v_{24} \vee v_{29} \vee \neg v_{23}) \wedge (\neg v_{17} \vee v_{19} \vee \neg v_{18}) \wedge (\neg v_{10} \vee \neg v_3 \vee \neg v_{22}) \wedge (v_{20} \vee v_{25} \vee \neg v_{18}) \wedge (\neg v_{16} \vee \neg v_{13} \vee \neg v_{19}) \wedge (\neg v_{17} \vee v_7 \vee \neg v_{23}) \wedge (v_{23} \vee \neg v_1 \vee \neg v_4) \wedge (\neg v_7 \vee \neg v_{30} \vee \neg v_{10}) \wedge (v_{20} \vee \neg v_{18} \vee v_1) \wedge (\neg v_{16} \vee v_{12} \vee \neg v_{26}) \wedge (\neg v_{16} \vee \neg v_{14} \vee \neg v_{11}) \wedge (\neg v_{28} \vee v_6 \vee v_{12}) \wedge (v_{14} \vee \neg v_5 \vee v_{23}) \wedge (v_{14} \vee v_{12} \vee \neg v_{25}) \wedge (v_{21} \vee \neg v_5 \vee v_{12}) \wedge (v_{26} \vee \neg v_{20} \vee \neg v_{29}) \wedge (v_6 \vee \neg v_{12} \vee \neg v_{26}) \wedge (\neg v_{30} \vee v_{27} \vee \neg v_1) \wedge (v_{17} \vee \neg v_{25} \vee \neg v_{10}) \wedge (\neg v_6 \vee v_1 \vee v_{22}) \wedge (\neg v_{12} \vee \neg v_{19}) \wedge (v_{23} \vee v_8 \vee \neg v_{25}) \wedge (\neg v_{21} \vee \neg v_7 \vee \neg v_{11}) \wedge (v_{25} \vee v_{10} \vee v_{21}) \wedge (\neg v_{21} \vee v_6 \vee \neg v_4) \wedge (v_{30} \vee v_{18}) \wedge (v_5 \vee \neg v_8 \vee v_{23})$$

$$FaultTree-1 : (\neg v_{27} \vee \neg v_3 \vee v_6) \wedge (v_{31} \vee v_{25} \vee v_{30}) \wedge (v_3 \vee v_7 \vee \neg v_{24}) \wedge (v_{19} \vee \neg v_7 \vee \neg v_{13}) \wedge (v_{12} \vee v_3 \vee \neg v_7) \wedge (\neg v_3 \vee \neg v_{18} \vee v_{16}) \wedge (\neg v_{15} \vee \neg v_{11} \vee v_5) \wedge (\neg v_{22} \vee \neg v_1 \vee v_6) \wedge (\neg v_8 \vee v_{21} \vee v_9) \wedge (\neg v_{14} \vee \neg v_{25} \vee \neg v_9) \wedge (\neg v_{24} \vee v_7 \vee v_{23}) \wedge (v_{12} \vee \neg v_{20} \vee \neg v_{19}) \wedge (v_7 \vee \neg v_{19} \vee \neg v_{10}) \wedge (v_{23} \vee v_{22} \vee v_{29}) \wedge (\neg v_{28} \vee \neg v_{29} \vee \neg v_{24}) \wedge (v_7 \vee \neg v_{21}) \wedge (v_{13} \vee \neg v_{14} \vee \neg v_{16}) \wedge (v_{14} \vee v_{11}) \wedge (\neg v_8 \vee \neg v_{27} \vee \neg v_{12}) \wedge (\neg v_{21} \vee \neg v_{30} \vee \neg v_{13}) \wedge (\neg v_{10} \vee v_{19} \vee v_{26}) \wedge (v_{16} \vee \neg v_{14} \vee \neg v_{23}) \wedge (v_{11} \vee v_{10} \vee \neg v_{16}) \wedge (v_{27} \vee v_6 \vee v_{28}) \wedge (v_{25} \vee \neg v_{24} \vee \neg v_{21}) \wedge (v_{25} \vee \neg v_7) \wedge (v_9 \vee \neg v_{22} \vee v_5) \wedge (\neg v_4 \vee v_{23} \vee v_{18}) \wedge (v_{26} \vee \neg v_{17} \vee \neg v_{11}) \wedge (v_6 \vee \neg v_{26} \vee v_{14})$$

$$FaultTree-2 : (\neg v_{17} \vee \neg v_6 \vee v_{22}) \wedge (\neg v_3 \vee v_{11} \vee v_{10}) \wedge (v_{32} \vee v_{11} \vee \neg v_9) \wedge (\neg v_{23} \vee \neg v_{13} \vee v_{15}) \wedge (v_1 \vee \neg v_{16} \vee \neg v_{22}) \wedge (v_{23} \vee v_{22} \vee \neg v_{16}) \wedge (v_{16} \vee \neg v_{21} \vee \neg v_{10}) \wedge (\neg v_{17} \vee v_8 \vee v_{12}) \wedge (v_{11} \vee v_{10} \vee v_9) \wedge (\neg v_9 \vee v_{10} \vee \neg v_{11}) \wedge (\neg v_7 \vee v_{21} \vee v_{26}) \wedge (\neg v_{13} \vee v_{14} \vee v_{18}) \wedge (v_{17} \vee v_8 \vee v_{25}) \wedge (v_{20} \vee \neg v_5 \vee \neg v_{23}) \wedge (\neg v_{18} \vee v_{27} \vee \neg v_2) \wedge (v_{32} \vee \neg v_6 \vee v_{10}) \wedge (\neg v_{23} \vee v_{19} \vee \neg v_{17}) \wedge (v_{22} \vee v_{25} \vee \neg v_4) \wedge (v_{18} \vee \neg v_3 \vee v_{16}) \wedge (\neg v_1 \vee \neg v_6 \vee \neg v_{32}) \wedge (\neg v_1 \vee \neg v_{22} \vee \neg v_6) \wedge (\neg v_3 \vee \neg v_{30} \vee v_{13}) \wedge (\neg v_{13} \vee v_{26} \vee \neg v_{25}) \wedge (v_4 \vee \neg v_{31} \vee \neg v_{30}) \wedge (v_8 \vee v_1 \vee v_{26}) \wedge (v_8 \vee v_2 \vee v_1) \wedge (\neg v_{15} \vee \neg v_{19} \vee \neg v_7) \wedge (v_{17} \vee \neg v_{18} \vee \neg v_2) \wedge (v_1 \vee v_6 \vee v_{32}) \wedge (\neg v_{13} \vee \neg v_{15} \vee \neg v_{26}) \wedge (v_{12} \vee \neg v_{24} \vee \neg v_3)$$

$$FaultTree-3 : (v_1 \vee v_{13} \vee \neg v_{33}) \wedge (\neg v_{32} \vee v_2 \vee v_9) \wedge (v_{22} \vee v_{10} \vee \neg v_8) \wedge (v_6 \vee v_7 \vee \neg v_{22}) \wedge (v_{27} \vee v_{13} \vee \neg v_{21}) \wedge (\neg v_{25} \vee \neg v_{14} \vee v_5) \wedge (\neg v_{27} \vee \neg v_{18} \vee v_{22}) \wedge (v_4 \vee v_8 \vee \neg v_{18}) \wedge (v_{12} \vee \neg v_8 \vee v_{15}) \wedge (\neg v_{20} \vee \neg v_{10} \vee \neg v_{17}) \wedge (v_{12} \vee v_3 \vee v_1) \wedge (v_1 \vee \neg v_{33} \vee v_6) \wedge (\neg v_{13} \vee \neg v_{17} \vee \neg v_4) \wedge (\neg v_{19} \vee v_7 \vee v_9) \wedge (v_5 \vee v_8 \vee v_9) \wedge (v_{16} \vee v_6 \vee \neg v_{18}) \wedge (v_{14} \vee v_{28}) \wedge (v_{11} \vee v_{14} \vee v_{22}) \wedge (\neg v_7 \vee \neg v_{27} \vee v_{15}) \wedge (\neg v_{16} \vee \neg v_{20} \vee v_{30}) \wedge (\neg v_6 \vee v_{27} \vee v_4) \wedge (\neg v_{24} \vee v_{33} \vee \neg v_5) \wedge (\neg v_{13} \vee \neg v_1 \vee \neg v_{30}) \wedge (\neg v_{19} \vee v_6 \vee \neg v_7) \wedge (\neg v_{21} \vee v_{27}) \wedge (\neg v_{27} \vee v_6 \vee \neg v_{29}) \wedge (v_{19} \vee v_{10}) \wedge (v_{31} \vee \neg v_2 \vee \neg v_{17}) \wedge (v_{13} \vee \neg v_1 \vee v_{20}) \wedge (\neg v_{15} \vee v_{28} \vee v_{13}) \wedge (\neg v_{15} \vee \neg v_{12} \vee \neg v_3)$$

$$FaultTree-4 : (v_{19} \vee \neg v_6 \vee v_8) \wedge (v_{11} \vee \neg v_{23} \vee v_{15}) \wedge (v_4 \vee \neg v_{19} \vee v_{17}) \wedge (\neg v_{29} \vee v_4 \vee \neg v_{16}) \wedge (\neg v_{12} \vee \neg v_{22} \vee v_{14}) \wedge (v_6 \vee \neg v_{21} \vee \neg v_{25}) \wedge (v_{23} \vee v_9 \vee \neg v_{24}) \wedge (\neg v_{14} \vee \neg v_7 \vee \neg v_{19}) \wedge (\neg v_{15} \vee \neg v_{30} \vee \neg v_3) \wedge (\neg v_{30} \vee \neg v_6 \vee \neg v_{31}) \wedge (\neg v_{11} \vee v_9 \vee v_{12}) \wedge (\neg v_{22} \vee v_8 \vee \neg v_{31}) \wedge (v_{18} \vee v_{12} \vee v_{21}) \wedge (\neg v_6 \vee \neg v_{17}) \wedge (v_{25} \vee \neg v_5 \vee \neg v_{17}) \wedge (\neg v_{17} \vee \neg v_{21}) \wedge (\neg v_{31} \vee v_{22} \vee v_{15}) \wedge (v_9 \vee v_{25} \vee v_{20}) \wedge (v_1 \vee \neg v_4 \vee v_{17}) \wedge (v_{15} \vee v_6 \vee \neg v_{24}) \wedge (\neg v_{27} \vee v_{15} \vee v_{14}) \wedge (\neg v_{12} \vee v_{14} \vee \neg v_6) \wedge (v_{34} \vee \neg v_{32} \vee v_9) \wedge (\neg v_{26} \vee v_2 \vee \neg v_{17}) \wedge (\neg v_9 \vee v_{34} \vee v_{10}) \wedge (\neg v_{18} \vee \neg v_{30}) \wedge (v_3 \vee v_{13} \vee \neg v_8) \wedge (v_{22} \vee \neg v_{12} \vee v_2) \wedge (v_1 \vee \neg v_{23} \vee v_{25}) \wedge (\neg v_2 \vee v_{19} \vee v_{29}) \wedge (\neg v_7 \vee \neg v_{34} \vee v_{22}) \wedge (v_{23} \vee v_6 \vee v_9)$$

$$FaultTree-5: (\neg v_{16} \vee v_8 \vee \neg v_{20}) \wedge (v_{35} \vee \neg v_6 \vee \neg v_{30}) \wedge (v_8 \vee v_5 \vee v_{22}) \wedge (\neg v_{27} \vee v_6 \vee \neg v_{17}) \wedge (v_{15} \vee v_{20} \vee \neg v_{25}) \wedge (\neg v_{11} \vee v_{29} \vee \neg v_{14}) \wedge (v_{18} \vee v_3 \vee \neg v_{25}) \wedge (\neg v_{30} \vee \neg v_{28} \vee v_9) \wedge (\neg v_{25} \vee \neg v_{32} \vee \neg v_{30}) \wedge (v_{14} \vee v_7 \vee v_{18}) \wedge (\neg v_9 \vee v_{14} \vee v_{19}) \wedge (\neg v_{17} \vee v_{25}) \wedge (v_{11} \vee \neg v_{28} \vee \neg v_{23}) \wedge (v_1 \vee \neg v_{16} \vee \neg v_{33}) \wedge (\neg v_{15} \vee v_{14} \vee \neg v_2) \wedge (v_9 \vee v_{17} \vee v_{21}) \wedge (\neg v_{34} \vee \neg v_3 \vee v_{16}) \wedge (v_{26} \vee \neg v_{22} \vee v_{12}) \wedge (\neg v_{15} \vee \neg v_{18} \vee v_{25}) \wedge (\neg v_{34} \vee \neg v_{11} \vee \neg v_{35}) \wedge (\neg v_4 \vee \neg v_{21} \vee v_{26}) \wedge (\neg v_{28} \vee v_{13} \vee \neg v_8) \wedge (\neg v_{21} \vee v_7) \wedge (v_{31} \vee v_{21} \vee v_{29}) \wedge (\neg v_8 \vee \neg v_{11} \vee \neg v_{12}) \wedge (v_{34} \vee \neg v_{26} \vee \neg v_{29}) \wedge (v_3 \vee \neg v_{12} \vee v_{25}) \wedge (\neg v_{18} \vee \neg v_6 \vee v_{13}) \wedge (\neg v_1 \vee \neg v_{22}) \wedge (\neg v_{33} \vee \neg v_{20} \vee \neg v_{23}) \wedge (\neg v_{14} \vee v_{20} \vee v_{15}) \wedge (\neg v_{31} \vee \neg v_{15} \vee \neg v_{19})$$

$$FaultTree-6: (v_{18} \vee \neg v_7 \vee \neg v_{11}) \wedge (\neg v_{15} \vee v_{26} \vee v_{13}) \wedge (\neg v_9 \vee v_{22} \vee \neg v_7) \wedge (v_7 \vee \neg v_{16} \vee v_{12}) \wedge (v_{21} \vee \neg v_6 \vee \neg v_{18}) \wedge (v_{36} \vee \neg v_{17} \vee \neg v_{24}) \wedge (v_{10} \vee \neg v_{29} \vee v_{18}) \wedge (v_{35} \vee \neg v_9) \wedge (v_{29} \vee \neg v_{15}) \wedge (\neg v_{33} \vee v_{31} \vee \neg v_{12}) \wedge (\neg v_{25} \vee v_8 \vee v_{12}) \wedge (v_{36} \vee \neg v_8 \vee \neg v_{31}) \wedge (\neg v_{25} \vee \neg v_1) \wedge (\neg v_{14} \vee \neg v_{17} \vee v_{24}) \wedge (v_8 \vee v_{36} \vee v_2) \wedge (\neg v_6 \vee \neg v_{13} \vee \neg v_{31}) \wedge (v_{23} \vee \neg v_{33} \vee v_{18}) \wedge (\neg v_{33} \vee \neg v_{27} \vee v_{34}) \wedge (\neg v_5 \vee v_{29} \vee \neg v_{16}) \wedge (v_3 \vee \neg v_{15} \vee \neg v_{24}) \wedge (\neg v_6 \vee \neg v_8 \vee v_{20}) \wedge (v_{19} \vee v_{16} \vee v_5) \wedge (v_{33} \vee v_{30} \vee \neg v_6) \wedge (v_2 \vee \neg v_{23} \vee \neg v_7) \wedge (v_2 \vee \neg v_{36} \vee v_{25}) \wedge (v_9 \vee v_{27} \vee v_{34}) \wedge (v_{21} \vee v_{22}) \wedge (\neg v_{22} \vee v_{20} \vee \neg v_{11}) \wedge (\neg v_{27} \vee v_{26} \vee v_6) \wedge (\neg v_{22} \vee \neg v_{10} \vee v_5) \wedge (\neg v_{34} \vee v_{28} \vee \neg v_{21}) \wedge (\neg v_{11} \vee \neg v_{10} \vee \neg v_{16}) \wedge (v_7 \vee \neg v_{19} \vee \neg v_{15})$$

$$FaultTree-7: (v_{20} \vee \neg v_5) \wedge (v_7 \vee v_{32} \vee \neg v_{20}) \wedge (v_2 \vee \neg v_9 \vee \neg v_{33}) \wedge (v_{25} \vee \neg v_{36}) \wedge (\neg v_{14} \vee v_{17} \vee \neg v_{28}) \wedge (v_9 \vee \neg v_{22} \vee v_{10}) \wedge (\neg v_{11} \vee v_3 \vee v_{36}) \wedge (v_{20} \vee \neg v_{35} \vee \neg v_{24}) \wedge (v_6 \vee v_{20} \vee \neg v_{23}) \wedge (\neg v_{30} \vee \neg v_{14} \vee v_4) \wedge (\neg v_6 \vee \neg v_{17} \vee v_{33}) \wedge (v_{11} \vee v_{19} \vee \neg v_{15}) \wedge (v_{36} \vee v_8 \vee \neg v_{16}) \wedge (v_{10} \vee v_{30} \vee \neg v_{12}) \wedge (\neg v_{24} \vee v_{30}) \wedge (v_{34} \vee \neg v_{14} \vee \neg v_{33}) \wedge (v_{15} \vee \neg v_{17} \vee \neg v_7) \wedge (\neg v_{17} \vee \neg v_{25} \vee v_{34}) \wedge (v_{37} \vee \neg v_{18} \vee \neg v_{33}) \wedge (v_3 \vee v_5 \vee \neg v_{12}) \wedge (\neg v_{14} \vee \neg v_{30} \vee v_7) \wedge (\neg v_{20} \vee \neg v_{35} \vee v_9) \wedge (\neg v_3 \vee v_{34} \vee v_2) \wedge (v_5 \vee v_{20} \vee \neg v_2) \wedge (v_{26} \vee \neg v_{13} \vee v_{14}) \wedge (\neg v_{16} \vee \neg v_{35} \vee v_{12}) \wedge (v_{20} \vee \neg v_4 \vee \neg v_{28}) \wedge (v_{26} \vee v_{17} \vee \neg v_{27}) \wedge (v_{34} \vee \neg v_{32} \vee \neg v_{31}) \wedge (v_{13} \vee \neg v_{11} \vee v_{33}) \wedge (\neg v_{18} \vee \neg v_{32} \vee \neg v_7) \wedge (\neg v_{22} \vee \neg v_{16} \vee v_{34}) \wedge (v_{34} \vee v_{15} \vee v_{32})$$