It's Quick to be Square: Fast Quadratisation for Quantum Toolchains

LUKAS SCHMIDBAUER, Computer Science, Technical University of Applied Sciences Regensburg, Regensburg, Germany
ELISABETH LOBE, Institute of Software Technology, Department High-Performance Computing, German Aerospace Center (DLR), Braunschweig, Germany
INA SCHAEFER, Institute of Information Security and Dependability (KASTEL)„ KIT, Karlsruhe, Germany
WOLFGANG MAUERER, Technical University of Applied Sciences Regensburg, Regensburg, Germany and Technology, Siemens AG, Munich, Germany

Many of the envisioned use-cases for quantum computers involve optimisation processes. While there are many algorithmic primitives to perform the required calculations, all eventually lead to quantum gates operating on quantum bits, with an order as determined by the structure of the objective function and the properties of target hardware. When the structure of the problem representation is not aligned with structure and boundary conditions of the executing hardware, various overheads degrading the computation may arise, possibly negating any possible quantum advantage.

Therefore, automatic transformations of problem representations play an important role in quantum computing when descriptions (semi-)targeted at humans must be cast into forms that can be "executed" on quantum computers. Mathematically equivalent formulations are known to result in substantially different non-functional properties depending on hardware, algorithm and detail properties of the problem. Given the current state of noisy intermediate-scale quantum (NISQ) hardware, these effects are considerably more pronounced than in classical computing. Likewise, efficiency of the transformation itself is relevant because possible quantum advantage may easily be eradicated by the overhead of transforming between representations. In this article, we consider a specific class of higher-level representations, that is, PUBOs, and devise novel automatic transformation mechanisms into widely used QUBOs that substantially improve efficiency and versatility over the state of the art. In addition, we conduct a comprehensive investigation of industry-relevant

This work was supported by the German Research Foundation, grant MA 9739/1-1 and SCHA 1635/20-1, as well as by the German Federal Ministry of Education and Research (BMBF), funding program "quantum technologies–from basic research to market", grant number 13N16092 and by the European Union (Project Reference 101083427) and the **European Funds for Regional Development** (**EFRE**) (Project Reference 20-3092.10-THD-105) and the project "Algorithms for quantum computer development in hardware-software codesign" (ALQU), qci.dlr.de/en/alqu, which was made possible by the DLR **Quantum Computing Initiative** (**QCI**) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK). WM acknowledges support by the High-Tech Agenda of the Free State of Bavaria.
Authors' Contact Information: Lukas Schmidbauer (corresponding author), Computer Science, Technical University of Applied Sciences Regensburg, Regensburg, Germany; e-mail: lukas.schmidbauer@othr.de; Elisabeth Lobe, Institute of Software Technology, Department High-Performance Computing, German Aerospace Center (DLR), Braunschweig, Germany; e-mail: elisabeth.lobe@dlr.de; Ina Schaefer, Institute of Information Security and Dependability (KASTEL), KIT, Karlsruhe, BW, Germany; e-mail: ina.schaefer@kit.edu; Wolfgang Mauerer, Technical University of Applied Sciences Regensburg, Regensburg, Germany, Technology and Siemens AG, Munich, BY, Germany; e-mail: Wolfgang.Mauerer@oth-regensburg.de.

ACM 2643-6817/2026/07-ART18
https://doi.org/10.1145/3800943

problem formulations and their conversion into a quantum-specific representation, identifying significant obstacles in scaling behaviour and demonstrating how these can be circumvented.

CCS Concepts: • **Theory of computation → Quantum computation theory**; **Graph algorithms analysis**; **Data structures design and analysis**; • **General and reference → Performance;**

Additional Key Words and Phrases: Pseudo boolean function, graphs, performance, algorithmic optimisation

ACM Reference Format:
Lukas Schmidbauer, Elisabeth Lobe, Ina Schaefer, and Wolfgang Mauerer. 2026. It's Quick to be Square: Fast Quadratisation for Quantum Toolchains. *ACM Trans. Quantum Comput.* 7, 3, Article 18 (July 2026), 46 pages. https://doi.org/10.1145/3800943

1 Introduction

Combinatorial Optimisation Problems (COPs) encode practically relevant problems, such as finding optimal time schedules or routes in planning and logistics. Many practically relevant COPs cannot be solved classically in polynomial time and thus need to be approximated.

There are a multitude of possibilities for (a) encoding a problem mathematically, (b) transforming the encoding into an equivalent representation that can be processed by quantum algorithms (quadratic polynomials are very frequently used for this purpose), and (c) transforming the quantum representation and the description of algorithmic processing steps into hardware-specific instructions. Many of the choices that must be taken during this chain of transformations influence properties like size of the problem representation, the size and structure of required interactions, and eventually also the obtained solution quality and performance.

Quantum compilers (or, more precisely: transpilers) transform a quantum circuit into a hardware-executable representation, which requires, among others, to consider the native hardware gate set into which logical operations must be transformed [46, 59, 63, 67], or which physical qubits can be brought into direct interaction [17, 30, 54, 60, 66, 68]. Transformations that address part (b) in the above list may have to deal with problem representations on higher abstraction layers. For example, a problem may be formulated in the form of a mathematical optimisation problem, like the representation of the join-ordering problem as a **Mixed Integer Linear Program (MILP)**, which can be transformed by discretization into a **Binary Integer Linear Program (BILP)**, which can be transformed into a **Quadratic Unconstrained Binary Optimisation (QUBO)** problem [58]. Starting from a QUBO representation, one has again many choices regarding a concrete solver strategy (*e.g.*, **Quantum Approximate Optimisation Algorithm (QAOA)** [10, 27, 41, 64], Annealing [1, 38, 55], and Grover search [26]), which then need to be transformed to hardware-compatible representations.

Our paper is concerned with a particular transformation to QUBOs, which are a relevant abstraction for quantum computers, since many available hardware vendors only support interactions between a maximum of two qubits [28], which (non-trivially) translate to at maximum quadratic interactions in problems represented as polynomials. The more general form of QUBO is called **Polynomial Unconstrained Binary Optimisation (PUBO)**, which allows for higher-degree interactions.

On the one hand, it is possible to directly encode higher-degree terms in quantum circuits and then use later transpilation steps to decompose them to hardware compatible gates. On the other hand, one can also reduce the degree of higher-degree interactions to quadratic ones and then encode the now quadratic terms in a quantum circuit. We compared these methods for a specific industry-relevant Job-Shop Scheduling problem with regard to QAOA circuits in a previous work [57] by using the framework `quark` [39, 65] and found beneficial effects for the latter reduction variant on

quantum circuit metrics (*i.e.*, number of gates, circuit depth, and gate distribution). In particular, the reviewed existing reduction method in `quark` [39, 65] is able to generate good structural properties. Meanwhile, the classical effort to compute a reduction also needs to be considered to enjoy any advantage gained from using quantum computers. We also showed in [57] that this classical effort is unfeasible with the current implementation for practically relevant problem sizes. In this article, by choosing a suited data structure that only changes locally during updates and considers subsequent steps, we are able to alleviate the influence of classical preparatory effort for general reductions from PUBO to QUBO. We also want to expand upon the application of our approach beyond a particular Job Shop Scheduling formulation in this work by considering a versatile set of problems—namely **Satisfiability (SAT)**.

In classical computing, fault tolerance is a given property of hardware. For a quantum computer—independent of its realisation [11]—it is believed that fault tolerance is the missing integral part of enjoying advantages gained from the fundamentally different computational model [4, 25, 53, 61]. Even if fault tolerant systems are available, it is still necessary to optimise properties of hardware-executable representations, such as above mentioned circuit metrics, since shorter execution times also come with a financial benefit. Even more, early fault-tolerant systems are limited and it is unknown if they can be made immune to environmental noise in a macroscopic scale. Hence, the importance for optimising transformations from high-level representations to low-level hardware-executable representations is pronounced for these upcoming systems.

The rest of this article is structured as follows: Section 2 formalises our problem, establishes notational conventions and reviews existing reduction methods. Section 3 introduces the data structures that our approach is based on and analyses them mathematically, which paves the way for complexity-theoretical performance gains. Section 4 goes into more detail about algorithmic steps and therefore lays the ground for a correctness argument, as well as for a complexity-theoretical analysis in Section 5. Furthermore, Section 5 shows empirically how a concrete implementation performs in comparison to a currently available implementation. Additionally, we show in Section 6 how our approach fits into transformation paths for quantum computers and explicitly consider the effects of transformations on k-SAT instances that are similar to industry instances in depth. We conclude our study in Section 7. The article is augmented by a supplementary website and a comprehensive reproduction package [43] (link in PDF) that allows for extending our work.

2 Mathematical Background

2.1 Pseudo-Boolean Functions

A prominent technique to formulate COPs includes **Pseudo-Boolean Functions (PBFs)** [7, 51, 69]. On the one hand, they are suitable for encoding optimisation problems with $\mathcal{NP}$-complete decision variants, given their complexity-theoretic properties [14, 24]. On the other hand, they provide a consolidated interface to encode many COPs in quantum frameworks [3, 12]—making them a fitting choice for abstraction and ease of integration in a quantum toolchain.

A PBF is a function

$$f : \{0,1\}^n \to \mathbb{R}. \tag{1}$$

Every PBF assumes a multi-linear polynomial representation [9]:

$$f(x_1, \ldots, x_n) = \sum_{S \subseteq \{1,\ldots,n\}} \alpha_S \prod_{j \in S} x_j, \tag{2}$$

where $\alpha_S \prod_{j \in S} x_j$ is called a monomial of f and $\alpha_S \in \mathbb{R}$. We always refer to this representation in the following, since it is unique with respect to monomials with non-zero coefficients α_S. For

example,

$$f(x_1, \ldots, x_6) = \pi x_1 x_2 x_3 - 13 x_2 x_4 x_5 x_6 + 7 x_1 x_3, \tag{3}$$

is a PBF and $\pi x_1 x_2 x_3$, $-13 x_2 x_4 x_5 x_6$ and $7 x_1 x_3$ are monomials of f. We say $m \in f$ for a monomial $m = \alpha_S \prod_{j \in S} x_j$ with index set S, if we have $\alpha_S \neq 0$ in the representation of f according to Equation (2). We also use this notation in particular for "unweighted" monomials with $m = \prod_{j \in S} x_j \in f$. Analogously, we say $x_j \in m$, if $j \in S$ for the corresponding index set S of m. Moreover, we define the degree-k density of f by the ratio of actually present to possible monomials of degree k in f, $d_k = t_k / \binom{n}{k}$, where the degree of a monomial[1] m is the number of variables it contains. For example, for $m = -13 x_2 x_4 x_5 x_6$, we have $\deg(m) = 4$. A short notation for the degree of a monomial is $|m| \coloneqq \deg(m)$. Furthermore, the degree of a PBF f is the maximum degree of its monomials. For the example of Equation (3), we thus have

$$\deg(f) = \max\{\deg(x_1 x_2 x_3), \deg(x_2 x_4 x_5 x_6), \deg(x_1 x_3)\} = 4.$$

In the context of quantum computing, QUBO problems are a highly-used abstraction from hardware-specific peculiarities [24, 37, 52]. They are a standard interface to widely used frameworks, such as quantum annealers [12] and digital annealing [3]. QUBOs are minimisation problems of quadratic PBFs f:

$$\min_{\vec{x} \in \{0,1\}^n} f(\vec{x}).$$

Usually a possibly existing constant term in f (i.e., when $\alpha_\emptyset \neq 0$) is omitted directly from the optimisation problem, since it only shifts the optimisation landscape.

Typically, when formulating optimisation problems for real world applications, higher-degree terms can provide better expressivity and are sometimes necessary to encode constraints [5]. (In-)equality constraints can be encoded in QUBOs by adding suiting penalty terms [24]. For example, it is not trivial to include the absolute value of terms in a PBF. However, one can square terms to achieve a similar effect that, when applied to a series of degree-2 monomials, results in degree-3 and degree-4 monomials. The resulting higher-degree monomials can no longer be directly mapped to QUBO problems and instead require an additional transformation step—also called *quadratisation*.

2.2 Quadratisation

Starting from a higher-degree PBF f, there are many methods, reviewed by Dattani [18], to reduce the degree of f. For example, it is possible to split the objective function [50] or to pre determine variable assignments and then exclude monomials in special cases [31]. A versatile *quadratisation* method, reviewed by Boros [9], can reduce the degree of an arbitrary PBF f to degree-2, *i.e.*, quadratise f, and is thus suited for an automatic transformation. In essence, it works on the multilinear representation of f by iteratively choosing a variable pair $x_i x_j$ and replacing it by a new binary variable y_h. By introducing a constraint term [9]

$$\mathrm{p}(x_i, x_j, y_h) = 3 y_h + x_i x_j - 2 x_i y_h - 2 x_j y_h, \tag{4}$$

which fulfils

$$\begin{aligned} x_i x_j = y_h &\Rightarrow \mathrm{p} = 0 \\ x_i x_j \neq y_h &\Rightarrow \mathrm{p} > 0, \end{aligned} \tag{5}$$

it is possible to preserve the values of f under the minimisation of the newly introduced variable. The constraint term p (also called the penalty) may need to be scaled by a constant $c \in \mathbb{R}^+$ when

[1] A monomial is itself also a PBF.

ALGORITHM 1: Basic steps for iterative *quadratisation*.

Input: PBF f
Output: PBF f' with $\deg(f') \leq 2$, penalty PBF p

1 $h \leftarrow 1$
2 $p \leftarrow 0$
3 **while** $\deg(f) > 2$ **do**
4 $\quad \{x_i, x_j\} \leftarrow$ GET_NEXT_VAR_PAIR(f)
5 $\quad f \leftarrow$ REPLACE_VAR_PAIR(f, x_i, x_j, y_h)
6 $\quad p \leftarrow p + \mathrm{p}(x_i, x_j, y_h)$
7 $\quad h \leftarrow h + 1$
8 **end**
9 **return** f, p

added to the objective function f to achieve the value preservation.[2] Newly introduced variables can be replaced as well, such that, at the end of the iteration, the new PBF is just quadratic but represents the original one. More technically, a PBF $f'(\vec{x}, \vec{y})$ is a *quadratisation* of $f(\vec{x})$, if $f'(\vec{x}, \vec{y})$ is a quadratic PBF ($\deg(f') = 2$) in $\vec{x} = x_1, \ldots, x_n$ and $\vec{y} = y_1, \ldots, y_m$,[3] and satisfies:

$$f(\vec{x}) = \min_{\vec{y} \in \{0,1\}^m} f'(\vec{x}, \vec{y}) \; \forall \vec{x} \in \{0,1\}^n. \tag{6}$$

However, this choice can influence the structural properties of the resulting quadratic PBF f', which can thus translate to varying properties in quantum programs that solve the quadratic PBF f'. Algorithm 1 shows the basic structure of an iterative *quadratisation*, as, for example, done in `quark` [39, 65], where we iteratively choose the next candidate variable pair with GET_NEXT_VAR_PAIR(.), replace it in all monomials of f with REPLACE_VAR_PAIR(.) and add the constraint term (Algorithm 1: l. 6). The function GET_NEXT_VAR_PAIR(.) is the decisive part of varying structural properties in f' and at the same time influences the run time decisively as evaluating the number of occurrences involves inefficient, repeated searching for a variable pair in all monomials of f per iteration in the shown implementation.

In [57] we have already discussed the different choices of the next variable pair in each iteration, that lead to vastly different degree-2 densities d_2 for f':

(1) *Dense*: Choosing the variable pair that appears most often among all monomials.
(2) *Medium*: Choosing the variable pair that appears most often among all highest degree monomials.
(3) *Sparse*: Choosing the first variable pair of a monomial with highest degree.

The *Dense* selection leads to d_2 tending towards 1 and the *Sparse* selection leads to d_2 tending towards 0 for an increasing size of the tested polynomials of degree 4, which stem from a Job-Shop Scheduling problem. This means that the resulting quadratic polynomials are densely and sparsely packed with terms of degree-2, respectively. Whether this convergence behaviour also holds for other polynomials is of interest for this work. However, the time to compute the *quadratisation* using the *Dense* and *Medium* selection type even for small problem instances was shown in [57] to be already in the order of days. Hence, we develop an efficient and more versatile algorithm for reductions and for this introduce an efficient graph structure in Section 3.1 and prove important properties that lay the basis for a complexity theoretic performance gain in Section 3.2.

[2] One could, for example, define c as the sum of all positive monomial coefficients in the non-reduced function f, which may, however, not be optimal for the optimisation landscape in particular with regard to quantum annealers.
[3] Note that it may be necessary to shift variable indices when the same variable name is used.

Table 1. Internal Representation of Example PBF of Equation (3) with the Associated Running Index z, which Uniquely Identifies Each Monomial

Running index z	Index set S	$\prod_{j\in S} x_j$	α_S
1	$\{1,2,3\}$	$x_1x_2x_3$	π
2	$\{2,4,5,6\}$	$x_2x_4x_5x_6$	-13
3	$\{1,3\}$	x_1x_3	7

3 Graph Representation

3.1 Fundamentals

Polynomials can be represented in a variety of ways [36]. In our implementation, to efficiently store the relevant information about the input PBF f, we create a multi-graph $G_f = (V_f, E_f)$ by iterating over all monomials $m \in f$ and adding an edge between nodes i and j if the variable pair x_ix_j is part of m (*i.e.*, $x_i \in m \wedge x_j \in m$). We enrich the graph with further information such that each edge refers to the monomial it stems from via an edge label to be able to differ between multi-edges. This is realised by firstly assigning an arbitrary but fixed and unique index to each monomial $m \in f$, as in Table 1, and secondly using these indices as edge labels. Internally, a dictionary represents this relation—allowing for fast average case access.

More formally, we define the set of edges E_f by including the edge label such that $E_f \subseteq V_f \times V_f \times \mathbb{N}$. For the example of Equation (3), recall that $f(x_1,\dots,x_6) = \pi x_1x_2x_3 - 13x_2x_4x_5x_6 + 7x_1x_3$ and let its monomial index mapping be defined as in Table 1. Then, the set of edges E_f is filled by iterating over all monomials present in f, calculating their variable pair combinations and adding the index to each edge according to the respective monomial:

$$\begin{aligned} E_f = \big\{&(1,2,1),(1,3,1),(2,3,1),\\ &(2,4,2),(2,5,2),(2,6,2),(4,5,2),(4,6,2),(5,6,2),\\ &(1,3,3)\big\}. \end{aligned}$$

It suffices to consider undirected edges for PBFs due to the commutative property of multiplication. In the following, we suppose that for any edge $e = (i,j,z) \in E_f$, it holds that $i < j$. We also exclude self-edges, since $x^2 = x\ \forall x \in \{0,1\}$. We let $E_f^{i,j} = \{(i,j,z) \in E_f\} \subseteq E_f$ denote the set of edges between nodes $i, j \in V_f$. Furthermore, we let the multiplicity $\beta_f(i,j) = \left|E_f^{i,j}\right|$ denote the number of edges between nodes $i, j \in V_f$. An example for the multi-graph representation is depicted in Figure 1.

Let $f : \{0,1\}^n \to \mathbb{R}$ be a PBF and let $G_f = (V_f, E_f)$ be the corresponding graph. We consider the set of all multiplicities in G_f: $B_f = \{\beta_f(i,j) \mid i,j \in V_f\}$. Firstly, f might be constant ($\deg(f) = 0$) or only consist of single variable monomials ($\deg(f) = 1$). In both cases, G_f has no edges and thus $B_f = \{0\}$. Secondly, if $\deg(f) \geq 2$, monomials in f introduce edges to G_f and therefore $a \in |B_f|, a \in \mathbb{N}$. Since a variable pair x_ix_j can at maximum occur in every (at least quadratic) monomial in f, the corresponding multiplicity $\beta(i,j) \leq T_f$, where T_f denotes the total number of monomials in f, and $B_f \subseteq \{1,\dots,T_f\}$. Furthermore, the number of different multiplicities $|B_f|$ is upper bounded by the number of monomials $m \in f$ with $|m| \geq 2$.

We can now define a function R_f that retrieves the node-pairs with multiplicity $\beta \in B_f$:

$$R_f : \beta \mapsto \left\{\{i,j\} : i,j \in V_f, \beta_f(i,j) = \beta\right\}.$$

Take into consideration that R_f maps to disjoint subsets of node-pairs, that is, $R_f(\beta_1) \cap R_f(\beta_2) = \emptyset$ $\forall \beta_1 \neq \beta_2 \in B_f$. By construction, the size of all sets that R_f maps to is given by $\sum_{\beta\in B_f} |R_f(\beta)| = |V_f|^2$.

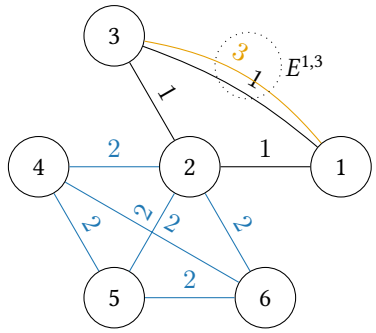


Fig. 1. Multi-graph representation of $f(x_1, \ldots, x_6) = \pi x_1 x_2 x_3 - 13 x_2 x_4 x_5 x_6 + 7 x_1 x_3$, where edge labels correspond to monomial indices according to Table 1.

Table 2. Mapping R_f of Increasing Multiplicity to Sets of Node-Pairs, for Example, of Figure 1

Multiplicity $\beta \in B_f$	Set of node-pairs with multiplicity β *i.e.*, $R_f(\beta)$
0	$\{\{1,4\},\{1,5\},\{1,6\},\{3,4\},\{3,5\},\{3,6\}\}$
1	$\{\{1,2\},\{2,3\},\{2,4\},\{2,5\},\{2,6\},\{4,5\},\{4,6\},\{5,6\}\}$
2	$\{\{1,3\}\}$

Table 2 continues the example from Figure 1 and shows the mapping R for its graph. Note that function R ranks variable pairs based on their occurrence in other monomials to compute the next candidate variable pair. It is motivated by the fact that reducing a variable pair which occurs in many monomials reduces the number of following iterations in the *quadratisation* process on the one hand. On the other hand, we can deliberately select a variable pair, which occurs in less monomials—increasing the number of variables and lowering the resulting PBF's density. Let $B_f = \{\beta_1, \ldots, \beta_{|B_f|}\}$, where $\beta_1 < \ldots < \beta_{|B_f|}$. Hence, $\beta_n \in B_f$ lets us compute the nth multiplicity via a percentile q:

$$\tilde{\beta}_q \coloneqq \beta_{\lceil q \cdot |B| \rceil}. \tag{7}$$

Then, function R_f allows access to node-pairs with such multiplicity:

$$R_f\left(\tilde{\beta}_q\right). \tag{8}$$

With the isomorphic nature of variable pair occurrence in monomials and multiplicity in the graph, we can therefore choose suiting variable pairs based on their occurrence via a percentile q in each iteration.

Furthermore, we propose an algorithm that neither needs to consider $R_f(0)$ (unconnected node-pairs) nor $R_f(1)$ (node-pairs connected by a single edge). This is an important insight for functions f that induce a sparse graph G_f, since the mapping R_f changes during reduction steps—thus saving computational effort, when omitting $R_f(0)$ and $R_f(1)$.[4]

During a reduction, monomials change and thus edges in the graph are removed or added. This leads to changing multiplicities on node-pairs in the graph. Therefore, set B_f changes and node-pairs need to be reallocated to the correct set in $R(\beta)$. Consider that this additional local effort recompenses when searching for the next variable pair, based on its number of occurrences.

[4]We also exclude 0 and 1 from B_f.

3.2 Properties

In the following, we propose a reduction algorithm that iteratively selects a multi-edge (*i.e.*, $e \in E_f^{i,j}$ with $\beta_f(i,j) > 1$) of a starting PBF $f_0 := f$ with $\deg(f) > 2$, reduces the corresponding monomials and updates the graph structure until no multi-edges are left—ending in a PBF f_t after t steps. However, f_t might not be quadratic (*i.e.*, $\deg(f_t) > 2$) as there might be monomials left that do not share variable pairs[5] and thus do not introduce multi-edges in G_{f_t}. Any remaining necessary reduction steps do not introduce multi-edges to the graph corresponding to a subsequent PBF $f_{t+i}, i \in \mathbb{N}$ [57]. We go into more detail about the inner workings of this algorithm in Section 4.

In the following, we list important properties that pave the way for algorithmic optimisation of Algorithm 1 and are the cornerstone for our proposed algorithm in Section 4, as well as for bounding runtime in Section 5. For all properties, we assume that $f : \{0,1\}^n \mapsto \mathbb{R}$ is a PBF and $G_f(E_f, V_f)$ its corresponding graph. Furthermore, we let $x_i x_j$ be the variable pair, that is, going to be replaced in f.

Property 1. If G_f has a node-pair $i, j \in V_f$ with multiplicity $\beta_f(i,j) > 1$, f contains a monomial m with $\deg(m) > 2$. For z being the index of m, the edge $(i,j,z) \in E_f^{i,j}$, that is, it is one of the multiple edges between i and j.

Proof. By construction only degree-k monomials with $k \geq 2$ introduce edges to the graph. With the representation of a PBF from Equation (2), monomials are unique.[6] That means, apart from $x_i x_j$, there is no other degree-2 monomial containing x_i and x_j. Hence, any further edge between nodes i and j must stem from a monomial with degree larger than two, providing the multiplicity larger than one. Let this be m with index z. By construction of G_f, m produces edge (i,j,z) in G_f. □

Property 2. All monomials that need to be updated during that reduction step, correspond to indices on edges between nodes i and j.

Proof. Monomials that do not contain x_i and x_j are by definition invariant under the effect of reduction (see Section 2.2). Hence, it suffices to show that all monomials containing x_i and x_j already occur on edges $e \in E_f^{i,j}$ (*i.e.*, edges between nodes i and j). Without loss of generality, let $m = x_1 x_2 x_3 \dots x_i x_j$, $i < j$ be an arbitrary monomial m, such that $x_i \in m \wedge x_j \in m$ and let z be its corresponding index. It introduces an edge (i,j,z). Furthermore, it introduces edges (a,i,z) and (a,j,z) $\forall a \in \{1,\dots,i-1\}$ or in other words edges that are not between nodes i and j, but are already associated to m on (i,j,z) via z. □

Property 3. The edges introduced by the penalty term $\mathrm{p}(x_i, x_j, y_h) = 3y_h + x_i x_j - 2x_i y_h - 2x_j y_h$, that is, (i,j,z_1), (i,h,z_2), (j,h,z_3) for some z_1, z_2, z_3 not equal to existing running indices (see Table 1) stay invariant under the effect of further reduction steps.

Proof. Since y_h is the newly added variable, (i,h,z_2), (j,h,z_3) are unique (*i.e.*, $\beta(i,h) = \beta(j,h) = 1$). y_h replaces the variable pair $x_i x_j$. Consequently, $x_i x_j$ only occurs in the penalty term, which is quadratic ($\deg(p) = 2$). Hence, (i,j,z_1) represents a single edge in the graph and is thus not a valid choice for an algorithm that only selects multi-edges. Furthermore it is not a valid choice for any subsequent steps, since it is already quadratic. □

As as side note, suppose one chooses the single-edge (i,j,z_1) and therefore the variable pair $x_i x_j$ after it has been reduced. Even then, the edges from the previous reduction remain invariant. The same applies for any other degree-2 monomial from the penalty term (*i.e.*, $x_i y_h$ and $x_j y_h$).

[5] For instance: $x_1 x_2 x_3 \in f_t$ and $x_3 x_4 x_5 x_6 \in f_t$.

[6] Monomials are totalled. For example, let $f(x_1, x_2, x_3) = x_2 x_1$. Then, $f(x_1, x_2, x_3) + x_1 x_2 = 2x_1 x_2$.

Property 4. Edges $e \in E_{f_t}$, not connected to i or j, are invariant under reduction of $x_i x_j$ in $E_{f_{t+1}}$.

Proof. Let $m_t = x_1 x_2 \dots x_k x_i x_j$ $(k < i < j)$ be a monomial and let $m_{t+1} = x_1 x_2 \dots x_k y_h$ be its reduced version. Let $P_S = \{\{i, j\} \mid i \in S \wedge j \in S \wedge i \neq j \wedge \alpha_S \neq 0\}$ be the two combination set of a monomial specified by the subset of indices S. Then,

$$P_{\{1,2,\dots,k\}} = P_{m_t} \setminus \{\{1,i\},\{2,i\},\dots,\{k,i\},\{1,j\},\{2,j\},\dots,\{k,j\}\} = P_{m_{t+1}} \setminus \{\{1,h\},\{2,h\},\dots\}. \quad (9)$$

Remark: m_{t+1} introduces edges to $E_{f_{t+1}}$ between node-pairs $\{\{1,h\},\{2,h\},\dots\}$. □

Property 5. When a node $a \in V_f$ is connected to exactly one of the nodes i or j, its edges are invariant under reduction.

Proof. If an edge $(a, i, z) \in E_f$ is affected by reduction, z must refer to a monomial m containing both x_i and x_j. Without loss of generality, let $m = \dots x_a x_i x_j$. Therefore, $(a, j, z) \in E_f$, which contradicts the assumption that node a is connected to either i or j. This argument is analogous for $(a, j, z) \in E_f$. □

Property 6. We consider nodes i, j and their multiplicity $\beta_f(i, j)$. Then,

$$\beta_f(i,j) \leq \sum_{k=0}^{\deg(f)-2} \binom{n-2}{k} \leq \sum_{k=0}^{n-2} \binom{n-2}{k}. \quad (10)$$

Proof. $\beta_f(i, j)$ depends on the number of monomials $m \in f$ with $|m| \geq 2$ containing the variable pair $x_i x_j$. There are $\sum_{k=0}^{\deg(f)-2} \binom{n-2}{k}$ many such degree-$k+2$ monomials in f at maximum. Therefore, $\beta_f(i,j) \leq \sum_{k=0}^{\deg(f)-2} \binom{n-2}{k} \leq \sum_{k=0}^{n-2} \binom{n-2}{k}$. □

Let f denote a higher-degree PBF and let T_f denote the number of monomials in f. At maximum there are

$$\sum_{\substack{m \in f, \\ |m| \geq 2}} (|m| - 2) = -2T_f + \sum_{\substack{m \in f, \\ |m| \geq 2}} |m|, \quad (11)$$

newly introduced variables—or similarly iterations in the reduction process. Thus, reducing multiple monomials at once, that is, when the candidate variable pair occurs in multiple monomials, decreases the remaining steps by the number of influenced monomials. For example, let $f(x_1, x_2, x_3, x_4) = x_1x_2 + x_1x_2x_3 + x_1x_2x_3x_4$. The maximum number of introduced variables is $0 + 1 + 2 = 3$. When choosing, x_3x_4 as the first reduction pair, x_2x_3 as the second and x_2y_0 as the third, then the above term is sharp: $x_1x_2 + x_1x_2x_3 + x_1x_2x_3x_4 \to x_1x_2 + x_1x_2x_3 + x_1x_2y_1 \to x_1x_2 + x_1y_2 + x_1x_2y_1 \to x_1x_2 + x_1y_2 + x_1y_3$. However, when choosing x_1x_2 as the first reduction pair and x_3x_4 as the second, we can quadratise f in 2 steps: $x_1x_2 + x_1x_2x_3 + x_1x_2x_3x_4 \to y_0 + y_0x_3 + y_0x_3x_4 \to y_0 + y_0x_3 + y_0y_1$.[7] The size of the highest degree monomial gives the minimum number of reduction steps or introduced variables via Boros' method [9], that is,

$$\max_{\substack{m \in f, \\ |m| \geq 2}} (|m| - 2) = \max_{\substack{m \in f, \\ |m| \geq 2}} (|m|) - 2. \quad (12)$$

Although the graph structure is by construction intertwined with the multi-linear polynomial representation of f, we show the necessary steps to arrive at graph $G_{f_{t+1}}$ based on G_{f_t}. This is to simplify the algorithm by concentrating on the graph representation. Consider an arbitrary reduction step from f_t to f_{t+1}. f_t and f_{t+1} both induce a graph, that is, G_{f_t} and $G_{f_{t+1}}$ by construction. We now consider how G_{f_t} can be transformed to arrive at $G_{f_{t+1}}$ without explicitly considering the effects of reduction on monomials in f_t. Figure 2 shows the open relation.

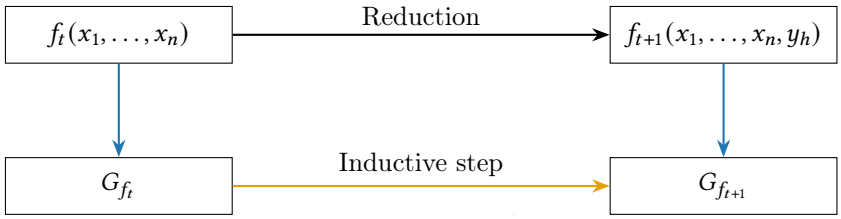


Fig. 2. Known reduction (black), introduced construction (blue), and graph evolution (yellow).

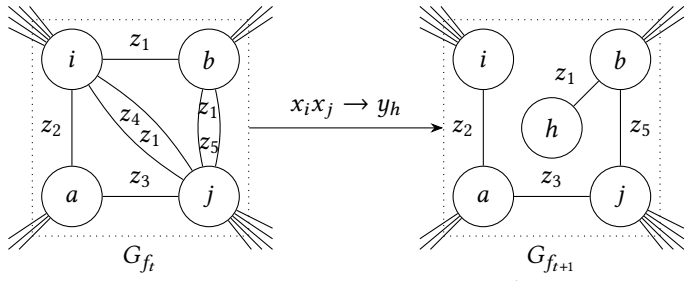


Fig. 3. Shows the inductive graph evolution for changing edges from G_{f_t} to $G_{f_{t+1}}$. The penalty term induced edges $\{(i, h, z_k), (j, h, z_{k+1}), (i, j, z_{k+2})\}$ for a fitting $k \in \mathbb{N}$ can be excluded from $G_{f_{t+1}}$ (see Property 3) and are not shown.

Following Property 4, we only need to consider edges connected to nodes i or j. Following the argument of Property 2, the set of indices on edges between nodes i and j refers to all monomials that need to be updated:

$$Z := \left\{ z \mid (i, j, z) \in E_{f_t}^{i,j} \right\}.$$

Since $z \in Z$ refers to a monomial, that is, affected by the reduction of $x_i x_j$ to y_h, it suffices to remap edges connected to node i or j that contain z to the newly introduced node h. Any neighbouring edge that contains $\bar{z} \notin Z$ is invariant under the effect of reduction, since its corresponding monomial does not contain the variable pair $x_i x_j$. Furthermore, it suffices to consider neighbouring nodes that are connected to both i and j (see Property 5). Figure 3 shows the above stated for a node a, that is, connected to both nodes i and j but does not contain $z \in Z = \{z_1, z_4\}$ on its edges (a, i, z_2) and (a, j, z_3). Hence edges from node a to i and j are invariant under the effect of reduction. Conversely, node b is connected to node i and j and contains $z_1 \in Z$ on its edges (b, i, z_1) and (b, j, z_1). Therefore, these edges are remapped to the newly introduced node h. The remaining edge (b, j, z_5) is invariant, since it stems from a monomial that does not contain the variable pair $x_i x_j$. In summary, the graph structure can be evolved without specifically referring to monomials and propagating their changes. This is mainly due to the fact that (a) all changing monomials are identified by indices occurring on edges between nodes i and j (see Property 2) and (b) a reduction acts on the local neighbourhood of nodes i and j.

4 Algorithm in Detail

Any node-pair $\{i, j\}$ with $\beta_f(i, j) > 1$ is suited for a reduction step, since it is guaranteed to originate from a higher-degree monomial m ($\deg(m) > 2$; see Property 1). We cannot state that for node-pairs connected by a single edge (*i.e.*, $\beta_f(i, j) = 1$) in general, since they might stem from a degree-2

[7]These examples do not show the already quadratic penalty term.

monomial. Hence we introduce the **Local Structure Reduction (LSR)** algorithm—subdivided into two stages:

LSR stage 1: Graph-based reduction
LSR stage 2: Independent monomial-based reduction

In stage 1, the graph structure is decisive in the performance gain and in stage 2, we no longer need the graph structure—although it proves useful in the runtime analysis in the following section. After first applying stage 1 and then stage 2 to a higher-degree PBF f, we arrive at a quadratic PBF, which adheres to the *quadratisation* criteria in Equation (6).

Recall the basic structure for a *quadratisation* algorithm (Algorithm 1), that is, (a) choosing a variable pair according to certain criteria and (b) replacing it with a new variable in the higher-degree function and thereby adding a penalty term. Motivated by the adaptability to degree-2 density in the resulting quadratic function, it is eminent to choose the next variable pair according to their occurrence in other monomials. Recall that this property is depicted by multiplicities in the graph. Hence, we alter the existing algorithmic structure in part (a) while making sure to advance the graph structure as shown in Section 3. Part (b) still results in the variable pair being replaced in all occurring monomials and is therefore left unchanged in terms of results.

Our proposed algorithm chooses the next variable pair in stage 1 according to a sorted set of multiplicities in the graph via percentile q. Recall that $B \coloneqq \{\beta_1, \ldots, \beta_{|B|}\}$ is the set of multiplicities in G and $\beta_1 < \ldots < \beta_{|B|}$, as in Section 3. We define $\tilde{\beta}_q \coloneqq \beta_{\lceil q \cdot |B| \rceil}$ for a percentile q (see Equation (7)). A percentile $q = 1$ results in a variable pair that occurs most often in all monomials, while a percentile $q = 0.5$ results in choosing the median. Recall that we exclude multiplicities 0 and 1 (see Section 3) from function R and multiplicity set B. Hence, choosing $q = 0$ will result in choosing a variable pair, that occurs in at least two monomials[8] m with $\deg(m) \geq 2$. Since function $R(\tilde{\beta}_q)$ returns a set of node-pairs with multiplicity $\tilde{\beta}_q$, we can choose a random element from it as the next variable pair.

After choosing a variable pair, we can alter the graph structure independently from the monomial reduction according to the introduced inductive step in Section 3. Function UPDATE_GRAPH_DATA(.) firstly selects monomial indices from edges between nodes i and j in set Z (Algorithm 2: l. 18), since all changing monomials occur on these edges (see Property 2). According to Property 5 we can restrict the search for changing edges to nodes connected to both node i and j. This property applies to nodes that we eventually save in set N (Algorithm 2: l. 19)—the set of nodes with changing edges. Without explicitly accessing the set of neighbours for nodes i and j, we can pre-compute changing edges in G. This method depicts a lower bound on finding changing edges in G for an iteration step, as it is linear in the number of changing edges. To clarify this point, let $m_z = x_1x_2...x_kx_ix_j$ $(k < i < j)$ be a monomial, let z be its index in Z, let x_ix_j be the chosen reduction pair and let $x_1x_2 \ldots x_ky_h$ be its reduced version (see Property 4 for a similar argument). Then, E_{removed} denotes the set of removed edges from G and E_{added} denotes the set of new edges to G or in other words the remapping of a subset of edges induced by monomial m_z[9]:

$$\begin{aligned} E_{\text{removed}} \coloneqq \{&(1,i,z),(2,i,z),\ldots,(k,i,z)\\ &(1,j,z),(2,j,z),\ldots,(k,j,z),(i,j,z)\} \\ E_{\text{added}} \coloneqq \{&(1,h,z),(2,h,z),\ldots,(k,h,z)\}. \end{aligned}$$

As a consequence of replacing edges (k,i,z) and (k,j,z) with (k,h,z) (Algorithm 2: ll. 24–25; 29), multiplicities change according to:

[8]It may be the case that $m_1 = x_ix_j$ and $\deg(m_2) > 2$.
[9]Take into consideration that for simplicity in pseudocode, we do not enforce a total ordering of indices in edges, as introduced in Section 3.

ALGORITHM 2: Local Structure Reduction (LSR).

Input: PBF f, percentile q ▷ $\deg(f) > 2, q \in [0, 1]$
Output: quadratic PBF f', penalty PBF p

▷ Stage 1: Graph-based reduction

1 $h \leftarrow 1$
2 $p \leftarrow 0$
3 $G = (V, E), R \leftarrow G_f = (V_f, E_f), R_f$
4 **while** G contains multi-edges ($\exists i, j \in V : \beta(i, j) > 1$) **do**
5 $\quad \{i, j\} \leftarrow$ CHOOSE_RANDOM_ELEMENT$(R(\tilde{\beta}_q))$
6 $\quad f \leftarrow$ REPLACE_VAR_PAIR(G, f, x_i, x_j, y_h)
7 $\quad G, R \leftarrow$ UPDATE_GRAPH_DATA(G, R, i, j, h)
8 $\quad p \leftarrow p + \mathrm{p}(x_i, x_j, y_h)$
9 $\quad h \leftarrow h + 1$
10 **end**

▷ Now we have $\forall i, j \in V : \beta(i, j) \leq 1$
▷ Stage 2: Independent monomial-based reduction

11 **for** $m \in f$ *with* $\deg(m) > 2$ **do**
12 $\quad$ **while** $\deg(m) > 2$ **do**
13 $\quad\quad f, p \leftarrow$ MULTI_REDUCE(f, p, m)
14 $\quad$ **end**
15 **end**
16 **return** f, p

17 **Procedure** UPDATE_GRAPH_DATA*(G, R, i, j, h)*
18 $\quad Z \leftarrow \{z : (i, j, z) \in E\}$
19 $\quad N \leftarrow \{h\},\ E_{\text{removed}} \leftarrow \{\},\ E_{\text{added}} \leftarrow \{\}$
20 $\quad$ **for** $z \in Z$ **do**
21 $\quad\quad$ **for** $k \in m_z \wedge k \neq h$ ▷ m_z retrieves monomial indexed by z
22 $\quad\quad$ **do**
23 $\quad\quad\quad N \leftarrow N \cup \{k\}$
24 $\quad\quad\quad E_{\text{removed}} \leftarrow E_{\text{removed}} \cup \{(k, i, z)\} \cup \{(k, j, z)\}$
25 $\quad\quad\quad E_{\text{added}} \leftarrow E_{\text{added}} \cup \{(k, h, z)\}$
26 $\quad\quad$ **end**
27 $\quad$ **end**
28 $\quad M \leftarrow \{(\beta(n, k), n, k) : n \in N, k \in \{i, j\}\}$
29 $\quad E \leftarrow \left(E \setminus (E_{\text{removed}} \cup E^{i,j})\right) \cup E_{\text{added}}$
30 $\quad R \leftarrow$ UPDATE_R(G, R, N, M, i, j, h)
31 **return** G, R

$$\begin{aligned} \beta(k, i) &\leftarrow \beta(k, i) - 1 \\ \beta(k, j) &\leftarrow \beta(k, j) - 1 \\ \beta(k, h) &\leftarrow \beta(k, h) + 1. \end{aligned} \tag{13}$$

Also, edges between nodes i and j are removed[10] (Algorithm 2: l. 29). Last but not least, function R changes, when the graph is altered (see Equation (13)). Since node-pairs are mapped from the set of former multiplicities in G, set M saves their values and corresponding node-pairs. Therefore,

[10]We exclude the penalty term (see Property 3).

any node-pair in M can firstly be deleted from R and secondly be added with its new multiplicity again (Algorithm 2: l. 28): In code, R can be implemented by a sorted dictionary of sets, which leads to logarithmic access times on multiplicities and to constant average access times on elements of the sets of node-pairs. Although the sorted property is not needed when updating the graph, it is required to select the next variable pair via percentile q, which deliberately influences properties of the resulting PBF.

Since we independently evolve the graph structure, it is necessary to update the polynomial representation separately. As an improvement to the standard implementation that needs to go over all monomials of f, we can use the graph structure to directly address only changing monomials. Function REPLACE_VAR_PAIR(.) (Algorithm 2: l. 6) takes advantage of Property 2 to identify all changing monomial indices by gathering all indices on edges between nodes i and j. Based on these indices, a dictionary implementation of Table 1 leads to the monomial representation of changing monomials. Since edges induced by the penalty term are invariant under reduction of further steps (see Property 3), it is excluded from the graph structure. Furthermore, it is saved in a separate PBF (Algorithm 2: l. 8) to be able to later scale it properly and therefore adhere to the *quadratisation* criteria given in Equation (6). For as long as there are remaining multi-edges in the graph, stage 1 continues this process. We know that this stage terminates, as we have shown in the supplementary material in a prior publication, by showing that the total number of edges in multi-edges decreases monotonically [57].

As a prerequisite of stage 2 (Algorithm 2: l. 11), we know that there are no more multi-edges in the graph:

$$\forall i, j \in V : \beta(i, j) \leq 1.$$

This means that there is no monomial left that shares a variable pair with other monomials. Since remaining reduction steps do not introduce multi-edges again [57], any two monomials will not share variable pairs for the remaining reduction steps. Hence, it is possible to reduce monomials independently of each other. Function MULTI_REDUCE(.) (Algorithm 2: l. 13) quadratises a monomial m and adds the necessary penalty terms to p. Hereafter (Algorithm 2: l. 16), input function f is now a quadratic PBF, constrained by the penalty term p.

5 Analysis

5.1 Correctness

Let $f : \{0,1\}^n \to \mathbb{R}$ be a PBF with $\deg(f) > 2$. We argue that the quadratised function f' resulting from f through our introduced LSR method adheres to the *quadratisation* criteria, given in Equation (6). The graph structure improves performance when finding the next variable pair, but does not affect f. Since the replacement of variable pairs in f and introduction of penalty term p adheres to the standard method (see Section 2.2), the introduced LSR algorithm adheres to the *quadratisation* criteria in Equation (6) [9]. Take into consideration that p may need to be scaled by a constant c. The minimum value for c is problem dependent and thus we leave this last step to the user by returning p unscaled.

5.2 Runtime Analysis

In this section, we characterise the runtime complexity of the existing-monomial-based reduction as implemented in `quark` (link in pdf) and the new LSR graph-based reduction. The basis for the main comparison is the runtime per iteration, but we also include estimations for complete runtimes. The main difficulty is that these algorithms can lead to different *quadratised* functions f depending on the lexicographical order in each step. Let $f_0 := f : \{0,1\}^n \to \mathbb{R}$ be a PBF such that $\deg(f) > 2$ and let T_f denote the number of monomials in f. Furthermore, let f_t denote a PBF after reduction

step $t \in \mathbb{N}$.[11] We define the size of the input function f_0 by the number and size of it's monomials, that is, $\sum_{m \in f_0} |m|$, where we write $|m|$ as a short version of $\deg(m)$.

5.2.1 Monomial-based Reduction. The monomial-based reduction has three variants: *Sparse*, *Medium*, and *Dense*. Each iteration consists of a two-stage process, that is, (a) searching for the next variable pair and (b) replacing that pair in every occurring monomial (see Algorithm 1). Part (b) is variant-independent and its runtime for a reduction step from f_{t-1} to f_t is given by:

$$\begin{aligned} \sum_{m \in f_{t-1}} 2|m| &\leq T_{f_{t-1}} \cdot 2 \max_{m \in f_{t-1}} |m| = T_{f_{t-1}} \cdot 2 \deg(f_{t-1}) \\ &\leq T_{f_{t-1}} \cdot 2 \deg(f_0) \leq T_{f_{t-1}} \cdot 2n. \end{aligned} \tag{14}$$

Here, it is required to check if the two candidate variables are present in each monomial of f_{t-1}. In the monomial-based array implementation of monomials [39, 65] the search for the next variable pair leads to a full traversal of each monomial. We also use the fact that a reduction step does not increase the degree of f.

In the following RT_{Sparse} and RT_{Dense} refer to the search in a single iteration for the *Sparse* and *Dense* variants. Part (a) is variant-dependent. In particular, the *Sparse* method searches for the highest-degree monomial. Without caching monomials, the search for the highest-degree monomial takes

$$\mathrm{RT}_{Sparse}(f_{t-1}) = T_{f_{t-1}}, \tag{15}$$

steps.[12] The *Dense* method searches for the variable pair that appears most often among all monomials. It therefore computes all **Pair Combinations (PCs)** of every monomial in f_t. Let PC_m denote the set of pair combinations resulting from a monomial $m \in f_{t-1}$. For any degree-k monomial m (*i.e.*, $|m| = k$) there are $\binom{|m|}{2}$ variable pairs and hence, the number of pairs to consider is $\sum_{m \in f_{t-1}} |\mathrm{PC}_m| = \sum_{m \in f_{t-1}} \binom{|m|}{2}$. Let **Unique pair combinations (UPCs)** denote the set of unique variable pairs: $\mathrm{UPC}_{f_t} = \bigcup_{m \in f_t} \mathrm{PC}_m$. By iterating over variable pairs $u \in \mathrm{UPC}$ and calculating the pair combinations of monomials in f_{t-1}, we get the count for u. Sorting the resulting list of counts by u, gives the total runtime for searching for the most occurring pair among all monomials $\mathrm{RT}_{Dense} = \log\left(|\mathrm{UPC}_{f_{t-1}}| \cdot \sum_{m \in f_{t-1}} \binom{|m|}{2}\right) \cdot |\mathrm{UPC}_{f_{t-1}}| \cdot \sum_{m \in f_{t-1}} \binom{|m|}{2}$. In an earlier work [57], we show that the total size of multi-edges decreases with every reduction step. Although the monomial-based method does not use a graph structure, we can still apply our theorem: Since the *Dense* variant searches for the most occurring pair, the preliminary condition of selecting multi-edges in the graph applies. Therefore, $|\mathrm{UPC}_{f_{t-1}}| \geq |\mathrm{UPC}_{f_t}| \, \forall t \in \mathbb{N}$ and the runtime for the *Dense* search in reduction step t is given by:

$$\begin{aligned} \mathrm{RT}_{Dense}(f_{t-1}) &\leq \log\left(|\mathrm{UPC}_{f_0}| \cdot \sum_{m \in f_{t-1}} \binom{|m|}{2}\right) \cdot |\mathrm{UPC}_{f_0}| \cdot \sum_{m \in f_{t-1}} \binom{|m|}{2} \\ &\leq \log\left(n^2 \cdot \sum_{m \in f_{t-1}} \binom{|m|}{2}\right) \cdot n^2 \cdot \sum_{m \in f_{t-1}} \binom{|m|}{2}. \end{aligned} \tag{16}$$

5.2.2 Graph-based Reduction.

Stage 1: Graph-based Reduction. Firstly, we show that the preparatory effort to compute the needed data structures does not exceed $O(T_{f_0} \cdot \binom{\deg(f_0)}{2} + n^2 \cdot \log(T_{f_0}))$. Secondly, we examine the effort to compute the inner part of the while loop (*i.e.*, a reduction iteration; see Algorithm 2: ll. 5-9).

[11] $f_0 \overset{\text{1. reduc step}}{\longrightarrow} f_1 \overset{\text{2. reduc step}}{\longrightarrow} f_2 \longrightarrow \cdots \longrightarrow f_{t-1} \overset{t\text{th reduc step.}}{\longrightarrow} f_t$.

[12] As for an implementation in python, the LEN(.) function accesses a cached attribute and therefore has a time complexity of $O(1)$.

Creating the monomial index dictionary, as in Table 1, iterates over all monomials present in f_0, that is, the size of the input dictionary: $O(\sum_{m\in f_0}|m|) \subseteq O(T_{f_0}\cdot\binom{\deg(f_0)}{2})$. Creating the graph structure requires iterating over all variable pair combinations per degree-k monomial m, of which there are $\binom{|m|}{2}=\binom{k}{2}$ many. In total, the number of edges in the graph is upper bounded by $O(\sum_{m\in f_0}\binom{|m|}{2}) \subseteq O(T_{f_0}\cdot\binom{\deg(f_0)}{2})$. Last but not least, creating function R, as in Section 3.1, originates from the graph's connected node-pairs, and therefore needs $O(|V_{f_0}|^2)=O(n^2)$ computational steps. Recall that function R is implemented as a sorted dictionary of sets. Each insertion has the potential to generate a new entry in the sorted dictionary, for which the access and insertion times are in $O(\log(n))$ [32]. Let the set of multiplicities be denoted by $B_{f_0}=\{\beta_{f_0}(i,j)\mid i,j\in V_{f_0}\}$ (see Section 3). To be more precise, the access time in the sorted dictionary (or domain of R_{f_0}) is given by $O(\log(|B_{f_0}|)) \subseteq O(\log(T_{f_0}))$.[13] Therefore, creating function R is upper bounded by $O(n^2\cdot\log(T_{f_0}))$. Hence, the preparatory effort to compute the needed data structures is bounded by:

$$O\left(T_{f_0}\cdot\binom{\deg(f_0)}{2}+n^2\cdot\log(T_{f_0})\right). \tag{17}$$

Computing the index to access the desired subset of nodes in R via percentile q is done in constant time. Using an iterator to access a random element of that subset also leads to a constant access time. Thus, CHOOSE_RANDOM_ELEMENT($R(\tilde{\beta}_q)$) (Algorithm 2: l. 5) is upper bounded by $O(\log(|B_{f_{t-1}}|)) \subseteq O(\log(T_{f_0}))$ with $B_{f_{t-1}}$ analogous to above. Exploiting the local influence of a reduction's iteration, that is, restricting the number of processed edges to the local neighbourhood of nodes i and j (assuming x_ix_j is going to be reduced), is decisive for the algorithm's performance gain. Note that this not only includes extracting relevant edges, but also includes the update procedure on edges and monomials in the polynomial representation. Let the set of indices referring to changing monomials be denoted by Z (Algorithm 2: l. 18). These indices occur on edges between nodes i and j and thus it takes $\beta_{f_{t-1}}(i,j)$ steps to compute Z—assuming an average case access time on the graphs edges of $O(1)$. The worst case runtime is given by $O(\beta_{f_{t-1}}(i,j)\cdot|V_{f_{t-1}}|^2)$.

Ultimately, the set of edges that change during this reduction iteration is relevant. Recall that these edges are split into sets E_{removed} and E_{added}. Property 4 states that any changing edge needs to be connected to i or j, which allows us to compute the set of removed and new edges E_{removed} and E_{added}, respectively, in $O(\sum_{z\in Z}|m_z|)$, where $\beta_{f_{t-1}}(i,j)=|Z|$ by iterating over each affected monomial once. m_z refers to the variable representation of a monomial m indexed by z. Due to—on average—constant access times in sets, this is an upper bound on the for-loop's runtime (Algorithm 2: ll. 20–27). Updating the edges of G (Algorithm 2: l. 29) is also bounded by above runtime, since the average runtime to additionally remove edges between nodes i and j is given by $\beta_{f_{t-1}}(i,j)$.

It is necessary to recalculate the multiplicity between node-pairs corresponding to edges that have changed during this iteration. These node-pairs can be extracted from set N in $O(\sum_{z\in Z}|m_z|)$ (Algorithm 2: l. 28). Due to the dictionary-like structure for the graph, computing the number of edges between two nodes is equal to accessing a cached property, namely the number of keys in the dictionary (*i.e.*, $O(1)$ (average case); $O(|V_{f_{t-1}}|^2)$ (worst case)). Let $B_{f_{t-1}}=\{\beta_{f_{t-1}}(i,j)\mid i,j\in V_{f_{t-1}}\}$ denote the set of multiplicities in the graph resulting from f_{t-1}. Recall that each element in the sorted dictionary implementation of function R can be accessed in $O(\log(|B_{f_{t-1}}|)) \subseteq O(\log(T_{f_{t-1}}))$. This results in a total runtime of $O(\log(|B_{f_{t-1}}|)\cdot\sum_{z\in Z}|m_z|)$ for UPDATE_R(.) (Algorithm 2: l. 30). Take into consideration that $|B_{f_t}|\,/\,|B_{f_{t-1}}|<c_{f_0}$ where c_{f_0} is a constant for input function f_0, since a reduction step acts locally (see Property 4 and Figure 3). Let ω be the index of the last reduction iteration in LSR stage 1. Then, $|B_{f_\omega}|=0$.

[13] Let $\beta_{\max}$ (compare to Property 6) denote the maximum number of edges between two nodes in G. Take into consideration that not every value in $\{0,\ldots,\beta_{\max}\}$ needs to be present in B_{f_0}—thus improving the logarithmic access time.

As a side note, suppose that the graph of a PBF f contains most of its edges between nodes i and j. One can argue that the effort to remove and add edges in this iteration scales with $O(|E_f|)$. However, recall that the total number of iterations is bounded (see Section 3.2). Since every edge $E_f^{i,j}$ (see Section 3) is removed,[14] this special case is in fact efficient in terms of introducing less variables.

Assuming an array implementation of monomials, it takes $O(|m|)$ steps to replace a variable pair in a monomial. Since we store a monomial an edge stems from indirectly (see Table 1), changing monomials in the reduction process do not alter the graph's structure—thus saving significant access time.[15] It suffices to change monomials in the index dictionary of f (see Table 1) with an average access time of $O(1)$, whereas the worst case access time is $O(T_{f_{t-1}}) \subseteq O(T_{f_0})$. Thus, the runtime of REPLACE_VAR_PAIR(.) (see Algorithm 2: l. 6) is upper bounded by $O(T_{f_0} \cdot \sum_{z\in Z} |m_z|)$ (worst case) and on average $O(\sum_{z\in Z} |m_z|)$. On average, dictionary insertion is achieved in $O(1)$ and in the worst case in $O(n)$. Thus, adding the penalty term (Algorithm 2: l. 8) has an average runtime of $O(1)$ and a worst case runtime of $O(t)$, since the penalty term introduces 4 monomials per iteration (see Equation (4)).

In a complete reduction iteration (Algorithm 2: ll. 5–9), function UPDATE_R(.) (Algorithm 2: l. 30) has the highest runtime—leaving an average case runtime of

$$\mathrm{RT}_{\mathrm{LSR1}} \in O\left(\log(|B_{f_{t-1}}|) \cdot \sum_{z\in Z} |m_z|\right), \tag{18}$$

per full iteration. Since the number of multi-edges in the graph strictly decreases with every reduction iteration [57], we can bound the number of iterations in Algorithm 2 to $|E_{f_0}|$.

Stage 2: Independent Monomial-based Reduction. Let $f_t : \{0,1\}^n \to \mathbb{R}$ be a PBF resulting from stage 1, that is, a function sharing no variable pairs among its monomials. Let $m = x_1x_2x_3x_4x_5 \in f_t$ be a monomial. In contrast to stage 1, we can no longer exploit replacing the same variable pair in different monomials. Therefore, using Boros' [9] reduction method, we can apply multiple reductions at once. For example, applying $x_1x_2x_3x_4x_5 \to y_1y_2x_5$ in a single step by replacing x_1x_2 with y_1 and x_3x_4 with y_2. MULTI_REDUCE(.) (Algorithm 2: l. 13) uses this fact and therefore has a time complexity of $\Theta(|m|)$. After applying this step once, only degree-3 and degree-4 monomials (*i.e.*, $|m| \in \{3,4\}$) become quadratic in general. Further steps are needed for higher-degree monomials. To be more precise, we differentiate two cases:

$$|m_{\mathrm{new}}| = \begin{cases} 0.5 \cdot |m|, & \text{if } |m| \text{ is even} \\ \lceil 0.5 \cdot |m| \rceil, & \text{if } |m| \text{ is odd} \end{cases}, \tag{19}$$

where m_{new} represents the monomial after MULTI_REDUCE(.) is applied to m. Instead of computing the ceiling function, we can depict the number of iterations necessary for m to become quadratic as $\tau = \log_2(\lfloor |m| \rfloor_2) = \lfloor \log_2(|m|) \rfloor$, where $\lfloor \cdot \rfloor_2$ rounds down to the nearest power of 2 and $|m| \in \mathbb{N}, |m| > 4$. Hence, τ is logarithmic in the monomial's size (*i.e.*, $\tau \in \Theta(\log_2(|m|))$)—leading to a partial geometric series for the runtime of the inner while-loop (Algorithm 2: l. 12–14) and, due to its convergence, a total runtime of

$$\mathrm{RT}_{\mathrm{LSR2}} \in \Theta(|m|), \tag{20}$$

per full monomial *quadratisation*. The summation over all left-to reduce monomials (*i.e.*, $|m| > 2, m \in f_t$) leads to the total runtime of stage 2. As a side note, we are not required to quadratise monomials in MULTI_REDUCE(.), but can stop the reduction process at an arbitrary point—leaving us with a degree-k, $k > 2$ function.

[14] We exclude the penalty term from the graph structure.

[15] Note that this statement assumes the specific algorithm's reduction process.

Since there are no common variable pairs among monomials in f_t, there are at maximum $O(|E_{f_t}|) \subseteq O(|V_{f_t}|^2)$ many monomials left to reduce in stage 2 and therefore as many necessary iterations in the for loop (Algorithm 2: l. 11–15).[16] Take into consideration that the previous stage 1 algorithm changes monomials through replacements. Thus, it reduces the size of monomials for stage 2, which lowers the runtime of the inner while loop (Algorithm 2: l. 12–14).

Despite changing the number of variables, stage 1 does not increase the number of monomials in the polynomial dictionary of f, since the penalty term is saved separately. On the one hand, this upper bounds the iteration count of the for loop in stage 2 (Algorithm 2: l. 11–15) to T_{f_0}. On the other hand, the total runtime for LSR stage 2 is given by:

$$\Theta\left(\sum_{\substack{m\in f_t,\\ |m|>2}} |m|\right). \tag{21}$$

Let $m = x_1 \dots x_k$ be a degree-k monomial (*i.e.*, $|m| = k$), that is, left to reduce from stage 1. It introduces $\binom{k}{2}$ edges to the graph. Since the graph for stage 2 has no multi-edges, the total runtime for stage 2 is upper bounded by:

$$O(E_{f_t}) \subseteq O(|V_{f_t}|^2). \tag{22}$$

One can parallelise stage 2 for any monomial $m \in f_t$, thus reducing the runtime to $O((T_{f_0}/W) \cdot$ <inner loop> + <distribute/gather>), where W denotes the number of worker tasks and T_{f_0} denotes the number of terms in f_0. Parallelisation is also possible in stage 1 (see Property 4): Node-pairs that are not connected to each other do not influence each other during a reduction step.

5.3 Average Runtime Guarantees and Optimality

Recall that a reduction iteration is characterised by two steps, that are, (a) finding the next variable pair and (b) replacing that pair by a new variable in all monomials it occurs in (see Algorithm 1). Let $f : \{0,1\}^n \mapsto \mathbb{R}$ be a PBF with T_f many monomials, let $G_f(V_f, E_f)$ be the corresponding graph of f and let $\beta_f(i,j)$ denote the number of occurrences of the variable pair $x_i x_j$ in monomials of f. The trivial lower bound of performing an iteration (*i.e.* (a) and (b)) is $\Omega(\beta_f(i,j))$, since we need access to all monomials in which $x_i x_j$ occurs. When accessing variables in a monomial m, where each variable is saved as an element of an array, the worst case access time is $O(|m|)$. Using a sorted array, it reduces to $O(\log(|m|))$, due to binary search. Another option is to use a lookup-table per monomial where variable index x is saved at position x, that is, mapping each variable in a monomial to the space of variables in the function. Therefore, we can guarantee an access time of $O(1)$, but use extra space, that is, $O(n)$ per monomial, where n is the number of variables in f. Hence, this method needs $O(n \cdot T_f)$ space in total to store a PBF. Recall that we defined the size of the input function f as the total size of its monomials, that is, $\sum_{m\in f} |m|$. Therefore, the extra space ($O(n \cdot T_f)$) needed is at most quadratic in the input size—assuming there are no unused variables in f. For practically relevant input functions f (*i.e.*, $n \ll T_f$), the extra space is sub-quadratic in the input size.

So far, we focused on the lower bound of (b), which corresponds to replacing the variable pair. Now, we consider searching for the next pair, which corresponds to finding a lower bound for (a), and requires a clear definition of required properties for the next pair. The trivial lower bound for an ambiguous choice is $\Omega(\sum_{m\in f_0} |m|)$ for the whole *quadratisation* process, since every monomial has to be considered at least once. Our proposed method needs to be able to differentiate between the number of occurrences of a variable pair in each iteration—motivated by the influence to the

[16] Note that, in stage 2, the graph does not contain multi-edges.

quadratised function (*i.e.*, its density, number of variables, and its monomial distribution among variables). Assuming a sorted array implementation of the number of occurrences, function R's implementation already achieves optimal runtime (avg. case) in each iteration in the first stage, due to binary search.

Assuming an array implementation of each monomial, the lower bound for part (b) rises to $\Omega(\sum_{z\in Z}|m_z|)$, where $|Z| = \beta_{f_{t-1}}(i,j)$, since searching in an unsorted array with size $|m|$ takes $O(|m|)$ steps. Take into consideration that, as mentioned above, an array implementation can be replaced by more efficient data-structures—therefore lowering the lower bound to $\Omega(\beta_{f_{t-1}}(i,j))$.

Other custom hash functions can be used to guarantee average case runtime in the mentioned data structures in Section 5.2. On the one hand, indexing monomials in a dictionary with exactly T_f (see Table 1) many entries (*i.e.*, the number of monomials in the input function[17]), enables us to use $f(x) = x$ as a hash function, without spacial overhead. On the other hand, each monomial can at most introduce a single edge between a particular node-pair in the graph. Hence, the maximum multiplicity is given by at most T_f (see Property 6). Therefore, $f(x) = x$ is a suitable hash function for the sorted dictionary of function R — taking no more space than the input function has monomials. Take into consideration that the size of the input function additionally includes the size of its monomials. Function R maps to at most $|V_{f_{t-1}}|^2$ node-pairs.[18] Unfortunately, $|V_{f_{t-1}}|^2$ scales with the number of iterations. Recall that we can, however, bound the maximum number of iterations I_f by $\sum_{\substack{m\in f_0,\\ |m|\geq 2}}(|m|-2) = -2T_{f_0} + \sum_{\substack{m\in f_0,\\ |m|\geq 2}}|m|$. Since the set of nodes V_f is isomorphic to the set of variables in f, the space complexity is $O((n+I_f)^2) \subseteq O((n-2T_{f_0}+\sum_{m\in f_0}|m|)^2)$.

Another type of perfect hash functions (*i.e.*, functions used whenever $O(1)$ worst case accesses are required) use two stage universal hashing [16]. This method guarantees a linear space complexity in the number of keys—although requiring a static set of keys to carefully choose the primary and secondary stage hash functions. The monomial index dictionary provides a static set of keys, since the penalty term is separately stored (see Property 3) and is therefore a candidate for this type of hashing. Despite not having a static set of keys, the dictionary of the input polynomial provides a static number of keys. Take into consideration that this type of hashing introduces randomised data structures.

5.4 Experimental Results

Although the asymptotic performance is most relevant in a complexity theoretic analysis, we also want to characterise the practical performance by evaluating a concrete implementation and thereby incorporating constant factors, structural effects of input functions and taking into account that the theoretical analysis overestimates the runtime. In the following, we test randomly chosen PBFs $f : \{0,1\}^n \mapsto \mathbb{R}$ such that $\deg(f) = 4$ and varying densities such that $d_1(f) = d_2(f) = d_3(f) = d_4(f)$ with the monomial-based implementation and the new LSR method (graph_based). Take into consideration that we do not use custom hash-functions and no parallelisation, as proposed in Sections 5.3 and 5.2, but a standard python dictionary implementation. Therefore, we expect the following scaling behaviour in runtime to be an upper bound.

Figure 4 compares the runtime for the introduced LSR algorithm (graph_based) and the monomial-based algorithm that uses a brute force search for the next variable pair. Since the *Dense/better* selection type of the monomial-based algorithm searches for a variable pair, which occurs most often among all monomials, it is comparable to the selection percentile $q = 1.0$ in the new algorithm. The x-axis shows the number of variables prior to reduction (*i.e.*, n) and the y-axis (log scale) shows the runtime in a logarithmic scale. Different PBF densities are shown as vertical facets. The

[17]The penalty dictionary is separable from it: see Property 3.
[18]We exclude node-pairs, containing less than two edges.

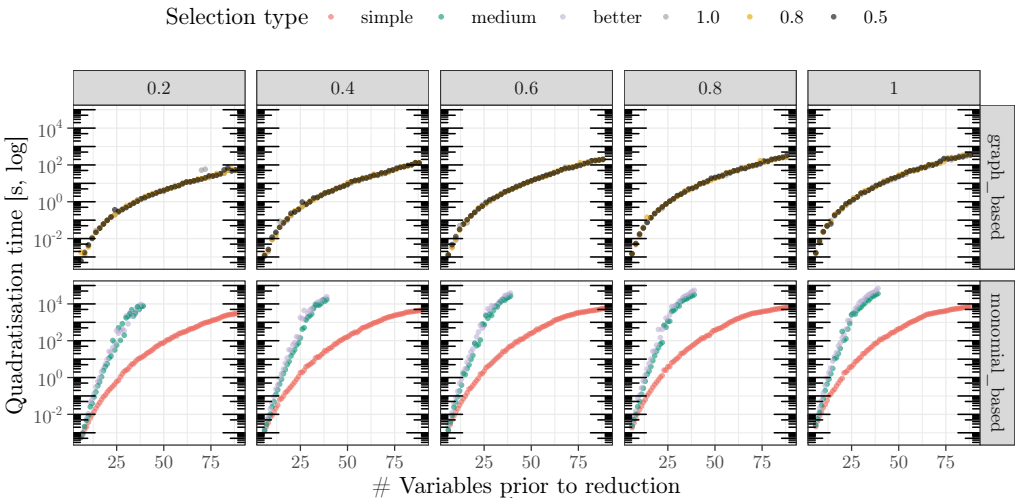


Fig. 4. Time in seconds (y-axis) for the *quadratisation* of a $\deg(f) = 4$ function f vs problem size (x-axis: Number of variables). New LSR algorithm (top row) compared to existing monomial-based (bottom row). Vertical facets: Different base polynomial densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f) \in \{0.2, 0.4, \ldots, 1.0\}$). Selection type 1.0 (LSR algorithm) comparable to *Dense/better* (monomial-based). The *Dense/better* and *Medium* type are cut off at 39 variables prior to reduction, which limits the runtime.

PBF's size increases when going from left to right. While the runtime of both implementations increases, we can see almost no differences in the percentiles $q \in \{0.5, 0.8, 1.0\}$, although the number of iterations increases with decreasing percentile q. Furthermore, the LSR algorithm achieves a runtime even better than the simple method of the monomial-based implementation. Recall that the simple method uses $T_{f_{t-1}}$ steps in search for the next variable pair (*i.e.*, step (a)) and $\sum_{m \in f_{t-1}} 2|m|$ steps for the replacement (*i.e.*, part (b)). Our proposed algorithm outperforms this simple approach. Furthermore, $q = 1.0$ and the *Dense/better* selection type are comparable in terms of selection strategy—except for lexicographic sorting—in each iteration. Thus, their comparison is the decisive one, when it comes to speedup. At 39 variables in the input function, the *Dense/better* selection type already needed more than a day to compute the quadratised function ($d_4(f) = 1$). We therefore did not compute bigger functions for that type. On the other hand, the new algorithm needs ≈ 10 seconds for the same input polynomial.

When considering the number of terms T_{f_0} in the input function f_0, Figure 5 shows their influence on the runtime (y-axis; log scale) for the same input PBFs as in Figure 4. Take into consideration that the *Dense/better* and *Medium* variant for the monomial-based algorithm are also cut off at 39 variables. Thus, the number of terms is limited, although it increases with increasing density. As before, we see the runtime benefit of the new algorithm.

Apart from runtime, structural properties of quadratised PBFs are interesting in regard of their influence on further steps in a toolchain from a higher-level problem description to executing the problem on quantum hardware. One aspect is the degree-1 and degree-2 density, which directly translate to QUBO densities and therefore give insights on interactions in a quantum circuit, when, for example, using QAOA. Assuming sufficiently many iterations in the algorithm, the degree-1 density tends towards its maximum value, that is, $d_1 \to 1$[57]. Figure 6 compares the degree-2 density (y-axis) for the introduced LSR algorithm (left) and the monomial-based algorithm (right) for different problem sizes (x-axis) and coloured by different PBF-densities (function density). Interestingly the monomial-based search variants do not differ visually, although they implement

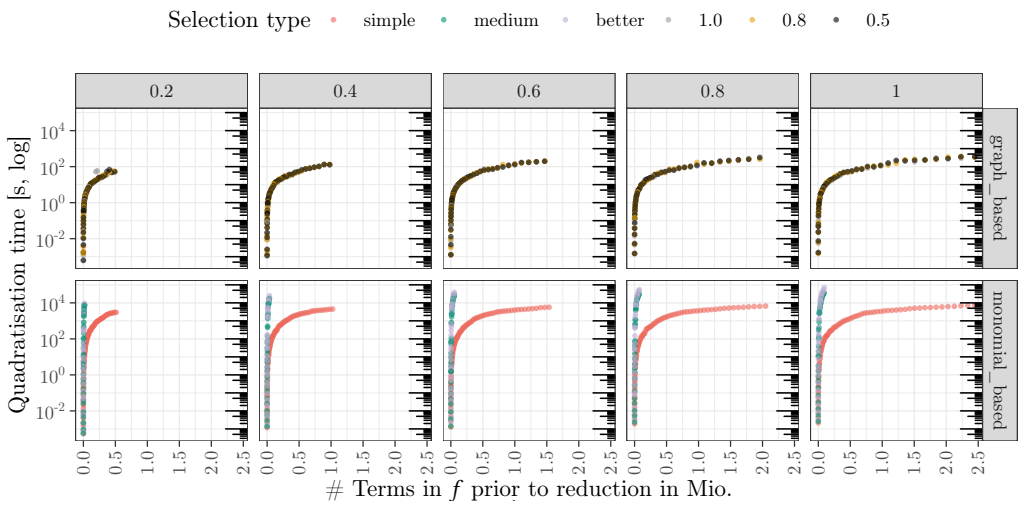


Fig. 5. Time in seconds (y-axis) for the *quadratisation* of a $\deg(f) = 4$ function f vs problem size (x-axis: Number of Terms). New LSR algorithm (top row) compared to existing monomial-based (bottom row). Vertical facets: Different base polynomial densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f) \in \{0.2, 0.4, \ldots, 1.0\}$). Selection type 1.0 (LSR algorithm) comparable to *Dense/better* (monomial-based). The *Dense/better* and *Medium* type are cut off at 39 variables prior to reduction, which limits the runtime.

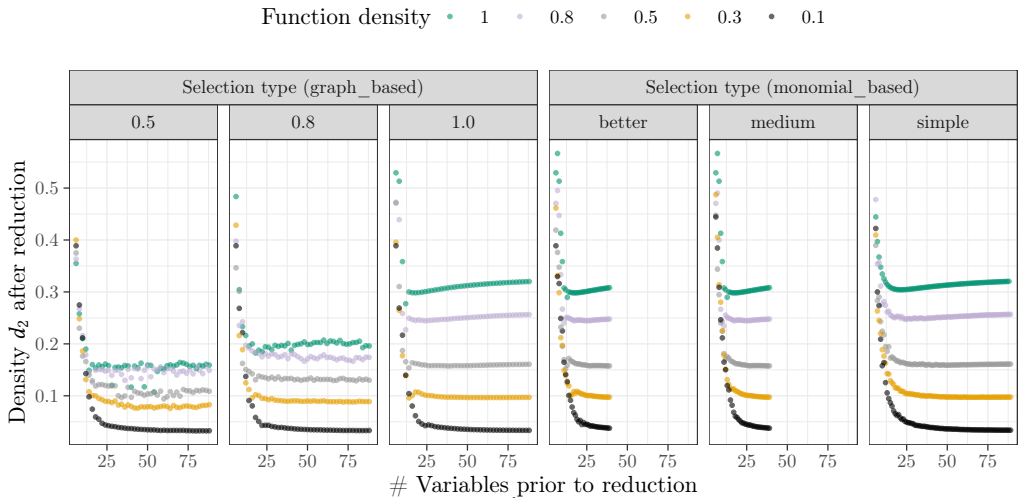


Fig. 6. Degree-2 density (y-axis) for the *quadratisation* of a $\deg(f) = 4$ function f vs problem size (x-axis: Number of variables) for different base polynomial densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f) \in \{0.1, 0.3, 0.5, 0.8, 1.0\}$). New algorithm on the left compared to monomial-based on the right. Selection type 1.0 (new algorithm) comparable to *Dense/better* (monomial-based). The *Dense/better* and *Medium* type are cut off at 39 variables prior to reduction, due to time limitations.

different search variants. As before, the *Dense/better* variant compares to $q = 1.0$, which also shows in the density plot for up to 39 variables, where the *Dense/better* variant is cut off due to runtime limitations. Lowering the percentile q lowers the degree-2 density in the quadratised function—although, we see a greater difference when varying the function density of the input PBF. Take

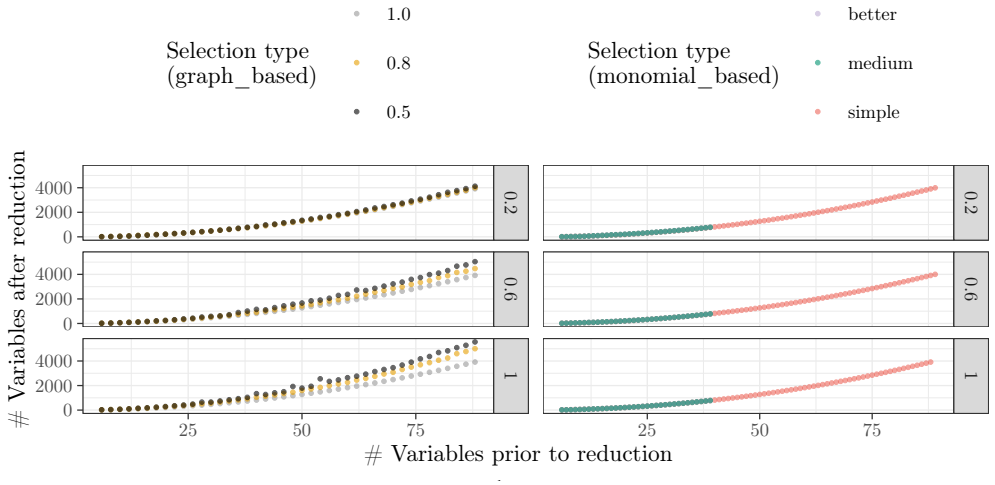


Fig. 7. Number of variables after *quadratisation* (y-axis) for the reduction of a $\deg(f) = 4$ function f vs problem size (x-axis: Number of variables). New algorithm (left column) compared to monomial-based (right column). Vertical facets: Different base polynomial densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f) \in \{0.2, 0.6, 1.0\}$). Selection type 1.0 (new algorithm) comparable to *Dense/better* (monomial-based). The *Dense/better* and *Medium* type are cut off at 39 variables prior to reduction, due to time limitations. The number of variables after reduction (y-axis) minus the number of variables prior to reduction (x-axis) depict the number of iterations in the LSR algorithm and therefore the number of additional variables.

into consideration that we generate the input function randomly, which encourages uniformly distributed variable pairs among monomials. Thus, exploiting problem inherent structures in the input function is not possible, which is contrary to our previous work [57], where we analyse the effect of monomial-based reduction in the context of a Job-Shop Scheduling problem.

The number of variables after *quadratisation* of $f : \{0,1\}^n \mapsto \mathbb{R}$ translates to the number of qubits in a quantum circuit, when, for example, using QAOA. Figure 7 compares the number of variables prior to reduction (*i.e.*, n: problem size) to the number of variables after reduction (*i.e.*, $n + I_f$; y-axis), where I_f corresponds to the number of reduction iterations. Figure 7 differentiates between the new graph_based (left) and the monomial-based algorithm (right), as well as different horizontally faceted input function densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f)$). Recall that the degree-2 density depends on the selection type, which is connected to the number of variables each variant introduces. While Figure 7 shows no difference between different monomial_based variants, the graph_based variant manages to extend I_f depending on selection type. Since the graph_based variant implements a variety of selection types (via percentiles of multiplicities; see Table 2), interpolation in an automated process is possible, which then influences properties of quantum circuits generated from the resulting function (*i.e.*, # qubits, gate distribution, circuit depth, and runtime). Consider that the baseline (*i.e.*, 1.0 and *better*) overlaps predominately in both the graph_based and monomial_based reduction algorithms.

We show how the number of terms in a function $f : \{0,1\}^n \mapsto \mathbb{R}$, where f has all possible terms up to a specified degree, scales with the number of variables. Recall that there are, $\binom{n}{k} \in O(n^k)$ possible degree-k monomials in f. Hence, when we see k as a constant and scale the number of variables n, the number of terms is bounded by a polynomial for a sufficiently large value of n. However, when k reaches n or in other words, when f has all possible monomials, their number scales exponentially with the number of variables in f, which is visualised in Figure 8. Therefore, as the number of

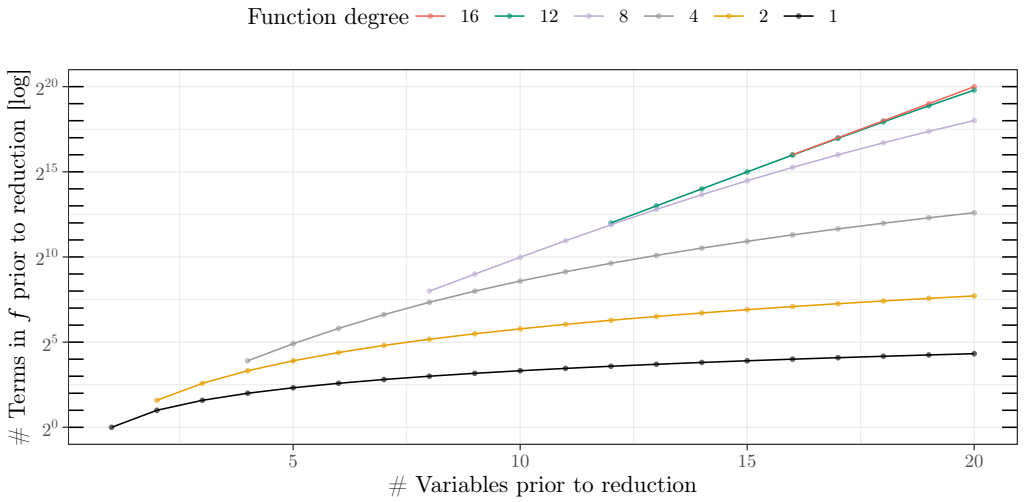


Fig. 8. Number of variables (x-axis) vs number of terms (y-axis) prior to reduction for functions f such that $deg(f)$ = Function degree and densities $d_1(f) = d_2(f) = \cdots = d_{\deg(f)}(f) = 1$.

variables (typically the problem size) grows in practically relevant problems, problem formulations tend towards sparse PBFs, since exponential space requirements to represent that function are infeasible. Thus, for larger problem sizes, this leads to less edges in the graph representation and potentially less connected nodes—further reducing the runtime of a reduction step.

6 Industrial Utility

6.1 SAT and its Connection to Quantum Computing

SAT is the seminal NP-complete problem that influenced and shaped major parts of complexity theory [15, 34]. SAT can naturally express logically constrained problems, and is relevant in many applications: This includes interdisciplinary verification problems (*e.g.*, checking against a formal specification, as in electronic circuits or software development) or scheduling problems (*e.g.*, in manufacturing or timetable planning) that are highly relevant industrial problems [8, 42]. For the following analysis, we consider SAT instances that arise from industrially inspired problems.

Recall that a SAT formula in **Conjunctive Normal Form (CNF)** consists of m clauses C_i that are conjoined by logical *and* ($\wedge$) operations:

$$\psi(\vec{x}) = \bigwedge_{i=1}^{m} C_i, \ \vec{x} \in \{T, F\}^n,$$

where each clause consists of literals l_i (*i.e.*, possibly negated variables x_i) that are conjoined by logical *or* ($\vee$) operations. For instance, $C_1 = (x_1 \vee x_2 \vee \overline{x_3})$ is a clause that consist of variables x_1, x_2 and x_3 and literals x_1, x_2 and $\overline{x}_3$ ($\overline{x}_3$ denotes the negation of x_3). For any given input $\vec{x} \in \{T, F\}^n$ of a SAT formula $\psi(\vec{x})$, $\psi(\vec{x})$ evaluates to either true (T) or false (F) (we substitute 1 for T and 0 for F in the following). SAT is the task of deciding whether or not there exists an input $\vec{x}$ such that $\psi(\vec{x})$ is true. $|C|$ denotes the number of literals or variables in a clause C.[19] We call $\psi(\vec{x})$ a k-SAT instance if k is the maximum number of literals in a clause (*i.e.*, $k = \max_{i \in \{1,\ldots,m\}} |C_i|$). If each clause in a

[19]Note that the number of literals in a clause is equal to the number of variables in the same clause. However, the number of literals for a given formula might be (and usually is) larger than the number of variables.

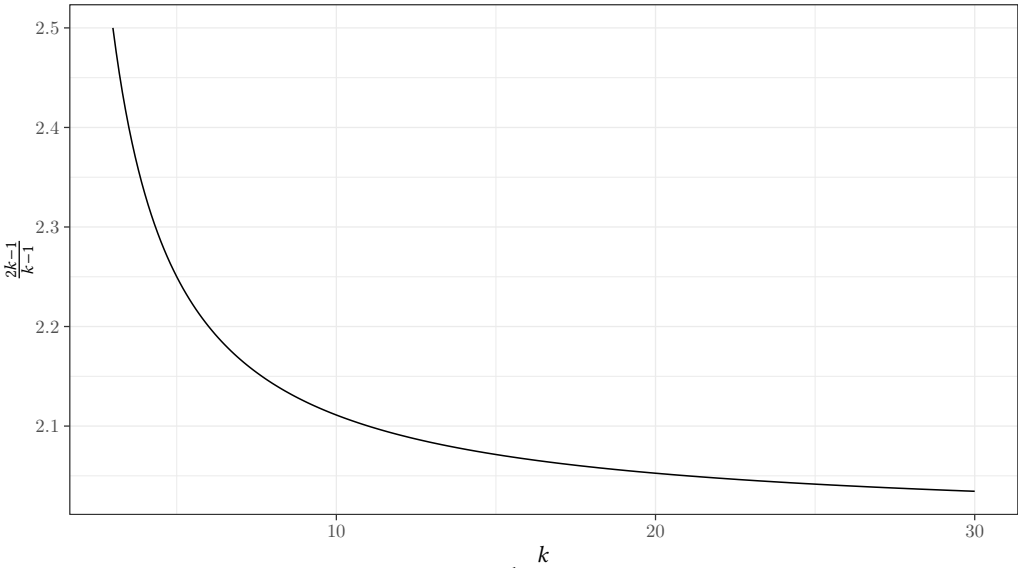


Fig. 9. Location of satisfiable to non-satisfiable phase transition for power law distributed k-SAT instances for varying k and small enough ratios $\frac{m}{n}$.

SAT instance $\psi(\vec{x})$ has the same number of literals k (*i.e.*, $|C_i| = k, i \in \{1, \ldots, m\}$), we call $\psi(\vec{x})$ an exact k-SAT instance.

The structural properties of SAT formulae are known to strongly influence the performance of SAT solvers. For instance, there are uniformly random generated SAT formulas, where exact k-SAT clauses are randomly populated with literals—each with identical probability.[20] By fine tuning the number of clauses m and variables n, the generation process delivers instances with a well-known phase transition for $k = 3$ at $\alpha = \frac{m}{n} \approx \alpha_C = 4.267$ from satisfiable to non-satisfiable instances [48]. While such instances are of fundamental interest [23], practical (industry-relevant) problems usually exhibit substantially different structures. This gives rise to targeted SAT solvers [2, 47, 62]. Since our proposed LSR method is not independent of the input structure, it is interesting what characteristics arise from different input structures, particularly ones close to industry-relevant problems. Ansótegui et al. [2] propose several methods to generate industrial-like SAT instances. In particular, they uses a power law distribution for variables in clauses; the resulting instances come close to industrial SAT instances, as Friedrich et al. show [21, 22]. Similar to uniform random SAT instances, they also show a phase transition, whose location, however, not only depends on the ratio of clauses to variables, but also on the power law distribution that variables are sampled from—defined by β_S. In particular, a phase transition from satisfiable to non-satisfiable instances occurs at $\beta_S = \frac{2k-1}{k-1}$ for a small enough constant $\alpha = \frac{m}{n}$ [21]. Figure 9 shows that the location of the phase transition assumes smaller values for higher k. In particular, it converges to 2 for high values of k: $\lim_{k\to\infty} \frac{2k-1}{k-1} = 2$. For example, fixing $\beta_S = 2.5$ and increasing $k > 3$ lets instances tend towards satisfiable instances, provided a small enough clause to variable ratio. Take into consideration that this cannot be stated for SAT instances within a certain region, that is, formed by higher clause to variable ratios and $\beta_S > \frac{2k-1}{k-1}$, since the location of a (possible) phase transition is not known precisely.

Transforming SAT instances into representations compatible with quantum solvers involves many choices and parameters (*e.g.*, concrete transformation method, (intermediate) optimisation

[20] Clauses that contain a variable and its negation are trivially satisfiable and can be excluded.

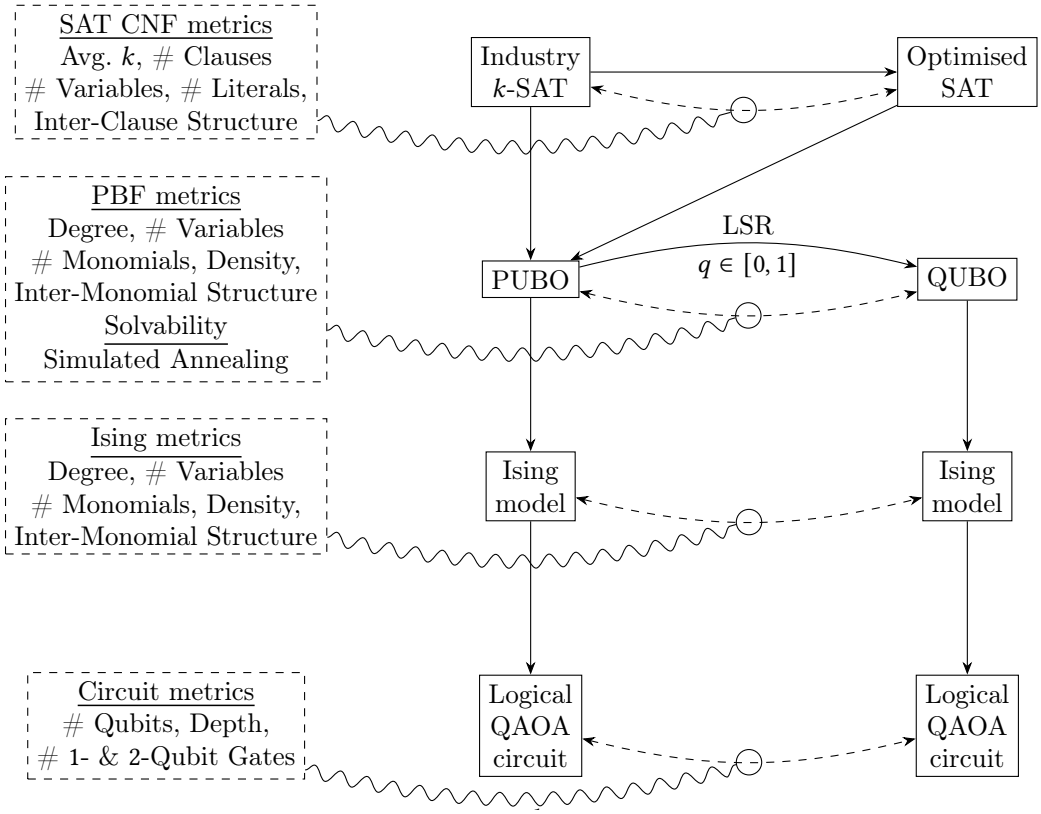


Fig. 10. Experiment setup: Starting at industrial like k-SAT formulas, their transformation (over an optimised SAT formulation) to PUBO, QUBO, the Ising model and logical quantum circuits for QAOA with special focus on properties at and between each respective horizontal representation (cf. Table 4). Solid arrows represent transformations (*e.g.*, from PUBO to QUBO); dashed arrows depict comparisons between two (similar) representations and are associated with specific properties (*e.g.*, metrics or solvers) as shown on the left.

steps, choice of (quantum) algorithm and its parameters, mapping and routing problem, and hardware platform) that influence properties of the resulting hardware executable representation. We zoomed into this pictured tree of transformation paths by setting the starting point at uniform random k-SAT instances and their transformation to QUBO in a previous work [56]. In contrast, the approach introduced in this article goes beyond uniform random k-SAT instances as well as the QUBO model by explicitly considering effects on the Ising model and QAOA circuits. We showed that the number of monomials in a particular k-SAT to PUBO transformation scales exponentially in the number of positive variables in a clause, which quickly prevails positive effects of this transformation path when increasing k. In this work, we introduce an additional prior optimisation step that prevents this unfavourable exponential scaling.

Figure 10 gives an overview of our experiments: Solid rectangles represent representations at different levels of abstraction—starting at k-SAT (top) over PUBO and QUBO (*i.e.*, PBFs), the Ising model and QAOA circuits (bottom). Solid arrows represent transformations (in-)between levels of abstraction. An industrial-like k-SAT instance, is either directly mapped to PUBO (referred to as *direct PUBO*) or first subjected to an optimisation step and then cast as PUBO (referred to as *optimised PUBO*) for comparison. These two PUBO formulations can then be used to either directly

transform into Ising models or to first transform to QUBO (referred to as *direct QUBO* or *optimised QUBO*) and then a mapping to Ising models. We then create logical QAOA circuits from each of the four Ising models. Logical quantum circuits are independent of the hardware platform and would need to undergo further transformations to be hardware executable (*e.g.*, mapping problem, qubit assignment problem, or transpilation to the hardware gate set). In Figure 10, we list important properties that are associated to the representation in each abstraction layer on the left. The main objective of the following experiments is to characterise how and to what extent they change when applying transformations in the same abstraction layer (*i.e.*, horizontal transformations) or applying transformations between abstraction layers (*i.e.*, vertical transformations). In the following sections, we first introduce each representation, necessary concepts and transformations and then characterise their respective influence—using the aforementioned k-SAT instances as input.

6.2 (Optimised) k-SAT to PBF

6.2.1 Optimised k-SAT. Recall that a PBF is a function $f : \{0,1\}^n \to \mathbb{R}$ that assumes a multilinear representation. Since the domain of f consists of all binary vectors with length n and is isomorphic to the domain of a SAT formula with n variables, it is natural to directly map variables from SAT to PUBO. By using De-Morgan's laws, it is possible to map clauses without introducing additional variables [19]. This is a local[21] benefit compared to other methods (*e.g.*, via the maximum independent set problem) that use additional variables [13, 56].

More technically, let $\psi(\vec{x}) = C_1 \wedge C_2 \wedge C_3$ be an exact 4-SAT instance, with

$$C_1 = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}), \qquad C_2 = (x_2 \vee x_3 \vee \overline{x_4} \vee x_5), \qquad C_3 = (x_2 \vee \overline{x_3} \vee x_4 \vee x_5).$$

While negative literals $\overline{x_i}$ can be directly mapped to variables x_i, positive literals x_i are mapped to terms $(1 - x_i)$, resulting in the following higher-order[22] PBFs of degree 4:

$$\begin{aligned} f_{C_1}(x_1, x_2, x_3, x_4) &= 1 - x_1x_2x_3x_4, \\ f_{C_2}(x_2, x_3, x_4, x_5) &= 1 - (1 - x_2)(1 - x_3)x_4(1 - x_5), \\ f_{C_3}(x_2, x_3, x_4, x_5) &= 1 - (1 - x_2)x_3(1 - x_4)(1 - x_5). \end{aligned}$$

The objective function encoding $\psi(\vec{x})$ is obtained by:

$$f(\vec{x}) = \sum_{m \in \{1,2,3\}} f_{C_m}. \tag{23}$$

$\psi(\vec{x})$ represents a maximisation problem, and the equivalent minimisation problem $-f(\vec{x})$, is easy to obtain. Let $C = (x_1 \vee x_2 \vee \ldots \vee x_k)$ be a k-SAT clause with $k \geq 2$ positive literals. Then, its corresponding degree-k PBF is given by $f_C(x_1, x_2, \ldots, x_k) = 1 - (1 - x_1)(1 - x_2)\ldots(1 - x_k)$. However, this leads to an exponential amount of monomials in f_C after term expansion. To be more precise, the total number of monomials for function f_C is given by:

$$T_{f_C} = 2^t - 1, \tag{24}$$

with t (for this example: $t = k$) being the number of positive literals in clause C. As mentioned in Section 5.4 (see also Figure 8), this is intractable for practical instances beyond a certain $k > k_\alpha$. Such clauses are formed naturally when variables are subject to an *at least one* relationship (*e.g.*, *at least* one machine in a factory has to fulfil conditions specified by the meaning of variables). Consider that the difference between positive and negative literals in clauses is not simply a matter of inverting the corresponding function, but a matter of shifting the minima (for maximisation) or maxima (for minimisation), which we illustrate in Table 3.

[21] In the sense of this isolated transformation—without considering effects of subsequent transformations.
[22] Note that this mapping automatically results in a quadratic, respectively, linear PBF for 2-SAT or 1-SAT, respectively.

Table 3. Effect of Literal Negation in SAT for $C_1 = (x_1 \vee x_2 \vee x_3)$, $C_2 = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$ and Corresponding Functions $f_{C_1}(x_1, x_2, x_3) = 1 - (1 - x_1)(1 - x_2)(1 - x_3)$ and $f_{C_2}(x_1, x_2, x_3) = 1 - x_1 x_2 x_3$

$x_1x_2x_3$	C_1	f_{C_1}	C_2	f_{C_2}
000	0	0	1	1
001	1	1	1	1
...	...	...	...	...
110	1	1	1	1
111	1	1	0	0

Hence, we desire an optimised input k-SAT instance with less positive variables. Therefore, we iteratively replace a positive literal in all clauses that it occurs in by a new negative literal and constrain it by two new clauses. More technically, let $\psi(\vec{x}) = C_1 \wedge C_2$ be an exact 5-SAT instance with

$$C_1 = (x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5),$$
$$C_2 = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee x_5).$$

Then, x_5 can be replace by a new negative literal $\overline{x_6}$:

$$\begin{aligned} C_1 &= (x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x_6}), \\ C_2 &= (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee x_4 \vee \overline{x_6}), \\ C_3 &= (x_5 \vee x_6), \\ C_4 &= (\overline{x_5} \vee \overline{x_6}), \end{aligned} \tag{25}$$

where C_3 and C_4 are constraint clauses that ensure $x_5 = \overline{x_6}$. These steps could be repeated until no more positive literals occur in C_1 or C_2.[23] However, introducing (and constraining) new variables creates new clauses and therefore introduces new monomials. To be more precise, constraint clauses C_3 and C_4 from (Equation (25)) are mapped to:

$$C_3 = (x_5 \vee x_6) \rightarrow f_{C_3}(x_5, x_6) = 1 - (1 - x_5)(1 - x_6) = x_5 + x_6 - x_5 x_6,$$
$$C_4 = (\overline{x_5} \vee \overline{x_6}) \rightarrow f_{C_4}(x_5, x_6) = 1 - x_5 x_6,$$

which in total leads to $f_{C_3}(x_5, x_6) + f_{C_4}(x_5, x_6) = 1 + x_5 + x_6 - 2x_5 x_6$ and therefore four monomials. Since the optimisation problem for $\psi(\vec{x})$ is obtained by summation (see Equation (23)), monomials that are introduced by the constraint clauses might be already present in other functions f_{C_i} and hence would not increase the total number of monomials. Since x_6 is a newly introduced variable that replaces all occurrences of positive literal x_5, monomials x_6 and $-2x_5x_6$ can only originate from the two constraint clauses. Monomial x_5 can occur in other functions f_{C_i}(*e.g.*, when $\overline{x_5}$ is the only negative literal in a clause C_i). We can now combine these observations with the exponential scaling behaviour of the simple mapping (see Equation (24)): Let C be a k-SAT clause with t positive literals:

$$C = (\overbrace{\ldots \vee}^{a} \mid \overbrace{\ldots \vee}^{t-a} \mid \underbrace{\ldots \vee \ldots}_{k-t \text{ negative}}), \tag{26}$$
$$\underbrace{\qquad\qquad}_{t \text{ positive}}$$

[23] As a side note, whenever a variable either is only present as positive or negative literals in a SAT formula, corresponding clauses can be trivially excluded (since they are trivially satisfiable), as it would be the case for x_4 and x_5 in clauses C_1 and C_2 (see Equation (25)).

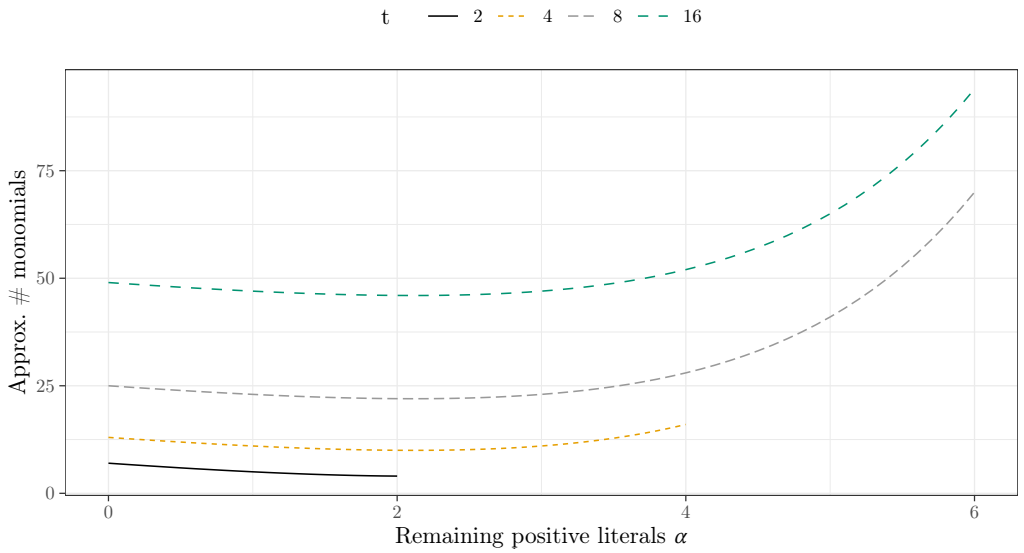


Fig. 11. Approximate number of monomials introduced by mapping a clause C (see Equation (26)). The x-axis shows the number of remaining positive literals after optimisation, while t is the number of positive literals prior to optimisation.

where $t - a$ are the number of positive literals that are replaced according to aforementioned optimisation method—leaving a positive literals in the optimised clause. The number of monomials introduced for clause C (including constraint terms) is approximately[24] given by $2^a + 3(t - a)$.

Figure 11 shows this function over a (x-axis) and different values of $t \in \{2, 4, 8, 16\}$. The minimum of each function ($t > 2$) is at $a \approx 2.11$. Since $2^2 + 3(t-2) < 2^3 + 3(t-3)$, leaving two positive literals results in approximately the least amount of monomials for an isolated clause. However, take into consideration that with every introduced additional variable, the (brute-force) search-space doubles in size. Hence, we choose to leave four positive literals, since the number of additional monomials is close to the optimum (see Figure 11). Also, the optimisation strategy replaces a positive literal in all clauses that it occurs in—further reducing the number of introduced variables.

For the following experiments, we use power-law distributed ($\beta = 2.5$) k-SAT instances, with $k \in \{3, 5, 7, 11\}$ and varying numbers of clauses (*i.e.*, $|C| \in \{13, 53, 107, 163, 263\}$). We also vary the number of possible variables $N \in \{7, 13, 23, 37, 47, 59, 71, 83, 101, 113\}$, while ensuring $N > k$. However, the actual number of variables n is lower than N, whenever variables are not chosen during the random clause sampling process. When applicable, we always show the actual number of variables n.

Figure 12 shows the effect of the optimisation strategy on three SAT metrics (*i.e.*, left: the number of variables, middle: the number of positive literals and right: the number of negative literals). Metrics for the optimised SAT instance are shown on the y-axis, while metrics for the instance prior to optimisation are shown on the x-axis. Furthermore, we differentiate between the number of clauses by colour and k by horizontal facets. From a birds eye view, we can see that the scaling behaviour follows a linear relationship for $k = 3$ with larger differences at higher k. This is due to the fact that we leave four positive variables per clause (see Figure 11) and thus 3-SAT instances are

[24]We deliberately leave out the constant monomial, as it is introduced by more than one function f_{C_i} for practically relevant problem instances.

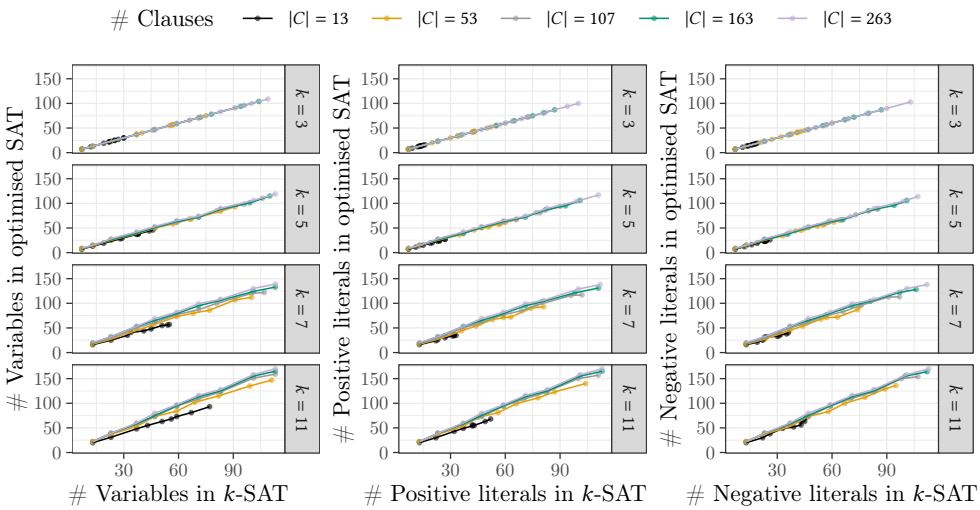


Fig. 12. Influence of optimisation strategy (see Equation (25)) on important SAT metrics (y-axis) based on SAT metrics prior to optimisation (x-axis). Number of Variables (left), number of positive literals (middle) and number of negative literals (right)—coloured by the number of clauses and horizontally faceted by k in the SAT instance prior to optimisation.

unaffected by this strategy. In contrast, for $k = 5$, only clauses that strictly contain positive literals are eligible for optimisation. Also, since we only replace positive literals, the number of additional variables is upper bounded by the number of positive literals, which is at maximum the number of variables. Figure 12 confirms the limited effect on the number of variables (and literals) for $k = 5$. For $k \in \{7, 11\}$, we see that instances with less clauses lead to less introduced variables (and literals). Furthermore, smaller number of variables have a similar effect. The effect of an optimisation step (*i.e.*, replacing a positive literal x_i by a new negative literal $\overline{x_j}$) can be generalised:

(1) The number of variables increases by one.
(2) Two additional 2-SAT clauses (*i.e.*, the constraint clauses) are added.
(3) The number of positive literals increases by one, since literals x_i and x_j only occur in one constraint clause.
(4) The number of negative literals increases by at least one ($\overline{x_j}$ in former clauses and in one constraint clause). If the negative literal $\overline{x_i}$ was not part of the SAT instance prior to optimisation, the number of negative literals (due to one constraint clause) additionally increases by one.

From (2), we can deduce that the average k for the optimised SAT instance depends on the number of optimisation steps (*i.e.*, the number of additional variables). Let $\psi(x_1, \ldots, x_n) = C_1 \wedge \ldots \wedge C_m$ be an exact k-SAT instance prior to optimisation. Then, after γ optimisation steps,

$$\text{Avg.}{-}k = \frac{1}{m+\gamma}\left(\sum_m |C_m| + 2\gamma\right).$$

Although the mentioned and analysed metrics (see Figure 11) are relevant to characterise the scaling behaviour for larger SAT instances, they do not give a detailed insight into the inner structure of an (optimised) SAT instance. Following our graph definition for PBFs (see Section 3), we now introduce a multi-graph representation for SAT formulas. Let $\psi(x_1, \ldots, x_n) = C_1 \wedge \ldots \wedge C_m$ be a k-SAT instance with m clauses and n variables. For the variable incidence graph $G_\psi(V_\psi, E_\psi)$, we define $V_\psi = \{x_1, \ldots, x_n\}$ as the set of variables of $\psi(\vec{x})$. In contrast to the graph representation

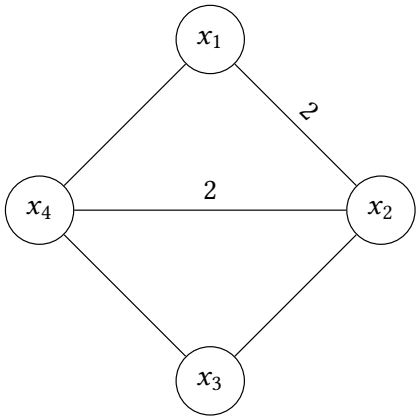


Fig. 13. Graph representation of $\psi(\vec{x}) = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$.

of PBFs, we no longer consider where edges stem from (*i.e.*, which monomials in the case of PBFs or which clauses in the case of SAT). If two variables x_i and x_j (can also be negated literals) both occur in a clause $C_i, i \in \{1, \ldots, m\}$, then edge $e = (x_i, x_j)$ is added to E_ψ. For example, let $\psi(x_1, x_2, x_3, x_4) = C_1 \wedge C_2 \wedge C_3$ with $C_1 = (x_1 \vee \overline{x_2})$, $C_2 = (x_2 \vee \overline{x_3} \vee \overline{x_4})$, and $C_3 = (\overline{x_1} \vee \overline{x_2} \vee x_4)$. Then for its corresponding graph $G_\psi(V_\psi, E_\psi)$, the set of nodes $V_\psi = \{x_1, x_2, x_3, x_4\}$ and the multi-set of edges

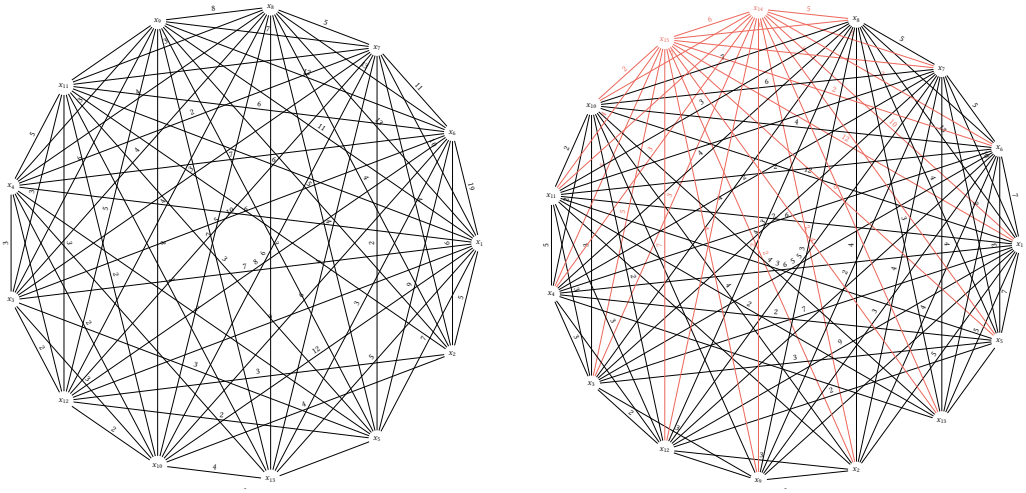
$$E_\psi = \big\{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_3, x_4), (x_1, x_2), (x_1, x_4), (x_2, x_4)\big\}.$$

Note that edges (x_1, x_2) and (x_2, x_4) occur two times in E_ψ (*i.e.*, have multiplicity 2). For simplicity and better visual representation, we show the multiplicity on edges as an edge label, instead of drawing multiple edges, which leads to the graph shown in Figure 13.

We now want to show the effect of the optimisation strategy on the inner structure of an exemplary power law distributed 5-SAT instance $\psi(\vec{x})$ with 13 variables and 37 clauses. Firstly, Figure 14(a) shows the graph representation of this instance. Secondly, Figure 14(b) shows the graph representation of the optimised SAT instance $\psi'(\vec{x})$, where new nodes and edges are coloured in red (see also the transformation paths in Figure 10). Note that an increase or decrease in multiplicity does not lead to a different colour on that edge. We can see that two new variables x_{14} and x_{15} are introduced that are both connected to all other variables via edges. Therefore, there are four new constraint clauses that introduce two edges with multiplicity 2, that are (x_{14}, x_j) and (x_{15}, x_i), where x_i and x_j are the positive literals that were replaced.[25] There are 6 connections between the two new variables and therefore there are exactly 6 clauses that contain these two variables. Recall that we choose to leave 4 positive variables per clause and hence there should be no connections between these two additional variables, since we started at a 5-SAT instance. However, positive literals are replaced in every clause they occur in (which further reduces the amount of monomials introduced in the PBF mapping). Hence, there must have been 6 clauses containing both positive literals x_i and x_j. We will use these two graph representations and their underlying SAT instances as the basis for the remaining transformation steps (see Figure 10) and figures down to a logical QAOA circuit.

In summary, the effects of the optimisation strategy on important metrics for SAT are predictable (analytically or quantitatively), which is an important factor when integrating this strategy into

[25]There might be additional edges (x_{14}, x_j) or (x_{15}, x_i) that increase the multiplicity between these nodes, since we do not replace negative literals.



(a) Graph G_ψ. (b) Graph $G_{\psi'}$ of optimised SAT.

Fig. 14. Inner structure of a power law distributed 5-SAT instance with 13 variables and 37 clauses (left) and inner structure of its optimised SAT formula (right). New nodes and edges are coloured in red.

Table 4. Naming Convention for Transformation Paths (see Figure 10)

Transformation path	Description
direct PUBO	PUBO from a non-optimised SAT instance.
optimised PUBO	PUBO from an optimised SAT instance.
direct QUBO	QUBO from a PUBO from a non-optimised SAT instance.
optimised QUBO	QUBO from a PUBO from an optimised SAT instance.

(automatic) toolchains. However, the question remains to what extent these SAT instances affect properties of following PBFs. Since we also want to consider the effects of a PUBO to QUBO transformation (see Figure 10), we refer to each path of transformation as in Table 4.

6.2.2 PUBO and QUBO. For the PUBO to QUBO transformation via the LSR method, we use percentile $q = 1$ (see Algorithm 2) to introduce less monomials. The following figures show the number of actually used variables of the non-optimised SAT instance on the x-axis. Figures 15 and 16 show the number of variables and the number of monomials (y-axis) in PBF, respectively. As in Figure 12, they are horizontally faceted by k. Conversely, vertical facets show the the number of clauses in the non-optimised SAT instance and colour encodes the specific path of transformation. Recall that the transformation from SAT to PBF does not introduce additional variables. Therefore, the difference between *direct PUBO* and *optimised PUBO* in Figure 15 shows the number of introduced variables by the optimisation strategy. Also, in Figure 15, *direct PUBO* is a linear relation and can therefore be used as reference for the logarithmic y-axis. For Figure 15, we can clearly distinguish between PUBO and QUBO, with the latter using more variables. Also, as k increases, the difference becomes larger, which is to be expected, since higher k leads to higher-degree monomials in PBF. With higher-degree monomials, more reduction steps are necessary and therefore more variables are introduced. However, this trend can break if specific structures (*e.g.*, if higher-degree monomials share many variable pairs) can be leveraged by the LSR method. With higher number of clauses, the difference between the number of variables in QUBO an PUBO becomes larger—meaning more

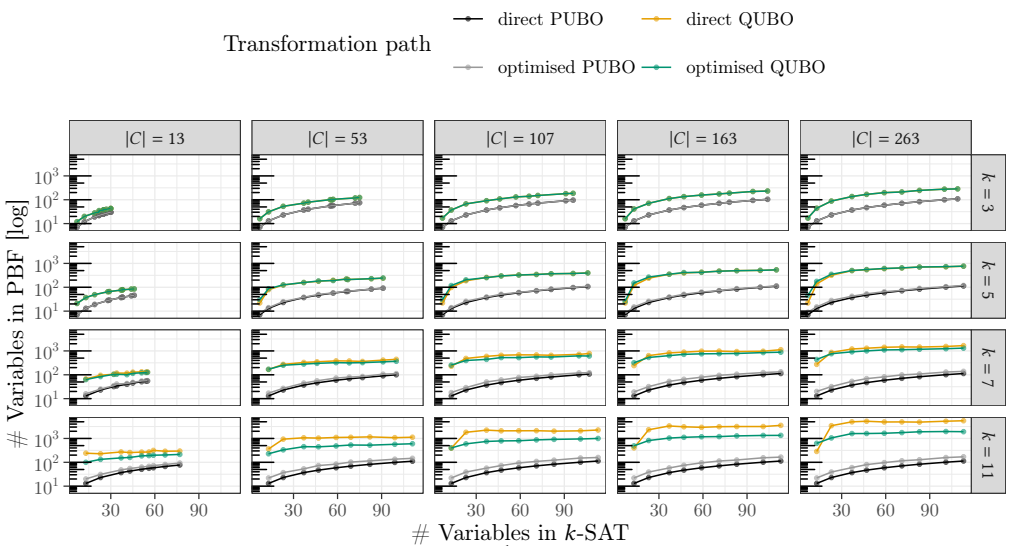


Fig. 15. Number of variables in PBF (y-axis) vs number of variables in non-optimised k-SAT (x-axis) horizontally faceted by k and vertically faceted by the number of clauses. Colour represents the concrete transformation path (*i.e.*, from non-optimised k-SAT: d. and from optimised k-SAT: opt.; see Table 4). The difference between *direct PUBO* and *direct QUBO* depicts the number of iterations in the LSR algorithm and therefore the number of additional variables in QUBO. The same applies when comparing *optimised PUBO* and *optimised QUBO*.

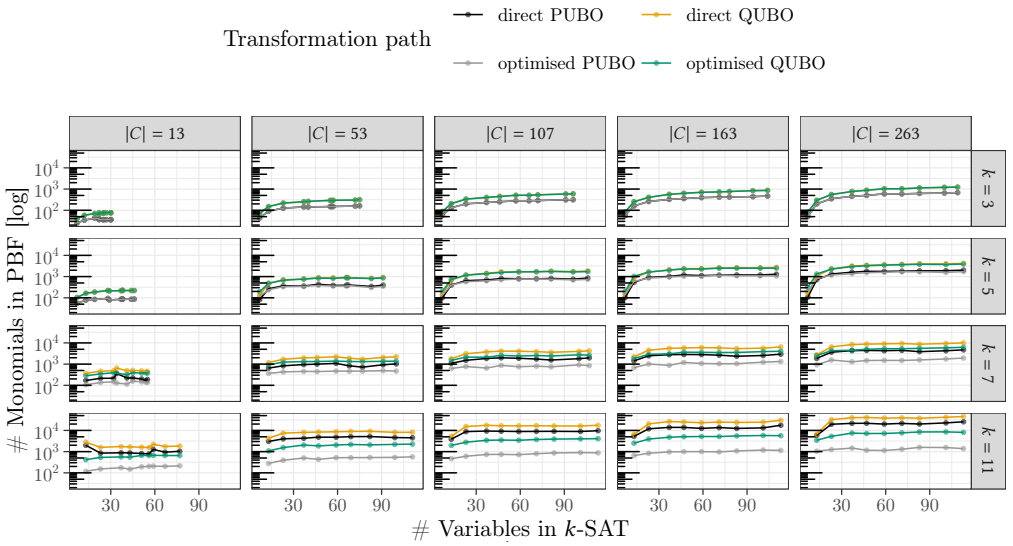


Fig. 16. Number of monomials in PBF (y-axis) vs number of variables in non-optimised k-SAT (x-axis) horizontally faceted by k and vertically faceted by the number of clauses. Colour represents the concrete transformation path (*i.e.*, from non-optimised k-SAT: d. and from optimised k-SAT: opt.; see Table 4).

variables are introduced. Interestingly, both QUBO formulations tend towards a relatively stable number of variables, when increasing the number of variables in SAT (and PUBO). We presume that for fixed number of clauses $|C|$ and fixed k, the number of variables in QUBO (after transformation from PUBO) is relatively stable for varying clause to variables ratios $\frac{m}{n} > 1$. Another interesting

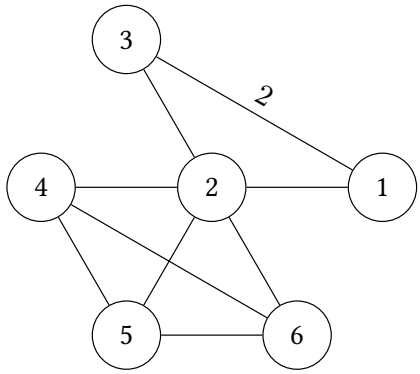


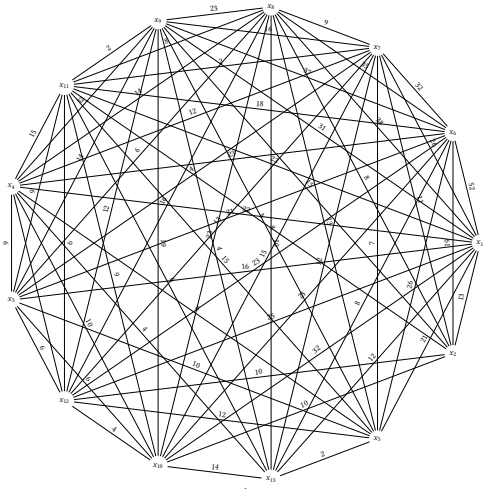
Fig. 17. Multi-graph G^f for $f(x_1, \ldots, x_6) = \pi x_1 x_2 x_3 - 13 x_2 x_4 x_5 x_6 + 7 x_1 x_3$.

aspect is the difference in both QUBO models for $k = 11$. Clearly, the SAT optimisation strategy is beneficial for the number of variables in QUBO—even for smaller k, the optimisation strategy does not increase the number of variables in QUBO, which is contrary to the PUBO formulation.

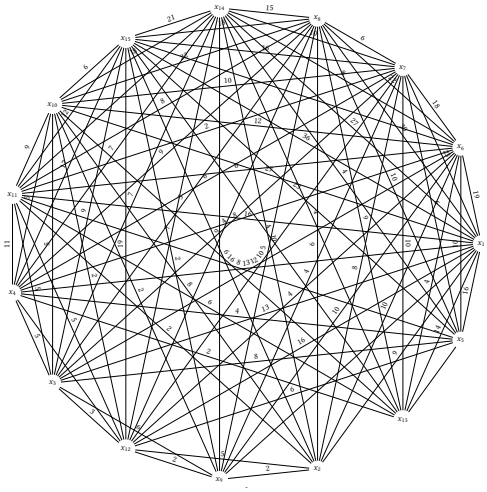
Apart from the number of variables, the number of monomials in PBFs majorly impact the performance of (quantum) solvers. For example, in Simulated Annealing, the objective function needs to be evaluated for every variable flip, which mostly governs the computational effort per iteration. Indubitably, finding a solution with Simulated Annealing depends on the structure and size of the search space, which (without further reduction) grows exponentially with increasing number of variables. Analogously to Figures 15, 16 shows the number of monomials on its y-axis. Note that for $k = 3$ the optimisation strategy has no effect and thus *direct PUBO* and *optimised PUBO*, as well as *direct QUBO* and *optimised QUBO* do not show differences. In contrast to Figure 15, it is evident that with increasing k, *optimised QUBO* outperforms *direct PUBO*, albeit *optimised PUBO* having the lowest number of monomials. To put this into perspective, the PUBO model can encode information into monomials ranging up to degree-k, while the QUBO model is restricted to degree-0, degree-1, and degree-2 monomials. Since we count the number of monomials, this highlights the impact of the SAT optimisation strategy and shows that upstream optimisation can have major impact on representations in lower layers. Analogously to Figure 15, the number of monomials in PUBO and QUBO are relatively stable with increasing number of variables in SAT, which means that (a) the optimisation strategy does not introduce specific structure that would significantly hinder or benefit the LSR method and (b) indicates why the number of variables for QUBO in Figure 15 is also relatively stable.

Apart from the degree, the number of variables and the number of monomials, we also want to characterise the inner structure of resulting PBFs. Hence, we define a similar multi-graph structure, as for SAT (see Figure 13). Let $f(x_1, \ldots, x_n)$ be a degree-k PBF with m monomials. We define the set of vertices V^f in its graph $G^f(V^f, E^f)$ as the set of variables from f ($V^f = \{x_1, \ldots, x_n\}$). Similarly, an edge $e = (x_i, x_j)$ is added to the multi-set of edges E^f, if both x_i and x_j occur in the same monomial. Note that this definition differs from the graph definition in Section 3 such that we no longer consider to which monomial an edge belongs to. As with the graph representation for SAT, we do not show multiple edges between two nodes, but rather denote the number of edges between two nodes by an edge label. For example, let $f(x_1, \ldots, x_6) = \pi x_1 x_2 x_3 - 13 x_2 x_4 x_5 x_6 + 7 x_1 x_3$ be defined as for Figure 1 and let $G^f(V^f, E^f)$ be its corresponding graph, which we show in Figure 17.

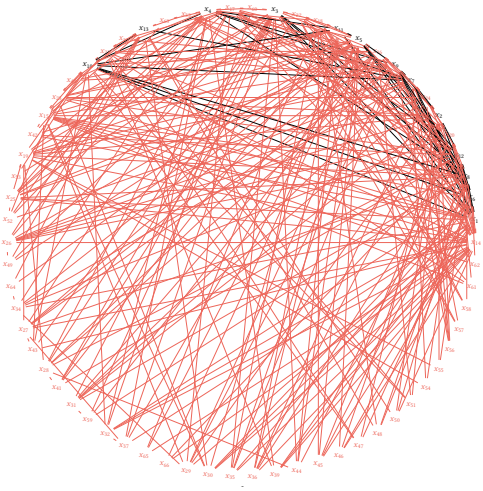
Following this definition, Figure 18 shows the graphs for *direct PUBO*, *optimised PUBO*, *direct QUBO*, and *optimised QUBO* for the same 5-SAT instance as in Figure 14 with the non-optimised SAT instance as basis on the left and the optimised instance on the right. As with the previous



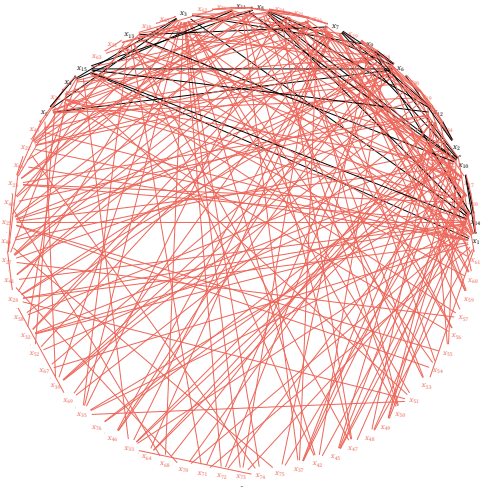
(a) Graph for *direct PUBO*.



(b) Graph for *optimised PUBO*.



(c) Graph for *direct QUBO*.



(d) Graph for *optimised QUBO*.

Fig. 18. Inner structure of PUBO (top) and QUBO (bottom) representations of a power law distributed 5-SAT instance with 13 variables and 37 clauses (left). Inner structure of its optimised SAT formula and derived PUBO (top) and QUBO (bottom) representations (right). New nodes and edges compared to the previous representation (see Figure 14 for PUBO models) are coloured in red. See Figure 10 and Table 4 for more information about the concrete transformation paths.

graphs, we colour new nodes and edges in red (changing multiplicities alone do not lead to the colour red). We can see that both PUBO representations do not introduce additional variables and do not connect previously unconnected node pairs compared to Figure 14. This is a result of the mapping procedure: Let $\psi(\vec{x}) = (x_1 \vee x_2 \vee x_3)$ be a 3-SAT instance, G_ψ its corresponding graph and let $f(x_1, x_2, x_3) = 1 - (1 - x_1)(1 - x_2)(1 - x_3) = x_1 + x_2 + x_3 - x_1x_2 - x_1x_3 - x_2x_3 + x_1x_2x_3$ be its corresponding degree-3 PBF and G^f its corresponding graph. G_ψ has exactly three edges, that connect all nodes—forming a 3-clique. Since f contains $x_1x_2x_3$ as a monomial, this monomial alone also leads to the same 3-clique in G^f. However, since (due to term expansion) lower-order

monomials of at least degree-2 are present in f, the multiplicities increase. This can also be seen in Figure 18(a) and 18(b), where a larger increase in multiplicities is an indicator of lower-order terms, which originate from term expansion of positive literals in the SAT instance. Therefore, we can see that in total, the increase in multiplicities is lower for *optimised PUBO* than for *direct PUBO* compared to Figure 14. Take into consideration that this effect is more pronounced for higher k. Figure 18(c) and 18(d) show the respective graphs for QUBO that originate from PUBO (top) via our introduced LSR algorithm ($q = 1$). We can see that, compared to the graphs for PUBO, many new edges and variables are introduced. For each new variable at least three new edges (stemming from the penalty term) are introduced. However, take into consideration that graphs G^f for quadratic PBFs cannot have multiple edges between nodes x_i and x_j, since there is only a single degree-k, $k \leq 2$ monomial containing both x_i and x_j (*i.e.*, $\alpha x_i x_j$). Hence, each edge directly visualises a degree-2 monomial. We can see a slightly higher concentration of edges among the former variables in the graph for *direct QUBO* than for *optimised QUBO*, although the differences are not clearly visible from this small 5-SAT instance. For higher k, larger number of variables and larger number of clauses, the graphs for QUBO (due to the direct visualisation of degree-2 monomials) will follow the observations of Figures 15 and 16—meaning that there are less edges and variables for *optimised QUBO*.[26]

6.2.3 Energy Landscape. To this point, we thoroughly analysed metrics and how they change when transforming SAT to PBF. What remains in this abstraction layer, is a characterisation of the energy landscape of resulting PBFs. For instance, Simulated Annealing is a classical heuristic to solve optimisation problems (encoded as PBFs) [35, 44, 49]. In essence, it is a probabilistic algorithm that iteratively proposes changes to the current solution and either trivially or probabilistically accepts the proposed change. More technically, let $f(x_1, \ldots, x_n)$ be a PUBO that encodes a minimisation problem. Then, Algorithm 3 shows the inner workings of a variant of Simulated Annealing, where function ACCEPT_ANYWAY($\Delta E, \kappa$) performs a random experiment to accept proposed changes that increase the energy of f (*i.e.*, $f(\vec{x}') > f(\vec{x})$; see [44]). The idea is to potentially escape local minima at the beginning of the algorithm with high probability (high temperature κ'). Note that the probability also depends on the change in energy (Algorithm 3: l. 17)—resulting in lower probability to accept the change if the energy change is larger. Also, the probability to accept a proposed change (Algorithm 3: l.17) decreases as the temperature decreases. Take into consideration that we deliberately trivially accept changes that do not change the energy (Algorithm 3: l. 6) to allow for exploration of flat energy landscapes. This is due to the unique properties of SAT to PUBO formulations that can tend towards (partially) flat energy landscapes. For example, let $\psi(x_1, x_2, x_3)$ be an exact 3-SAT formula with 7 out of 8 possible unique clauses (meaning no two clauses contain the same literals). For instance, let $C_{\text{missing}} = (\overline{x_1} \vee \overline{x_2} \vee x_3)$. Then, $\psi(\vec{x})$ has exactly one satisfying solution $\vec{x} = 110$. Its corresponding PBF f also has exactly one optimum at $\vec{x} = 110$. For any two bit vectors $\vec{x}, \vec{y} \in \{0,1\}^n \setminus 110$, $f(\vec{x}) = f(\vec{y})$. Hence, f has a flat energy landscape, except for $\vec{x} = 110$, which encodes the optimum.

Since the performance of Simulated Annealing depends on the structure of the energy landscape of a PBF, we use it to compare PUBO and QUBO formulations. This is also interesting in view of a recent study by Dobrynin et al. [19] who compare the energy landscapes of PUBO and QUBO for combinatorial optimisation problems. Recall that a *quadratisation* has to adhere to Equation (6), which ensures that (under the minimisation of newly introduced variables) each value of the original function is preserved. This is usually achieved by multiplying penalty terms (introduced through LSR) by a large positive factor. For the Simulated Annealing experiment, we choose Boros

[26]Note that we also count the number of degree-0 and degree-1 monomials in Figure 16. However, there is at maximum only one degree-0 monomial and the number of degree-1 monomials is at maximum the number of variables.

ALGORITHM 3: Basic steps for Simulated Annealing (minimisation).

Input: PBF $f(\vec{x})$, steps S, initial temperature κ
Output: Vector $\vec{x}^*$

$\vec{x} \leftarrow$ CHOOSE_RANDOM_ELEMENT$(\{0,1\}^n)$, $s \leftarrow 0$
while $s < S$ **do**
  $\kappa' \leftarrow -(s - S) \cdot \kappa$ ▷ Current temperature
  $i \leftarrow$ CHOOSE_RANDOM_ELEMENT$(\{1, \ldots, n\})$
  $\vec{x}' \leftarrow$ FLIP_BIT$(i, \vec{x})$
  if $f(\vec{x}') \leq f(\vec{x})$ **then**
    $\vec{x} \leftarrow \vec{x}'$ ▷ Trivially accept flip
  else
    if ACCEPT_ANYWAY$(f(\vec{x}') - f(\vec{x}), \kappa')$ **then**
      $\vec{x} \leftarrow \vec{x}'$ ▷ Accept based on random experiment
    end
  end
  $s \leftarrow s + 1$
end
return $\vec{x}$

Procedure ACCEPT_ANYWAY$(\Delta E, \kappa)$
  $p \leftarrow \min\{1, e^{-\Delta E/\kappa}\}$
  if CHOOSE_RANDOM_ELEMENT$([0,1]) < p$ **then**
    return *True*
  end
return *False*

penalty factor [9] for a PBF f in its multi-linear representation (see Equation (2)):

$$1 + 2 \cdot \sum_{S \subseteq \{1,\ldots,n\}} |\alpha_S|. \tag{27}$$

Note that this penalty factor differs in value for the non-optimised and optimised PUBOs. Our primary goal is to compare *direct PUBO* with *direct QUBO* and *optimised PUBO* with *optimised QUBO*. Since a *quadratisation* does not alter the meaning of previous variables $\vec{x}$ in PUBO $f(\vec{x})$, we can evaluate PUBO $f(\vec{x})$ with solutions of QUBO for a fair comparison. More precisely, solutions from *direct QUBO* can be evaluated in *direct PUBO* and solutions from *optimised QUBO* can be evaluated in *optimised PUBO*. If, for example, a penalty term is not satisfied, this leads to a large increase in energy for QUBO (see Equation (27)), but not necessarily for PUBO. For the Simulated Annealing experiments, we use a set $K = \{\kappa_i | \kappa_i \geq 1, i \in \mathbb{N}\}$ of initial temperatures to accommodate for different energy landscapes with a recursive sequence $(\kappa_i)_{i \in \mathbb{N}}$, such that $\kappa_1 = 1 + 2 \cdot \sum_{S \subseteq \{1,\ldots,n\}} |\alpha_S|$, as in Equation (27), and $\kappa_{i+1} = \frac{\kappa_i}{2}$. We then show 100 runs for the initial temperature that achieves the lowest energy in PUBO for either a linear or a quadratic number of steps in the number of actually used variables in k-SAT in Figure 19. Similar to previous Figures 15 and 16, in Figure 19, we show the number of actually used variables in k-SAT on the x-axis, use colour for the concrete transformation path (see Table 4) and show k as horizontal facets. In contrast, the y-axis shows the energy in PUBO and we use vertical facets to differentiate between a linear and quadratic amount of steps. Additionally, we only show instances with $|C| = 53$ clauses (prior to SAT optimisation). For $k = 3$ the energy for the optimised and non-optimised PUBO and QUBO are similar—except for random fluctuations—, since our optimisation strategy does not affect 3-SAT instances (see Figure 11). For higher k, the minimum of *optimised PUBO* is generally lower than for *direct PUBO*,

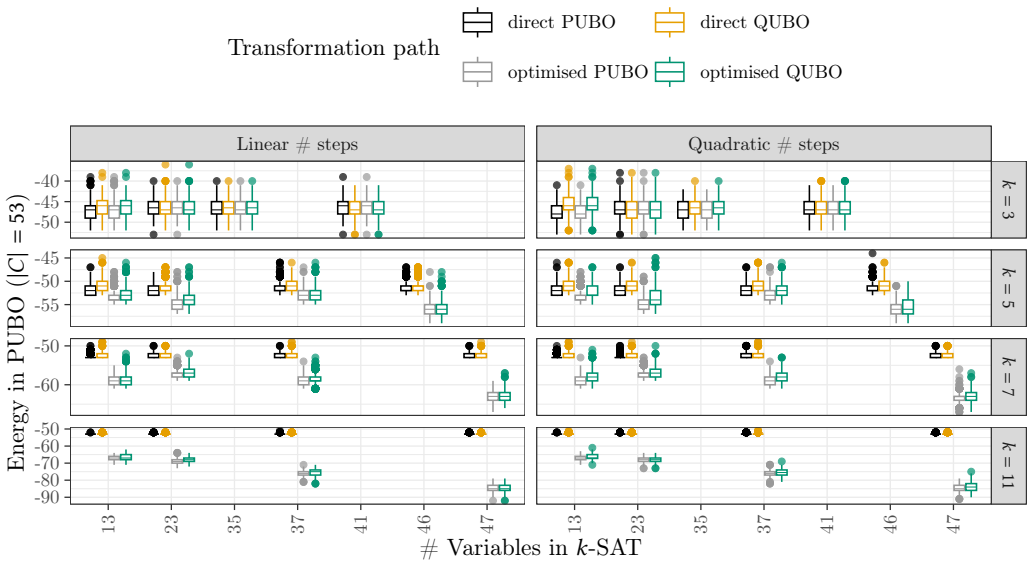


Fig. 19. Energy in PUBO (y-axis) vs number of variables in non-optimised k-SAT (x-axis) horizontally faceted by k and vertically faceted by the number of steps in Simulated Annealing (*i.e.*, either linear or quadratic in the y-axis). Each box-plot shows 100 runs of the best performing initial temperature for varying random initial assignments of variables. Colour represents the concrete transformation path (*i.e.*, from non-optimised k-SAT: d. and from optimised k-SAT: opt.; see Table 4). Since *optimised PUBO* results from a SAT instance with a higher number of clauses, its energy minimum is lower (except for $k = 3$) than for *direct PUBO*. Since both stem from the same non-optimised SAT instance, the difference in energy for *optimised PUBO* and *direct PUBO* has no meaning.

whenever the optimisation strategy introduces additional clauses. Hence, *direct PUBO* and *optimised PUBO* are not comparable in energy. When comparing *direct PUBO* with *direct QUBO* and *optimised PUBO* with *optimised QUBO*, we can see that overall PUBO performs (slightly) better. Take into consideration that QUBOs are at a disadvantage over PUBOs, due to an increased number of variables (see Figure 15), since the number of steps is fixed and Simulated Annealing considers a random variable flip per iteration. Since Simulated Annealing—in essence—heuristically traverses the node-weighted Hamming Graph of a PBF f, which not only depends on the structure of f, but also on the cooling schedule, we cannot conclude that QUBOs are inferior to PUBOs in Simulated Annealing from Figure 19.

6.3 PBF to Ising

Ising models and PBFs are similar representations with the key difference being their domain. Both models can be used to encode NP-hard optimisation problems [14, 40]. While PBFs are functions $f : \{0, 1\}^n \to \mathbb{R}$, Ising models are functions $f_I : \{-1, 1\}^n \to \mathbb{R}$. Hence, PBFs can be transformed to Ising models via the following relation between their variables [24]:

$$x_i = \frac{1 - s_i}{2},$$

where $x_i \in \{0, 1\}$ represent variables in the domain of PBFs and $s_i \in \{-1, 1\}$ represent variables in the domain of Ising models. This mapping leads to equivalent models such that the energy landscape is persevered with $x_i = 0 \iff s_i = 1$ and $x_i = 1 \iff s_i = -1$. For PBFs, $x^i = x$

($x \in \{0,1\}$, $i \in \mathbb{N}$) and similar for Ising models ($s \in \{-1,1\}$, $i \in \mathbb{N}$)[27]:

$$s^i = \begin{cases} 1, & \text{if } i \text{ even} \\ s, & \text{if } i \text{ odd,} \end{cases}$$

which allows for an analogous representation to multi-linear PBFs (see Equation (2)). However, the subtle difference in the domains of PBFs and Ising models has key implications for the metrics shown in Figure 10, when mapping from PBF to Ising. For instance, let $f(x_1,x_2,x_3) = x_1x_2x_3$ be a PBF. Then, its corresponding Ising

$$\begin{aligned} f_I(s_1,s_2,s_3) &= \frac{1-s_1}{2}\frac{1-s_2}{2}\frac{1-s_3}{2} \\ &= \frac{1}{8}\Big((1-s_1)(1-s_2)(1-s_3)\Big) \\ &= \frac{1-s_1-s_2-s_3+s_1s_2+s_1s_3+s_2s_3-s_1s_2s_3}{8} \end{aligned}, \tag{28}$$

contains all possible monomials up to degree-3. As illustrated in Figure 8, with degree-k functions f, the number of possible monomials up to degree-k grows exponentially in k, which points to a similar problem as in the non-optimised mapping from SAT to PBF (see Section 6.2). Hence, using lower-degree monomials in PBF f (*e.g.*, via *quadratisation*) can be beneficial in terms of the number of monomials in Ising f_I. However, when lower-degree monomials are present in f, they might be subsumed by higher-degree monomials. For example, when PBF $f(x_1,x_2,x_3) = x_1x_2x_3 + x_1x_2$, then Ising

$$\begin{aligned} f_I(s_1,s_2,s_3) &= \frac{1-s_1}{2}\frac{1-s_2}{2}\frac{1-s_3}{2} + \frac{1-s_1}{2}\frac{1-s_2}{2} \\ &= \frac{1}{8}\Big((1-s_1)(1-s_2)(1-s_3)\Big) + \frac{1}{4}\Big((1-s_1)(1-s_2)\Big) \\ &= \frac{1-s_1-s_2-s_3+s_1s_2+s_1s_3+s_2s_3-s_1s_2s_3}{8} + \frac{1-s_1-s_2+s_1s_2}{4}, \\ &= \frac{3-3s_1-3s_2-s_3+3s_1s_2+s_1s_3+s_2s_3-s_1s_2s_3}{8} \end{aligned} \tag{29}$$

has the same number of monomials as in Equation (28). Hence, if PBF f has all possible monomials for a subset of variables, the mapping to Ising does not introduce additional monomials for this subset of variables. However, as we illustrated in Figure 8, an exponential amount of monomials in PBF is infeasible to represent when scaling to larger problems and therefore PBFs tend towards sparse formulations. Therefore, the question arises how this transformation to Ising affects the QUBO and PUBO models.

Figure 20 shows the number of monomials in the Ising models (that originate from the respective PBF; see Section 6.2) on its y-axis for all four transformation paths (*i.e.*, *direct PUBO*, *optimised PUBO*, *direct QUBO*, and *optimised QUBO*). For the following observations, take into consideration that both Ising models that originate from PUBO, feature higher-order monomials (*i.e.*, of degree-k), while Ising models originating from QUBO are at most quadratic, although both encode the same SAT problem. This aspect is not captured in the mere number of monomials which we show on the y-axis. While for $k = 3$ and $k = 5$ each path seems to perform equally well, for $k > 5$ both Ising models that originate from QUBO have significantly less monomials than those that originate from PUBO. However, recall that even for $k = 3$ in Figure 16, both QUBO models have significantly more monomials than their corresponding PUBO models. Upon closer inspection, for $k = 5$ in

[27] $s_1 s_2^4 = s_1 \cdot 1 = s_1$. Therefore, monomial size can potentially decrease in Ising models. However, this is not the case for the transformation from PBF, since the transformation does not multiply PBF monomials.

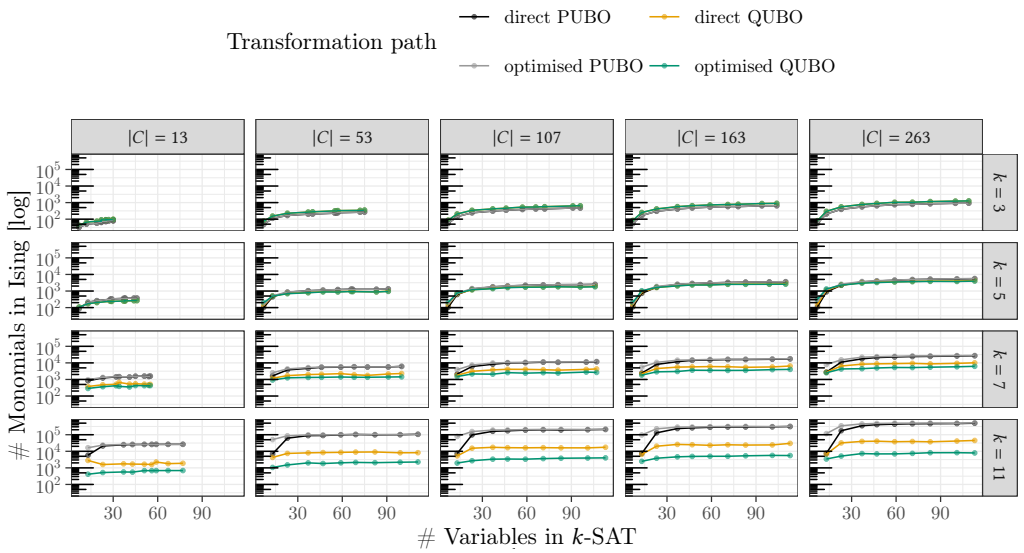
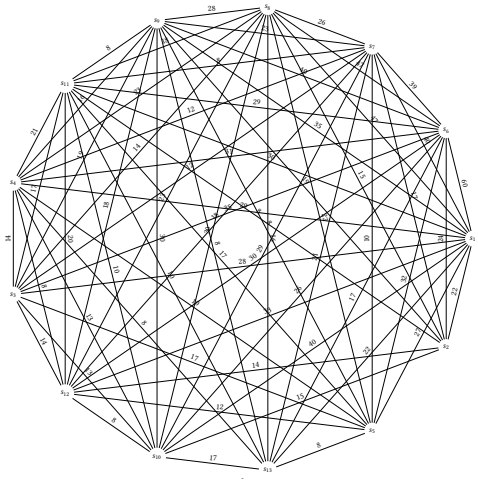


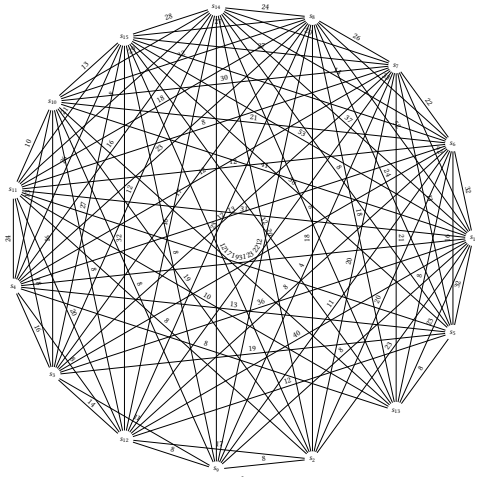
Fig. 20. Number of monomials in Ising model (y-axis) vs number of variables in non-optimised k-SAT (x-axis) horizontally faceted by k and vertically faceted by the number of clauses. Colour represents the concrete transformation path (*i.e.*, from non-optimised k-SAT: d. and from optimised k-SAT: opt.; see Table 4).

Figure 20, Ising models stemming from QUBO already outperform those stemming from PUBO. For $k = 11$ the difference is eminent with ≈ 66 times less monomials for QUBO in the case of $|C| = 263$ clauses, more than 101 variables and path *optimised QUBO* vs *optimised PUBO*. We can also see that reducing the number of monomials in PUBO (see Figure 16) via previous optimisation of the SAT instances also lowers the number of monomials in Ising. We do not explicitly show the number of variables in Ising, since they are exactly the same as in Figure 15 (*i.e.*, the *quadratisation* introduces new variables). However, the number of variables for $k = 11$, $|C| = 263$ and paths *optimised QUBO* vs *optimised PUBO* increases by a factor of ≈ 11 for QUBO (vs factor ≈ 66 for monomials in Figure 20). Take into consideration that these effects are more pronounced with higher k — except for (specifically generated) instances with inner structures that can be exploited by these transformations.

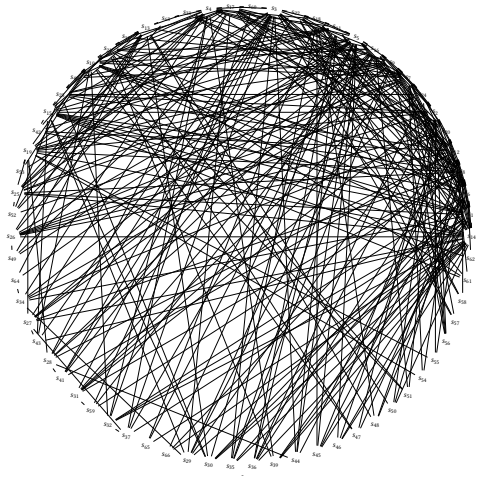
Since Ising models can also be represented as multi-linear polynomials (over a different domain compared to PBFs), we reuse the graph definition for PBFs and show the inner structure of Ising models in Figure 21. Since no edges or nodes in Figure 21 are coloured red, there are no new connections between nodes and no new nodes in the Ising model compared to their previous PUBO or QUBO model. For Ising models that originate from QUBO, we can generally state that no additional degree-2 monomials are introduced (see Equation (29) for an example), since there are no degree-k, $k > 2$ monomials in QUBO. Hence, their graphs are isomorphic. However, there are most certainly (new) degree-1 monomials in Ising that are not shown in these graphs. More generally, if a QUBO graph has no unconnected nodes, then its corresponding Ising model f_I (via the shown transformation) has all possible degree-1 monomials (or, equivalently, $d_1(f_I) = 1$). This is a result of the term expansion, shown in Equation (28): Every degree-2 monomial $\alpha x_i x_j$ in PBF generates two degree-1 monomials s_i and s_j in Ising (via the shown transformation). Although, for the higher-order Ising models that originate from PUBO (Figure 21(a) and 21(b)), no new connections are formed, the multiplicities increase, that stem from lower-order terms via term expansion. Take into consideration that, since we use a 5-SAT instance with $|C| = 37$ for these



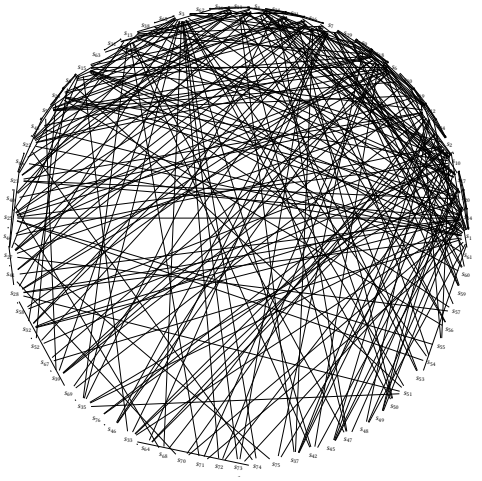
(a) Graph for Ising via *direct PUBO*.



(b) Graph for Ising via *optimised PUBO*.



(c) Graph for Ising via *direct QUBO*.



(d) Graph for Ising via *optimised QUBO*.

Fig. 21. Inner structure of Ising models stemming from PUBO (top) and from QUBO (bottom) that originate from a power law distributed 5-SAT instance with 13 variables and 37 clauses (left). Inner structure of its optimised SAT formula and derived Ising models stemming from PUBO (top) and from QUBO (bottom) on the right. New nodes and edges compared to the previous representation (see Figure 18 for PUBO models) are coloured in red. See Figure 10 and Table 4 for more information about the concrete transformation paths.

graphs (compare to Figure 20), the increase in multiplicity is rather limited. However, for higher k, the increase in multiplicities is going to follow the trend of Figure 20 with a potentially different factor that depends on the metric under consideration. For instance, the sum over all multiplicities will exceed the number of monomials for Ising models originating from PUBO, since edges in the graph represent pairs of variables that are contained in monomials.

6.4 Ising to QAOA

Ising models form the basis for quantum algorithms, such as QAOA, which can solve combinatorial optimisation problems [33]. Farhi et al. [20] originally proposed QAOA as a hybrid

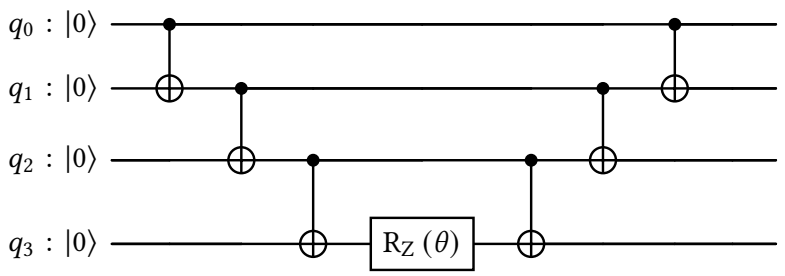


Fig. 22. Decomposition of $R_{ZZZZ}(\theta) = e^{-i\frac{\theta}{2}Z\otimes Z\otimes Z\otimes Z}$ into two-qubit CX and single qubit $R_Z(\theta)$ [10].

quantum-classical algorithm. Its quantum circuit has a layered structure, which consists of p layers—each composed of a problem specific unitary $U(H_C, \gamma_i) = e^{-i\gamma_i H_C}$ and a mixer $U(H_M, \beta_i) = e^{-i\beta_i H_M}$. Both $U(H_C, \gamma_i)$ and $U(H_M, \beta_i)$ are parametrised by γ_i and β_i that are subject to a classical optimiser (*e.g.*, a gradient based approach). The goal is to minimise the expectation value of the cost Hamiltonian H_C that encodes the (optimisation) problem by starting in an equal superposition and then applying $U(H_C, \gamma_i)$ and $U(H_M, \beta_i)$ p-times in alternation:

$$|\vec{\beta}, \vec{\gamma}\rangle = U(H_M, \beta_p)U(H_C, \gamma_p)\dots U(H_M, \beta_1)U(H_C, \gamma_1)H^{\otimes n}\,|0\rangle^{\otimes n},$$

where $H^{\otimes n}\,|0\rangle^{\otimes n}$ creates an equal superposition on all n qubits. H_C can be derived from an Ising model by replacing variables s_i with Pauli-Z operators acting on qubit i [51]. Note that, due to exponentiation of H_C, Pauli-Z operators become rotation gates in $U(H_C, \gamma_i)$. For instance, monomial αs_1 is mapped to a rotation gate $R_{Z_1}(\alpha\gamma_i)$ that acts on qubit one. Higher-order monomials in the Ising model (*e.g.*, $\alpha s_1 s_2 s_3$) lead to rotation gates acting on multiple qubits (*e.g.*, $R_{Z_1Z_2Z_3}(\alpha\gamma_i)$—acting on qubit one, two, and three). Since most currently available hardware natively supports up to two qubit interactions and usually does not directly support R_{ZZ} gates, we decompose higher-order rotation gates as it is shown in Figure 22 (although, in principle, it is possible to realise multi-qubit gates in trapped ion devices [59]). Also, when generating quantum circuits from Ising models, this is in favour of a fair comparison between higher-order and lower-order Ising models. The mixer $U(H_M, \beta_i)$ can be represented as $U(H_M, \beta_i) = \sum_j R_{X_j}(\beta_i)$, where R_{X_j} is a rotation around the x-axis acting on qubit j.

Recently, Montañez-Barrera and Michielsen [45] analysed a fixed linear-ramp schedule for QAOA for combinatorial optimisation problems. In contrast to iterative QAOA, the linear-ramp variant does not require a classical optimiser to adjust parameters γ_i and β_i in each layer of the quantum circuit and thus the quantum circuit only needs to be executed once (with a specific number of shots). The use of predetermined values for γ_i and β_i (*i.e.*, linear-ramp) requires that the objective function is normalised, which would otherwise be delegated to the classical optimiser.

We will use **Linear Ramp QAOA (LR-QAOA)** for the following experiments. Since the structure of circuits in LR-QAOA is equal to those in QAOA, the results—featuring circuit metrics—are also applicable to QAOA. Figure 23 shows the number of single-qubit gates in a quantum circuit for LR-QAOA with one layer ($p = 1$). This includes all gates introduced by $U(H_M, \beta_i)$, all Hadamard gates $H^{\otimes n}$ (n is the number of qubits) and all (decomposed) single-qubit gates from $U(H_C, \gamma_i)$. Since we decompose higher-order gates—leaving one single-qubit R_Z gate per monomial[28]—, the number of monomials in Ising (see Figure 20) closely depicts the number of single-qubit gates introduced by $U(H_C, \gamma_i)$. Moreover, since the number of single qubit gates introduced by $U(H_M, \beta_i)$ and $H^{\otimes n}$

[28]Except for the constant monomial.

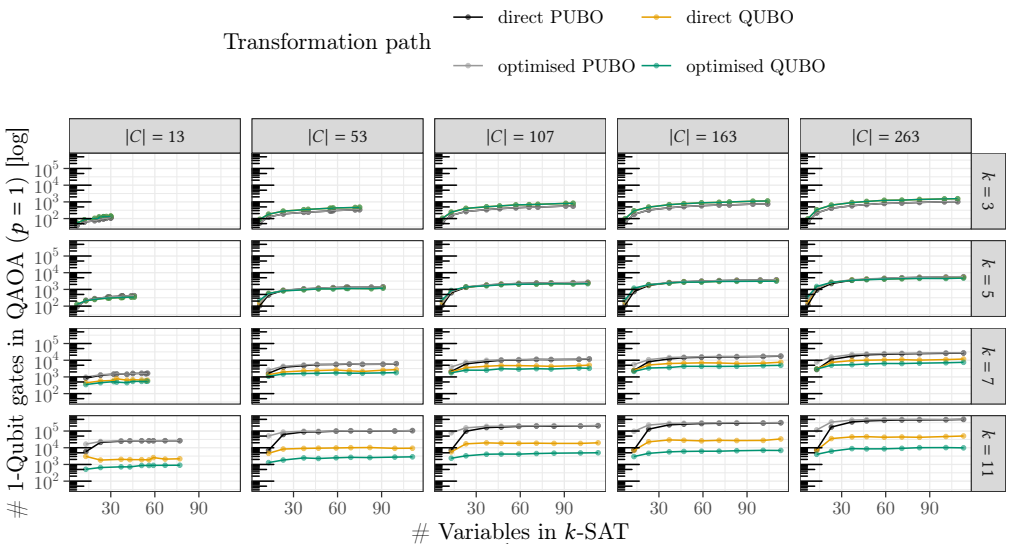


Fig. 23. Number of single-qubit gates in quantum circuit for LR-QAOA for $p = 1$ (y-axis) vs number of variables in non-optimised k-SAT (x-axis) horizontally faceted by k and vertically faceted by the number of clauses. Colour represents the concrete transformation path (*i.e.*, from non-optimised k-SAT: d. and from optimised k-SAT: opt.; see Table 4).

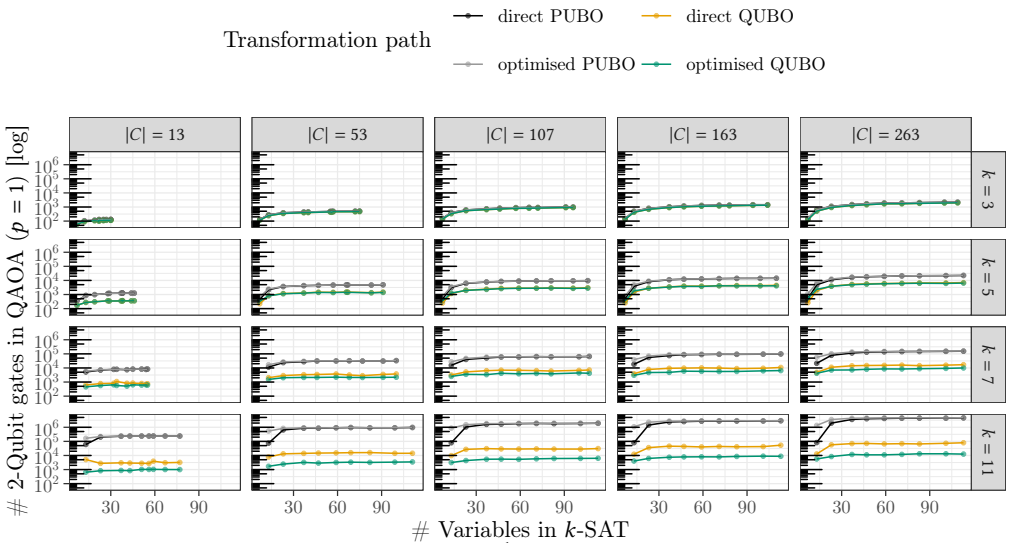


Fig. 24. Number of two-qubit gates in quantum circuit for LR-QAOA for $p = 1$ (y-axis) vs number of variables in non-optimised k-SAT (x-axis) horizontally faceted by k and vertically faceted by the number of clauses. Colour represents the concrete transformation path (*i.e.*, from non-optimised k-SAT: d. and from optimised k-SAT: opt.; see Table 4).

is exactly two times the number of qubits, Figure 23 closely depicts the sum of Figures 15 and 20—with analogous observations.

Apart from single-qubit gates, Figure 24 shows the number of two-qubit gates in the aforementioned quantum circuit for LR-QAOA (see Figure 23). In contrast to Figure 23, (decomposed)

gates only stem from (higher-order) monomials in the corresponding Ising model. Recall that the mere number of monomials in Ising (Figure 20) does not depict the size of monomials. Conversely, Figure 24 takes this into account due to the applied decomposition: Any degree-k monomial in Ising results in $2(k-1)$ two-qubit gates in the quantum circuit. Hence, the beneficial effects of QUBO models in terms of the number of gates are even more pronounced than in Ising (Note that Figures 20 and 24 have different y-axis scales). Comparing the paths for optimised QUBO and PUBO for $k = 11$, $|C| = 263$ and the maximum tested amount of variables in k-SAT, the path for QUBO copes with ≈ 383 times less gates in the quantum circuit, while requiring ≈ 11 times more variables. While for Ising models, these effects are more pronounced for higher k, they are especially pronounced for quantum circuits, due to the decomposition strategy (a more fair comparison). Moreover, quantum circuits for LR-QAOA that originate from quadratic Ising models can be depth optimised almost optimally (up to one additional layer) in polynomial time, due to Vizing's theorem [6]. More technically, a (at most) quadratic Ising model is mapped to a graph, for which a proper edge-colouring then determines the circuits depth. In higher-order Ising models, this mapping results in a hypergraph (*i.e.*, an edge can connect multiple nodes) for which (in general) it is difficult to find an optimal edge-colouring (see [29] for more information). Also, take into account that all tested models (via paths *direct PUBO*, *optimised PUBO*, *direct QUBO*, and *optimised QUBO*) encode the same initial k-SAT instances—highlighting the relevance of transformation paths and their study, due to vastly different properties of resulting quantum specific representations (especially when considering scaling behaviour).

7 Conclusion and Outlook

We introduce an optimised algorithm, subdivided into two stages, to compute a *quadratisation* for pseudo Boolean functions **(PBFs)** that play a major role in the formulation of combinatorial optimisation problems (**COPs**)—not limited to quantum computing. We give a thorough mathematical analysis of its inner workings and prove properties related to the underlying graph structure, which ultimately leads to the performance gain. Furthermore, we give complexity theoretic bounds on its performance and show empirically that the proposed algorithm outperforms the existing monomial-based algorithm. On top of that, the introduced algorithm is more versatile in terms of selecting specific characteristics of the quadratised function (*i.e.*, the degree-2 density and the number of introduced variables)—enabling it to be, in turn, used in an automatic transformation process that optimises quantum circuit metrics. We prove that a reduction iteration acts locally on the graph representation, which paves the way for future parallel execution. Moreover, it is easy to extend our proposed algorithm to not only allow for *quadratisations* (*i.e.*, the reduction to a degree-2 function), but also for higher-degree reductions (*i.e.*, degree-k, $k > 2$).

Moreover, we show in detail how industrial-like SAT instances are connected to quantum computing by a multitude of explicit transformation processes—also putting into perspective where *quadratisations* are located. We identify major limitations of scaling to larger problem instances and show explicitly how to circumvent them. As a result of that, we identify that lower-order models (such as QUBO) have advantages over higher-order models (**PUBO**), when it comes to scaling behaviour in quantum circuits (*e.g.*, for QAOA) and intermediate representations (*e.g.*, Ising models). However, PUBO models have better expressivity in higher abstraction levels and typical formulations of SAT lead to PUBO models. Our introduced LSR algorithm allows for fast transformation from higher-order to lower-order models (*e.g.*, QUBO), which on the one hand enables practitioners to use higher-order models in higher abstraction, while on the other hand taking advantage over the positive scaling behaviour of lower-order models in lower abstraction layers.

Ultimately, we desire an (automated) quantum toolchain that optimises metrics of hardware specific representations. Such a toolchain benefits from predictable transformation processes. With

this work, we show analytically and quantitatively to which degree and how transformations change properties of (intermediate) (quantum specific) representations.

References

[1] Tameem Albash and Daniel A. Lidar. 2018. Adiabatic quantum computation. *Reviews of Modern Physics* 90, 1 (Jan 2018), 015002. DOI: https://doi.org/10.1103/RevModPhys.90.015002

[2] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. 2009. Towards industrial-like random SAT instances. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence* (Pasadena, California, USA) *(IJCAI'09)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 387–392.

[3] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G. Katzgraber. 2019. Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics* 7 (2019). DOI: https://doi.org/10.3389/fphy.2019.00048

[4] Ryan Babbush, Jarrod R. McClean, Michael Newman, Craig Gidney, Sergio Boixo, and Hartmut Neven. 2021. Focus beyond quadratic speedups for error-corrected quantum advantage. *PRX quantum* 2, 1 (March 2021). DOI: 10.1103/prxquantum.2.010103

[5] Andreas Bayerstadler, Guillaume Becquin, Julia Binder, Thierry Botter, Hans Ehm, Thomas Ehmer, Marvin Erdmann, Norbert Gaus, Philipp Harbach, Maximilian Hess, et al. 2021. Industry quantum computing applications. *EPJ Quantum Technology* 8, 1 (11 2021). DOI: https://doi.org/10.1140/epjqt/s40507-021-00114-x

[6] Claude Berge and Jean Claude Fournier. 1991. A short proof for a generalization of Vizing's theorem. *Journal of Graph Theory* 15, 3 (July 1991), 333–336. DOI: https://doi.org/10.1002/jgt.3190150309

[7] Zhengbing Bian, Fabián A. Chudak, William G. Macready, and Geordie Rose. 2010. The Ising model : teaching an old problem new tricks. Retrieved from https://api.semanticscholar.org/CorpusID:15182277

[8] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. 2020. SMT encodings for Resource-Constrained Project Scheduling Problems. *Computers & Industrial Engineering* 149 (Nov. 2020), 106777. DOI: https://doi.org/10.1016/j.cie.2020.106777

[9] Endre Boros and Peter L. Hammer. 2002. Pseudo-Boolean optimization. *Discrete Applied Mathematics* 123, 1 (2002), 155–225. DOI: https://doi.org/10.1016/S0166-218X(01)00341-9

[10] Colin Campbell and Edward Dahl. 2022. QAOA of the highest order. In *Proceedings of the 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. 141–146. DOI: https://doi.org/10.1109/ICSA-C54293.2022.00035

[11] Cecilia Carbonelli, Michael Felderer, Matthias Jung, Elisabeth Lobe, Malte Lochau, Sebastian Luber, Wolfgang Mauerer, Rudolf Ramler, Ina Schaefer, and Christoph Schroth. 2024. *Challenges for Quantum Software Engineering: An Industrial Application Scenario Perspective*. Springer Nature Switzerland, 311–335. DOI: https://doi.org/10.1007/978-3-031-64136-7_12

[12] McGeoch Catherine and Farré Pau. 2020. *The D-Wave Advantage System: An Overview*. Technical Report MSU-CSE-06-2. DWave. 22 pages. Retrieved from https://www.dwavesys.com/media/3xvdipcn/14-1058a-a_advantage_processor_overview.pdf

[13] Vicky Choi. 2010. Adiabatic quantum algorithms for the NP-Complete maximum-weight independent set, exact cover and 3SAT problems. arXiv:1004.2226. Retrieved from https://arxiv.org/abs/1004.2226. (2010).

[14] Barry A. Cipra. 2000. The ising model is NP-complete. *SIAM News* 33, 6 (2000), 1–3.

[15] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing* (Shaker Heights, Ohio, USA) *(STOC'71)*. Association for Computing Machinery, New York, NY, USA, 151–158. DOI: https://doi.org/10.1145/800157.805047

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[17] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. 2019. On the qubit routing problem. In *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 135 (2019), 5:1–5:32. DOI: https://doi.org/10.4230/LIPICS.TQC.2019.5

[18] Nike Dattani. 2019. Quadratization in discrete optimization and quantum mechanics. DOI: https://doi.org/10.48550/ARXIV.1901.04405

[19] Dmitrii Dobrynin, Adrien Renaudineau, Mohammad Hizzani, Dmitri Strukov, Masoud Mohseni, and John Paul Strachan. 2024. Energy landscapes of combinatorial optimization in Ising machines. *Physical Review E* 110, 4 (Oct. 2024). DOI: https://doi.org/10.1103/physreve.110.045308

[20] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. arXiv:1411.4028. Retrieved from https://arxiv.org/abs/1411.4028. (2014).

[21] Tobias Friedrich, Anton Krohmer, Ralf Rothenberger, Thomas Sauerwald, and Andrew M. Sutton. 2017. Bounds on the satisfiability threshold for power law distributed random SAT. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. DOI: https://doi.org/10.4230/LIPICS.ESA.2017.37

[22] Tobias Friedrich, Anton Krohmer, Ralf Rothenberger, and Andrew M. Sutton. 2017. Phase transitions for scale-free SAT formulas. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence* (San Francisco, California, USA) *(AAAI'17)*. AAAI Press, 3893–3899.

[23] Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda. 2021. SAT competition 2020. *Artificial Intelligence* 301 (Dec. 2021), 103572. DOI: https://doi.org/10.1016/j.artint.2021.103572

[24] Fred W. Glover and Gary A. Kochenberger. 2018. A tutorial on formulating QUBO models. arXiv:1811.11538. Retrieved from https://arxiv.org/abs/1811.11538. (2018).

[25] Felix Greiwe, Tom Krüger, and Wolfgang Mauerer. 2023. Effects of imperfections on quantum algorithms: A software engineering perspective. In *Proceedings of the 2023 IEEE International Conference on Quantum Software (QSW)*. IEEE. DOI: https://doi.org/10.1109/qsw59989.2023.00014

[26] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th annual ACM symposium on Theory of computing (STOC '96)*. ACM Press, 212–219. DOI: https://doi.org/10.1145/237814.237866

[27] Matthew P. Harrigan, Kevin J. Sung, Matthew Neeley, Kevin J. Satzinger, Frank Arute, Kunal Arya, Juan Atalaya, Joseph C. Bardin, Rami Barends, Sergio Boixo, et al. 2021. Quantum approximate optimization of non-planar graph problems on a planar superconducting processor. *Nature Physics* 17, 3 (feb 2021), 332–336. DOI: https://doi.org/10.1038/s41567-020-01105-y

[28] Philipp Hauke, Helmut G. Katzgraber, Wolfgang Lechner, Hidetoshi Nishimori, and William D. Oliver. 2020. Perspectives of quantum annealing: Methods and implementations. *Reports on Progress in Physics* 83, 5 (May 2020), 054401. DOI: https://doi.org/10.1088/1361-6633/ab85b8

[29] Rebekah Herrman, James Ostrowski, Travis S. Humble, and George Siopsis. 2021. Lower bounds on circuit depth of the quantum approximate optimization algorithm. *Quantum Information Processing* 20, 2 (Feb. 2021), 59. DOI: https://doi.org/10.1007/s11128-021-03001-7

[30] Yuichi Hirata, Masaki Nakanishi, Shigeru Yamashita, and Yasuhiko Nakashima. 2009. An efficient method to convert arbitrary quantum circuits to ones on a linear nearest neighbor architecture. In *Proceedings of the 2009 3rd International Conference on Quantum, Nano and Micro Technologies*. 26–33. DOI: https://doi.org/10.1109/ICQNM.2009.25

[31] Hiroshi Ishikawa. 2014. Higher-order clique reduction without auxiliary variables. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. 1362–1369. DOI: https://doi.org/10.1109/CVPR.2014.177

[32] Grant Jenks. 2019. Sorted Dict. Retrieved August 26, 2024 from https://grantjenks.com/docs/sortedcontainers/sorteddict.html#sortedcontainers.SortedDict.peekitem

[33] Matthias Jung, Sven O. Krumke, Christof Schroth, Elisabeth Lobe, and Wolfgang Mauerer. 2025. *QCEDA: Using Quantum Computers for EDA*. Springer Nature Switzerland, 32–46. DOI: https://doi.org/10.1007/978-3-031-78380-7_3

[34] Richard M. Karp. 1972. *Reducibility among Combinatorial Problems*. Springer US, 85–103. DOI: https://doi.org/10.1007/978-1-4684-2001-2_9

[35] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (May 1983), 671–680. DOI: https://doi.org/10.1126/science.220.4598.671

[36] Donald E. Knuth. 1997. *The Art of Computer Programming, volume 2 (3rd ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA.

[37] Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. 2014. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization* 28, 1 (April 2014), 58–81. DOI: https://doi.org/10.1007/s10878-014-9734-0

[38] Tom Krüger and Wolfgang Mauerer. 2020. Quantum annealing-based software components: An experimental case study with SAT solving. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 445–450. DOI: https://doi.org/10.1145/3387940.3391472

[39] Elisabeth Lobe. 2023. quark: Quantum application reformulation kernel. (2023). DOI: https://doi.org/10.18420/INF2023_123

[40] Andrew Lucas. 2014. Ising formulations of many NP problems. *Frontiers in Physics* 2 (2014). DOI: https://doi.org/10.3389/fphy.2014.00005

[41] Ritajit Majumdar, Dhiraj Madan, Debasmita Bhoumik, Dhinakaran Vinayagamurthy, Shesha Raghunathan, and Susmita Sur-Kolay. 2021. Optimizing Ansatz Design in QAOA for Max-cut. DOI: https://doi.org/10.48550/ARXIV.2106.02812

[42] Joao Marques-Silva. 2008. Practical applications of boolean satisfiability. In *Proceedings of the 2008 9th International Workshop on Discrete Event Systems*. 74–80. DOI: https://doi.org/10.1109/WODES.2008.4605925

[43] Wolfgang Mauerer and Stefanie Scherzinger. 2022. 1-2-3 reproducibility for quantum software experiments. In *Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1247–1248. DOI: https://doi.org/10.1109/SANER53432.2022.00148

[44] Catherine C. McGeoch. 2014. *Adiabatic Quantum Computation and Quantum Annealing*. Springer International Publishing, Cham.

[45] J. A. Montañez-Barrera and Kristel Michielsen. 2025. Toward a linear-ramp QAOA protocol: Evidence of a scaling advantage in solving some combinatorial optimization problems. *npj Quantum Information* 11, 1, Article 131 (Aug. 2025). DOI: https://doi.org/10.1038/s41534-025-01082-1

[46] T. Monz, K. Kim, W. Hänsel, M. Riebe, A. S. Villar, P. Schindler, M. Chwalla, M. Hennrich, and R. Blatt. 2009. Realization of the quantum toffoli gate with trapped ions. *Physical Review Letters* 102, 4 (1 2009), 040501.. DOI: https://doi.org/10.1103/PhysRevLett.102.040501

[47] Antonio Morgado, Federico Heras, Mark Liffiton, Jordi Planes, and Joao Marques-Silva. 2013. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints* 18, 4 (July 2013), 478–534. DOI: https://doi.org/10.1007/s10601-013-9146-2

[48] Marc Mézard and Riccardo Zecchina. 2002. Random K-satisfiability problem: From an analytic solution to an efficient algorithm. *Physical Review E* 66, 5 (Nov. 2002). DOI: https://doi.org/10.1103/physreve.66.056126

[49] Alexander G. Nikolaev and Sheldon H. Jacobson. 2010. Simulated annealing. In *International Series in Operations Research & Management Science*. Springer US, Boston, MA, 1–39.

[50] Emile Okada, Richard Tanburn, and Nikesh S. Dattani. 2015. Reducing multi-qubit interactions in adiabatic quantum computation without adding auxiliary qubits. Part 2: The "split-reduc" method and its application to quantum determination of Ramsey numbers. arXiv:1508.07190.. Retrieved from https://arxiv.org/abs/1508.07190. (2015).

[51] Elijah Pelofske, Andreas Bärtschi, and Stephan Eidenbenz. 2024. Short-depth QAOA circuits and quantum annealing on higher-order ising models. *npj Quantum Information* 10, 1 (2024), 30.

[52] Abraham P. Punnen (Ed.). 2022. *The Quadratic Unconstrained Binary Optimization Problem*. Springer International Publishing. DOI: https://doi.org/10.1007/978-3-031-04520-2

[53] Salonik Resch and Ulya R. Karpuzcu. 2021. Benchmarking quantum computers and the impact of quantum noise. *ACM Computing Surveys* 54, 7, Article 142 (July 2021), 35 pages. DOI: https://doi.org/10.1145/3464420

[54] Hila Safi, Karen Wintersperger, and Wolfgang Mauerer. 2023. Influence of HW-SW-Co-design on quantum computing scalability. In *Proceedings of the 2023 IEEE International Conference on Quantum Software (QSW)*. IEEE. DOI: https://doi.org/10.1109/qsw59989.2023.00022

[55] Irmi Sax, Sebastian Feld, Sebastian Zielinski, Thomas Gabor, Claudia Linnhoff-Popien, and Wolfgang Mauerer. 2020. Approximate approximation on a quantum annealer. In *Proceedings of the 17th ACM International Conference on Computing Frontiers*. Association for Computing Machinery, New York, NY, USA, 108–117. DOI: https://doi.org/10.1145/3387902.3392635

[56] Lukas Schmidbauer and Wolfgang Mauerer. 2025. SAT strikes back: Parameter and path relations in quantum toolchains. In *Proceedings of the 2025 IEEE International Conference on Quantum Software (QSW)*. IEEE, 01–12. DOI: https://doi.org/10.1109/qsw67625.2025.00021

[57] Lukas Schmidbauer, Karen Wintersperger, Elisabeth Lobe, and Wolfgang Mauerer. 2024. Polynomial reduction methods and their impact on QAOA circuits. In *Proceedings of the 2024 IEEE International Conference on Quantum Software (QSW)*. IEEE, 35–45. DOI: https://doi.org/10.1109/qsw62656.2024.00018

[58] Manuel Schönberger, Stefanie Scherzinger, and Wolfgang Mauerer. 2023. Ready to leap (by Co-Design)? join order optimisation on quantum hardware. In *Proceedings of the ACM SIGMOD/PODS International Conference on Management of Data*. DOI: https://doi.org/10.1145/3588946

[59] Yotam Shapira, Ravid Shaniv, Tom Manovitz, Nitzan Akerman, Lee Peleg, Lior Gazit, Roee Ozeri, and Ady Stern. 2020. Theory of robust multiqubit nonadiabatic gates for trapped ions. *Physical Review A* 101, 3 (3 2020), 032330. DOI: https://doi.org/10.1103/PhysRevA.101.032330

[60] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintao Pereira. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization* (Vienna, Austria) *(CGO 2018)*. Association for Computing Machinery, New York, NY, USA, 113–125. DOI: https://doi.org/10.1145/3168822

[61] Andrew Steane. 1998. Quantum computing. *Reports on Progress in Physics* 61, 2 (Feb. 1998), 117–173.

[62] Chico Sundermann, Tobias Heß, Michael Nieke, Paul Maximilian Bittner, Jeffrey M. Young, Thomas Thüm, and Ina Schaefer. 2023. Evaluating state-of-the-art # SAT solvers on industrial configuration spaces. *Empirical Software Engineering* 28, 2, Article 29 (Jan. 2023). DOI: https://doi.org/10.1007/s10664-022-10265-9

[63] Richard Tanburn, Emile Okada, and Nike Dattani. 2015. Reducing multi-qubit interactions in adiabatic quantum computation without adding auxiliary qubits. Part 1: The "deduc-reduc" method and its application to quantum factorization of numbers. arXiv:1508.04816. Retrieved from https://arxiv.org/abs/1508.04816. (2015).

[64] Simon Thelen, Hila Safi, and Wolfgang Mauerer. 2024. Approximating under the influence of quantum noise and compute power. In *Proceedings of the 2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 274–279. DOI: https://doi.org/10.1109/qce60285.2024.10291

[65] Lukas Windgätter and Elisabeth Lobe. 2025. Quantum optimization applications with quark and quapps: Bridging the gap between application and hardware [Focus: Quantum Software and its Engineering]. *IEEE Software* 42, 5 (Sept. 2025), 34–42. DOI: https://doi.org/10.1109/ms.2025.3564146

[66] Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. 2015. Swapping labeled tokens on graphs. *Theoretical Computer Science* 586 (2015), 81–94. DOI: https://doi.org/10.1016/j.tcs.2015.01.052. Fun with Algorithms.

[67] Tao Yue, Wolfgang Mauerer, Shaukat Ali, and Davide Taibi. 2023. *Challenges and Opportunities in Quantum Software Architecture*. Springer Nature Switzerland, 1–23. DOI: https://doi.org/10.1007/978-3-031-36847-9_1

[68] Chi Zhang, Ari B. Hayes, Longfei Qiu, Yuwei Jin, Yanhao Chen, and Eddy Z. Zhang. 2021. Time-optimal qubit mapping. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS'21)*. Association for Computing Machinery, New York, NY, USA, 360–374. DOI: https://doi.org/10.1145/3445814.3446706

[69] Sebastian Zielinski, Jonas Nüßlein, Jonas Stein, Thomas Gabor, Claudia Linnhoff-Popien, and Sebastian Feld. 2023. Pattern QUBOs: Algorithmic construction of 3SAT-to-QUBO transformations. *Electronics* 12, 16 (Aug. 2023), 3492. DOI: https://doi.org/10.3390/electronics12163492

Received 20 January 2025; revised 4 September 2025; accepted 15 February 2026