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

LUKAS SCHMIDBAUER, Technical University of Applied Sciences, Germany
ELISABETH LOBE, German Aerospace Center (DLR), Institute of Software Technology, Department High-Performance Computing, Germany
INA SCHAEFER, KIT, Institute of Information Security and Dependability (KASTEL), Germany
WOLFGANG MAUERER, Technical University of Applied Sciences, Germany and Siemens AG, Technology, 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 paper, we consider a specific class of higher-level representations, i.e. polynomial unconstrained binary optimisation problems, and devise novel automatic transformation mechanisms into widely used quadratic unconstrained binary optimisation problems that substantially improve efficiency and versatility over the state of the art. We also identify what influence factors of lower-level details can be abstracted away in the transformation process, and which details must be made available to higher-level abstractions.

CCS Concepts: • **Theory of computation → Quantum computation theory**; *Computational complexity and cryptography*; **Graph algorithms analysis**; **Data structures design and analysis**; • **Software and its engineering** → *Formal methods*; • **General and reference** → **Performance**; • **Applied computing** → **Physics**;

Additional Key Words and Phrases: Pseudo boolean function, Graphs, Performance, Algorithmic optimisation

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

Authors' Contact Information: Lukas Schmidbauer, lukas.schmidbauer@othr.de, Technical University of Applied Sciences, Regensburg, Germany; Elisabeth Lobe, elisabeth.lobe@dlr.de, German Aerospace Center (DLR), Institute of Software Technology, Department High-Performance Computing, Braunschweig, Germany; Ina Schaefer, ina.schaefer@kit.edu, KIT, Institute of Information Security and Dependability (KASTEL), Karlsruhe, Germany; Wolfgang Mauerer, wolfgang.mauerer@othr.de, Technical University of Applied Sciences, Regensburg, Germany and Siemens AG, Technology, Germany.

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 [27, 35, 38], or which physical qubits can be brought into direct interaction [11, 18, 36, 40, 41]. 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 [34]. Starting from a QUBO representation, one has again many choices regarding a concrete solver strategy (*e.g.*, Quantum Approximate Optimisation Algorithm (QAOA) [7, 16, 25, 39], Annealing [1, 23, 32], Grover search [15]), 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 [17], which 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 in a previous work with regard to QAOA circuits [33] 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 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 that this classical effort is unfeasible in the current implementation for practically relevant problem sizes for an existing monomial-based implementation. 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.

In classical computing, fault tolerance is a given property of hardware. In quantum computing it is believed that fault tolerance is the missing integral part of enjoying advantages gained from the fundamentally different computational model [3, 14, 31, 37]. 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 paper is structured as follows: Sec. 2 formalises our problem, establishes notational conventions and reviews existing reduction methods. Sec. 3 introduces the data structures that our approach is based on and analyses them mathematically, which paves the way for complexity-theoretical performance gains. Sec. 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 Sec. 5. Furthermore, Sec. 5 shows empirically how a concrete implementation performs in comparison to a currently available implementation. We conclude our study in Sec. 6.

The paper is augmented by a supplementary website and a comprehensive reproduction package [26] (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) [5, 29, 42]. On the one hand, they are suitable for encoding optimisation problems with **NP**-complete decision variants, given their complexity-theoretic properties [9, 13]. On the other hand, they provide a consolidated interface to encode many COPs in quantum frameworks [2, 8]—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 [6]:

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

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 Eq. 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 $|m| := \deg(m)$. Furthermore, the degree of a PBF f is the maximum degree of its monomials. For the example of Eq. 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 [13, 22, 30]. They are a standard interface to widely used frameworks, such as quantum annealers [8] and digital annealing [2]. 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 formulation optimisation problems for real world applications, higher-degree terms can provide better expressivity and are sometimes necessary to encode constraints [4]. (In-)equality constraints can be encoded in QUBOs by adding suiting penalty terms [13]. 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*.

[1] A monomial is itself also a PBF.

2.2 Quadratisation

Starting from a higher-degree PBF f, there are many methods, reviewed by Dattani [12], to reduce the degree of f. For example, it is possible to split the objective function [28] or to pre determine variable assignments and then exclude monomials in special cases [19]. A versatile *quadratisation* method, reviewed by Boros [6], 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 [6]

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

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 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:

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

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 while**
9: return f, p

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'. Alg. 1 shows the basic structure of an iterative *quadratisation*, as for example done in `quark` [24], 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 (Alg. 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

[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.

inefficient repeated searching for a variable pair in all monomials of f per iteration in the shown implementation.

In [33] 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 [33] 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 Sec. 3.1 and prove important properties that lay the basis for a complexity theoretic performance gain in Sec. 3.2.

3 Graph Representation

3.1 Fundamentals

Polynomials can be represented in a variety of ways [21]. 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 $x_i x_j$ 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 Tab. 1, and secondly using these indices as edge labels. Internally, a dictionary represents this relation—allowing for fast average case access.

Table 1. Internal representation of example PBF of Eq. 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

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 Eq. (3), recall that $f(x_1, \ldots, x_6) = \pi x_1x_2x_3 - 13x_2x_4x_5x_6 + 7x_1x_3$ and let its monomial index mapping be defined as in Tab. 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 Fig. 1.

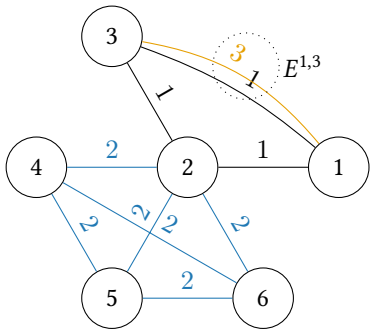

Fig. 1. Multi-graph representation of $f(x_1, \ldots, x_6) = \pi x_1 x_2 x_3 - 13 x_2 x_4 x_5 x_6 + 7 x_1 x_3$, where edge labels correspond to monomial indices according to Tab. 1.

Let $f : \{0, 1\}^n \to \mathbb{R}$ be a PBF and let $G_f = (V_f, E_f)$ be the corresponding graph. Let us now 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 $\deg(f) > 2$, monomials in f introduce edges to G_f and therefore $|B_f| > 1$. Since a variable pair $x_i x_j$ can at maximum occur in every (at least quadratic) monomial in f, we have for the corresponding multiplicity $\beta(i, j) \leq T_f$, where T_f denotes the total number of monomials in f, and we have $B_f \subseteq \{1, \ldots, T_f\}$. 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 $\sum_{\beta \in B_f} |R_f(\beta)| = |V_f|^2$. Tab. 2 continues the example from Fig. 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 n-th 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:

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

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.

Table 2. Mapping R_f of increasing multiplicity to sets of node-pairs for example of Fig. 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	$\{\{1,3\}\}$

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 $f_0 \coloneqq f$ 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 $f_{t+i}, i \in \mathbb{N}$ [33]. We go into more detail about the inner workings of this algorithm in Sec. 4.

In the following, we list important properties that pave the way for algorithmic optimisation of Alg. 1 and are the cornerstone for our proposed algorithm in Sec. 4, as well as for bounding runtime in Sec. 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 Eq. 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 Sec. 2.2). Hence, it suffices to show that all monomials containing x_i and x_j

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

[5] For instance: $x_1x_2x_3 \in f_t$ and $x_3x_4x_5x_6 \in f_t$.

[6] Monomials are totalled. For example, let $f(x_1,x_2,x_3) = x_2x_1$. Then, $f(x_1,x_2,x_3) + x_1x_2 = 2x_1x_2$.

already occur on edges $e \in E_f^{i,j}$ (*i.e.*, edges between nodes i and j). Without loss of generality, let $m = x_1x_2x_3 \dots x_ix_j$, $i < j$ 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 $\mathrm{p}(x_i, x_j, y_h) = 3y_h + x_ix_j - 2x_iy_h - 2x_jy_h$, 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 Tab. 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_ix_j. Consequently, x_ix_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_ix_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_iy_h and x_jy_h).

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

Proof. Let $m_t = x_1x_2 \dots x_kx_ix_j$ $(k < i < j)$ be a monomial and let $m_{t+1} = x_1x_2 \dots x_ky_h$ 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 on 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 $m = \dots x_ax_ix_j$. 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_ix_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: $x_1x_2 + x_1x_2x_3 + x_1x_2x_3x_4 \longrightarrow x_1x_2 + x_1x_2x_3 + x_1x_2y_1 \longrightarrow x_1x_2 + x_1y_2 + x_1x_2y_1 \longrightarrow x_1x_2 + x_1y_2 + x_1y_3$. 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: $x_1x_2 + x_1x_2x_3 + x_1x_2x_3x_4 \longrightarrow y_0 + y_0x_3 + y_0x_3x_4 \longrightarrow y_0 + y_0x_3 + y_0y_1$.[7] The size of the highest degree monomial gives the minimum number of reduction steps or introduced variables via Boros' method [6], that is,

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

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. Fig. 2 shows the open relation.

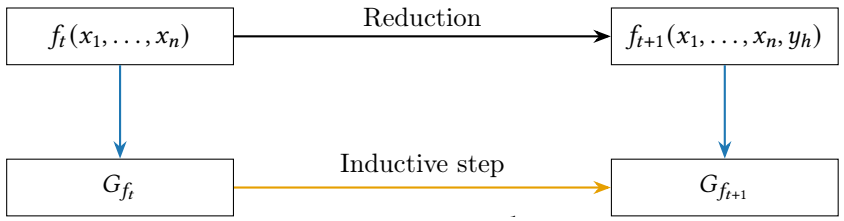

Fig. 2. Known reduction (black), introduced construction (blue) and graph evolution (yellow).

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_ix_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_ix_j. Furthermore, it suffices to consider neighbouring nodes that are connected to both i and j (see Property 5). Fig. 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_ix_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.

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

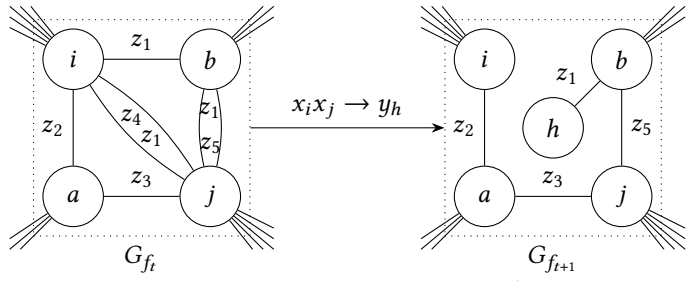

Fig. 3. Shows the inductive graph evolution for changing edges from G_{f_t} to $G_{f_{t+1}}$. The penalty term induced edges $\{(i, h, z_k), (j, h, z_{k+1}), (i, j, z_{k+2})\}$ for a fitting $k \in \mathbb{N}$ can be excluded from $G_{f_{t+1}}$ (see Property 3) and are not shown.

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 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 Eq. 6.

Recall the basic structure for a *quadratisation* algorithm (Alg. 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 Sec. 3. Part (b) still results in the variable pair being replaced in all occurring monomials and is therefore left unchanged result-related.

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 $B := \{\beta_1, \ldots, \beta_{|B|}\}$ is the set of multiplicities in G and $\beta_1 < \ldots < \beta_{|B|}$, as in Sec. 3. In Eq. 7, we define $\tilde{\beta}_q := \beta_{\lceil q \cdot |B| \rceil}$ for a percentile q. 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 Sec. 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 Sec. 3. Function UPDATE_GRAPH_DATA(.) firstly selects monomial indices from edges between nodes i and j in set Z (Alg. 2: l. 19), since all

[8]It may be the case that $m_1 = x_i x_j$ and $\deg(m_2) > 2$.

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 (Alg. 2: l. 20)—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 $x_1x_2 \dots x_ky_h$ 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]:

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

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

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

Also, edges between nodes i and j are removed[10] (Alg. 2: l. 29). Last but not least, function R changes, when the graph is altered (see Eq. 13). Since node-pairs are mapped from the set of former multiplicities in G, set M saves their values and corresponding node-pairs. Therefore, any node-pair in M can firstly be deleted from R and secondly be added with its new multiplicity again (Alg. 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(.) (Alg. 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 Tab. 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 (Alg. 2: l. 8) to be able to later scale it properly and therefore adhere to the *quadratisation* criteria given in Eq. 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 [33].

As a prerequisite of stage 2 (Alg. 2: l. 12), we know that there are no more multi-edges in the graph:

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

[9] Take into consideration that for simplicity in pseudocode, we do not enforce a total ordering of indices in edges, as introduced in Sec. 3.

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

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 [33], 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(.) (Alg. 2: l. 14) quadratises a monomial m and adds the necessary penalty terms to p. Hereafter (Alg. 2: l. 17), input function f is now a quadratic PBF, constrained by the penalty term p.

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

1: $h \leftarrow 1$ ▷ Stage 1: Graph-based reduction
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: $\{i, j\} \leftarrow$ CHOOSE_RANDOM_ELEMENT$(R(\tilde{\beta}_q))$
6: $f \leftarrow$ REPLACE_VAR_PAIR(G, f, x_i, x_j, y_h)
7: $G, R \leftarrow$ UPDATE_GRAPH_DATA(G, R, i, j, h)
8: $p \leftarrow p + \mathrm{p}(x_i, x_j, y_h)$
9: $h \leftarrow h + 1$
10: **end while**
11: ▷ Now we have $\forall i, j \in V : \beta(i, j) \leq 1$
12: **for** $m \in f$ with $\deg(m) > 2$ **do** ▷ Stage 2: Independent monomial-based reduction
13: **while** $\deg(m) > 2$ **do**
14: $f, p \leftarrow$ MULTI_REDUCE(f, p, m)
15: **end while**
16: **end for**
17: return f, p

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

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 Eq. 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 Sec. 2.2), the introduced LSR algorithm adheres to the *quadratisation* criteria in Eq. 6 [6]. 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 $f_0 \coloneqq f : \{0,1\}^n \to \mathbb{R}$ 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 3 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 Alg. 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 [24] 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

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

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

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 [33], 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 Alg. 2: ll. 5-9).

Creating the monomial index dictionary, as in Tab. 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 Sec. 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 $O(\log(n))$ [20]. Let the set of multiplicities be denoted by $B_{f_0} = \{\beta_{f_0}(i,j) \mid i,j \in V_{f_0}\}$ (see Sec. 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($R(\tilde{\beta}_q)$) (Alg. 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 $x_i x_j$ 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 (Alg. 2: l. 19). 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 $O(\beta_{f_{t-1}}(i,j) \cdot |V_{f_{t-1}}|^2)$.

[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.

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 (Alg. 2: ll. 21-27). Updating the edges of G (Alg. 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 $O(\sum_{z\in Z}|m_z|)$ (Alg. 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(.) (Alg. 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 Fig. 3). Let ω be the index of the last reduction iteration in LSR stage 1. Then, $|B_{f_\omega}| = 0$.

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 Sec. 3.2). Since every edge $E_f^{i,j}$ (see Sec. 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 Tab. 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 Tab. 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 Alg. 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 (Alg. 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 Eq. 4).

In a complete reduction iteration (Alg. 2: ll. 5-9), function UPDATE_R(.) (Alg. 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 [33], we can bound the number of iterations in Alg. 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' [6] reduction method, we can apply multiple

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

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

reductions at once. For example, applying $x_1x_2x_3x_4x_5 \rightarrow y_1y_2x_5$ in a single step by replacing x_1x_2 with y_1 and x_3x_4 with y_2. MULTI_REDUCE(.) (Alg. 2: l. 14) 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:

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

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 (Alg. 2: l. 13-15) and, due to its convergence, a total runtime of

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

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.

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 (Alg. 2: l. 12-16)[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 (Alg. 2: l. 13-15).

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 (Alg. 2: l. 12-16) to T_{f_0}. On the other hand, the total runtime for LSR stage 2 is given by

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

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 Alg. 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_ix_j in monomials of f. The

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

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 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 Sec. 5.2. On the one hand, indexing monomials in a dictionary with exactly T_f (see Tab. 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 [10]. 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

[17]The penalty dictionary is separable from it: see Property 3.

[18]We exclude node-pairs, containing less than two edges.

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 Sec. 5.3 and Sec. 5.2, but a standard python dictionary implementation. Therefore, we expect the following scaling behaviour in runtime to be an upper bound.

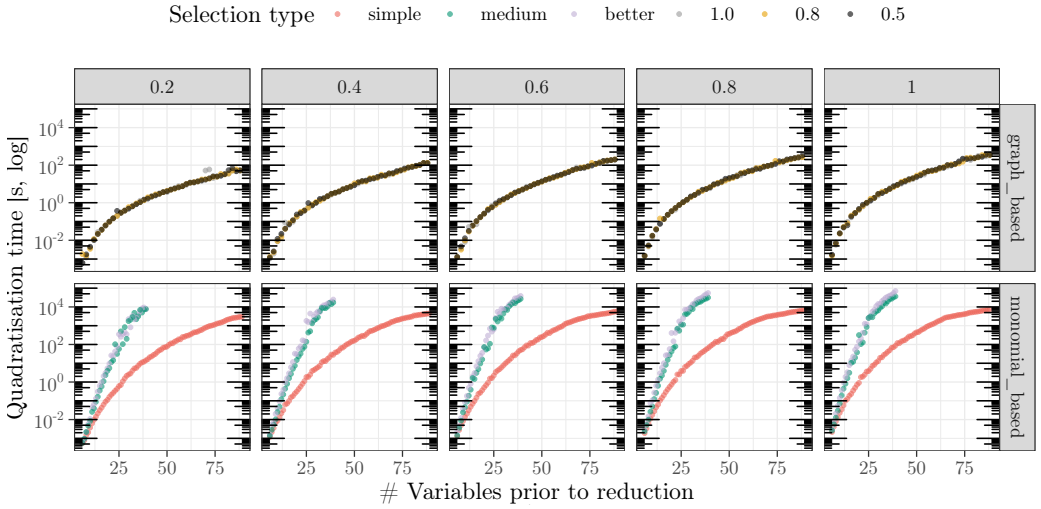

Fig. 4. Time in seconds (y-axis) for the *quadratisation* of a $\deg(f) = 4$ function f vs problem size (x-axis: Number of variables). New LSR algorithm (top row) compared to existing monomial-based (bottom row). Vertical facets: Different base polynomial densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f) \in \{0.2, 0.4, \ldots, 1.0\}$). 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.

Fig. 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 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 $\sum_{m \in f_{t-1}} 2|m|$

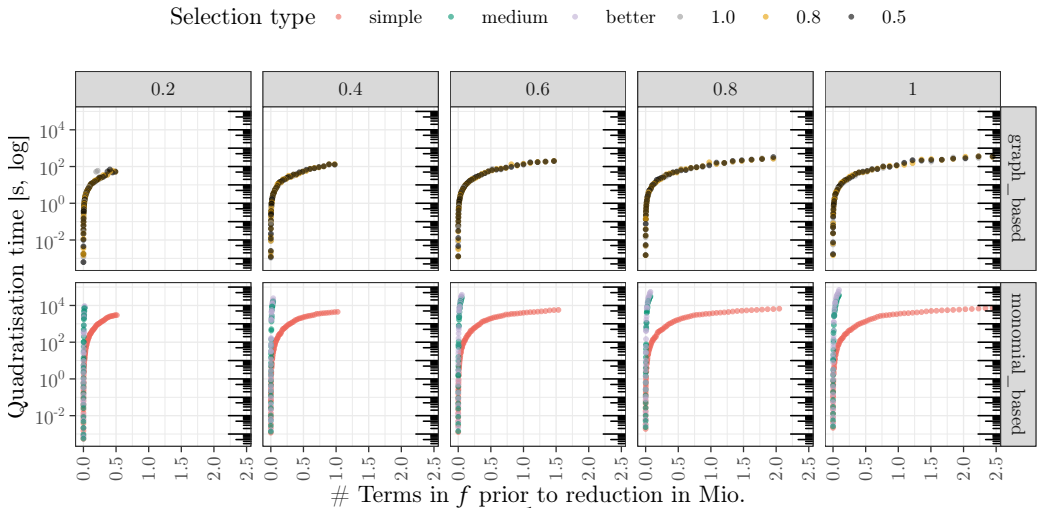

Fig. 5. Time in seconds (y-axis) for the *quadratisation* of a $\deg(f) = 4$ function f vs problem size (x-axis: Number of Terms). New LSR algorithm (top row) compared to existing monomial-based (bottom row). Vertical facets: Different base polynomial densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f) \in \{0.2, 0.4, \ldots, 1.0\}$). 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.

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, Fig. 5 shows their influence on the runtime (y-axis; log scale) for the same input PBFs as in Fig. 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 -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, $d_1 \to 1$[33]. Fig. 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 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 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 [33], where we analyse the effect of monomial-based reduction in the context of a Job-Shop Scheduling problem.

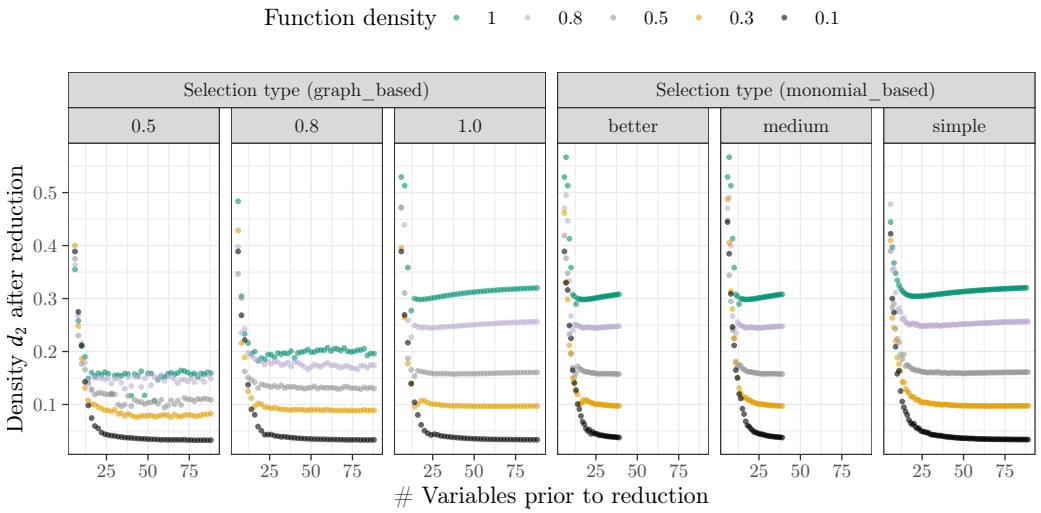

Fig. 6. Degree-2 density (y-axis) for the *quadratisation* of a $\deg(f) = 4$ function f vs problem size (x-axis: Number of variables) for different base polynomial densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f) \in \{0.1, 0.3, 0.5, 0.8, 1.0\}$). New algorithm on the left compared to monomial-based 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.

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. Fig. 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. Fig. 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 Fig. 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 Tab. 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.

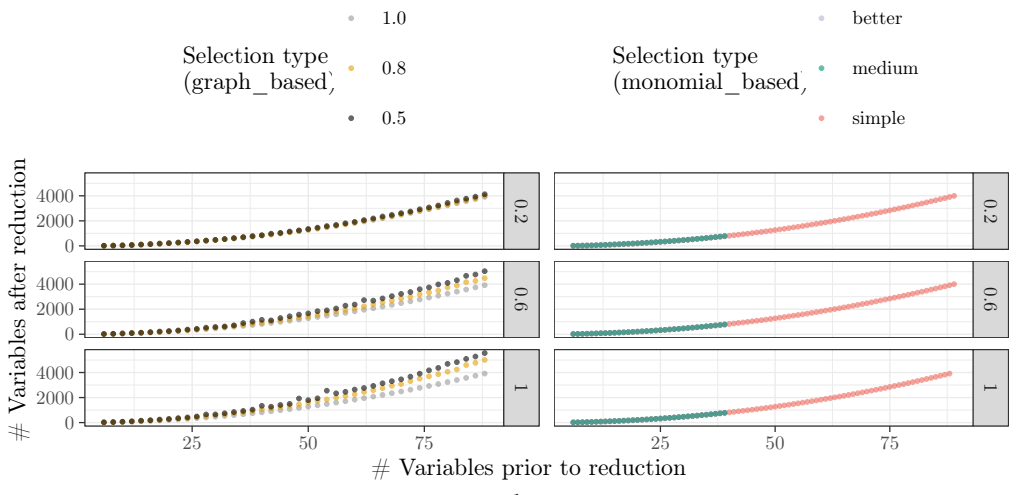

Fig. 7. Number of variables after *quadratisation* (y-axis) for the reduction of a $\deg(f) = 4$ function f vs problem size (x-axis: Number of variables). New algorithm (left column) compared to monomial-based (right column). Vertical facets: Different base polynomial densities (*i.e.*, $d_1(f) = d_2(f) = d_3(f) = d_4(f) \in \{0.2, 0.6, 1.0\}$). 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.

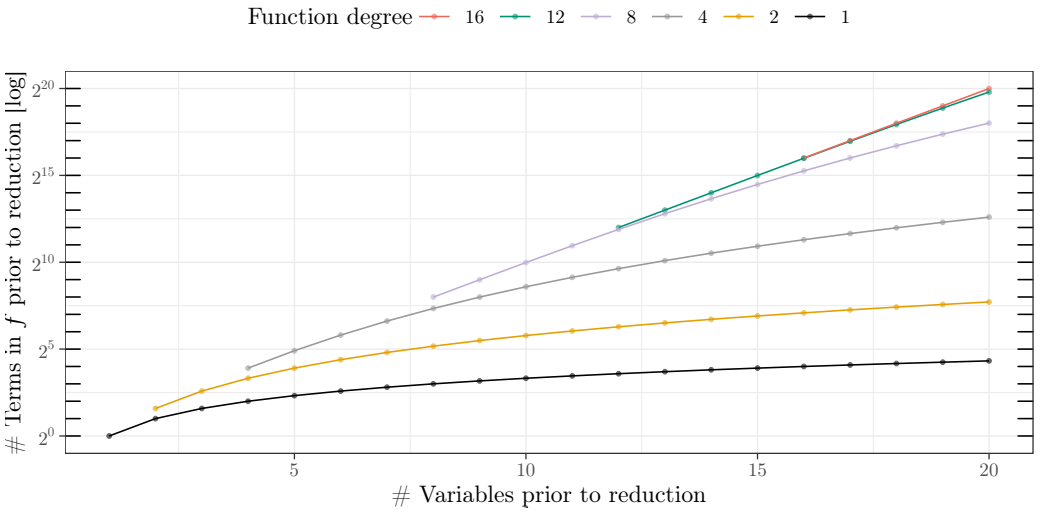

Fig. 8. Number of variables (x-axis) vs number of terms (y-axis) prior to reduction for functions f such that $deg(f)$ = Function degree and $d_1(f) = d_2(f) = \ldots = d_{\deg(f)}(f) = 1$.

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 Fig. 8. Therefore, as the number of 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 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 also give an argument on why higher-degree formulations of COPs tend towards sparse PBFs.

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, $k > 2$).

Acknowledgements We acknowledge support from German Federal Ministry of Education and Research (BMBF), funding program "Quantum Technologies—from Basic Research to Market", grant #13N15647 and #13N16092 (LS, WM). WM acknowledges support by the High-Tech Agenda Bavaria.

References

[1] Tameem Albash and Daniel A. Lidar. 2018. Adiabatic quantum computation. *Rev. Mod. Phys.* 90 (Jan 2018), 015002. Issue 1. https://doi.org/10.1103/RevModPhys.90.015002

[2] 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). https://doi.org/10.3389/fphy.2019.00048

[3] 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).

[4] Andreas Bayerstadler, Guillaume Becquin, Julia Binder, Thierry Botter, Hans Ehm, Thomas Ehmer, Marvin Erdmann, Norbert Gaus, Philipp Harbach, Maximilian Hess, Johannes Klepsch, Martin Leib, Sebastian Luber, Andre Luckow, Maximilian Mansky, Wolfgang Mauerer, Florian Neukart, Christoph Niedermeier, Lilly Palackal, Ruben Pfeiffer, Carsten Polenz, Johanna Sepulveda, Tammo Sievers, Brian Standen, Michael Streif, Thomas Strohm, Clemens Utschig-Utschig, Daniel Volz, Horst Weiss, and Fabian Winter. 2021. Industry Quantum Computing Applications. *EPJ Quantum Technology* 8, 1 (11 2021). https://doi.org/10.1140/epjqt/s40507-021-00114-x

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

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

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

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

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

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

[11] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. 2019. On the Qubit Routing Problem. (2019). https://doi.org/10.4230/LIPICS.TQC.2019.5

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

[13] Fred W. Glover and Gary A. Kochenberger. 2018. A Tutorial on Formulating QUBO Models. *CoRR* abs/1811.11538 (2018). arXiv:1811.11538 http://arxiv.org/abs/1811.11538

[14] Felix Greiwe, Tom Krüger, and Wolfgang Mauerer. 2023. Effects of Imperfections on Quantum Algorithms: A Software Engineering Perspective. In *2023 IEEE International Conference on Quantum Software (QSW)*. IEEE. https://doi.org/10.1109/qsw59989.2023.00014

[15] Lov K. Grover. 1996. A fast quantum mechanical algorithm for database search. https://doi.org/10.48550/ARXIV.QUANT-PH/9605043

[16] Matthew P. Harrigan, Kevin J. Sung, Matthew Neeley, Kevin J. Satzinger, Frank Arute, Kunal Arya, Juan Atalaya, Joseph C. Bardin, Rami Barends, Sergio Boixo, Michael Broughton, Bob B. Buckley, David A. Buell, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Sean Demura, Andrew Dunsworth, Daniel Eppens, Austin Fowler, Brooks Foxen, Craig Gidney, Marissa Giustina, Rob Graff, Steve Habegger, Alan Ho, Sabrina Hong, Trent Huang, L. B. Ioffe, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Cody Jones, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Seon Kim, Paul V. Klimov, Alexander N. Korotkov, Fedor Kostritsa, David Landhuis, Pavel Laptev, Mike Lindmark, Martin Leib, Orion Martin, John M. Martinis, Jarrod R. McClean, Matt McEwen, Anthony Megrant, Xiao Mi, Masoud Mohseni, Wojciech Mruczkiewicz, Josh Mutus, Ofer Naaman, Charles Neill, Florian Neukart, Murphy Yuezhen Niu, Thomas E. O'Brien, Bryan O'Gorman, Eric Ostby, Andre Petukhov, Harald Putterman, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Andrea Skolik, Vadim Smelyanskiy, Doug Strain, Michael Streif, Marco Szalay, Amit Vainsencher, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Leo Zhou, Hartmut Neven, Dave Bacon, Erik Lucero, Edward Farhi, and Ryan Babbush. 2021. Quantum approximate optimization of non-planar graph problems on a planar superconducting processor. *Nature Physics* 17, 3 (feb 2021), 332–336. https://doi.org/10.1038/s41567-020-01105-y

[17] 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. https://doi.org/10.1088/1361-6633/ab85b8

[18] 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 *2009 Third International Conference on Quantum, Nano and Micro Technologies*. 26–33. https://doi.org/10.1109/ICQNM.2009.25

[19] Hiroshi Ishikawa. 2014. Higher-Order Clique Reduction without Auxiliary Variables. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 1362–1369. https://doi.org/10.1109/CVPR.2014.177

[20] Grant Jenks. [n. d.]. Sorted Dict. https://grantjenks.com/docs/sortedcontainers/sorteddict.html#sortedcontainers.SortedDict.peekitem. Accessed: 2024-08-26.

[21] Donald E. Knuth. 1997. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA.

[22] 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. https://doi.org/10.1007/s10878-014-9734-0

[23] 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. https://doi.org/10.1145/3387940.3391472

[24] Elisabeth Lobe. 2023. quark: QUantum Application Reformulation Kernel. (2023). https://doi.org/10.18420/INF2023_123

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

[26] Wolfgang Mauerer and Stefanie Scherzinger. 2022. 1-2-3 Reproducibility for Quantum Software Experiments. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1247–1248. https://doi.org/10.1109/SANER53432.2022.00148

[27] 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. *Phys. Rev. Lett.* 102 (1 2009), 040501. Issue 4. https://doi.org/10.1103/PhysRevLett.102.040501

[28] 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 [quant-ph]

[29] 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.

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

[31] Salonik Resch and Ulya R. Karpuzcu. 2021. Benchmarking Quantum Computers and the Impact of Quantum Noise. *ACM Comput. Surv.* 54, 7, Article 142 (July 2021), 35 pages. https://doi.org/10.1145/3464420

[32] 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. https://doi.org/10.1145/3387902.3392635

[33] Lukas Schmidbauer, Karen Wintersperger, Elisabeth Lobe, and Wolfgang Mauerer. 2024. Polynomial Reduction Methods and their Impact on QAOA Circuits. In *IEEE International Conference on Quantum Software (QSW)*.

[34] Manuel Schönberger, Stefanie Scherzinger, and Wolfgang Mauerer. 2023. Ready to Leap (by Co-Design)? Join Order Optimisation on Quantum Hardware. In *Proceedings of ACM SIGMOD/PODS International Conference on Management of Data*. https://doi.org/10.1145/3588946

[35] 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. *Phys. Rev. A* 101 (3 2020), 032330. Issue 3. https://doi.org/10.1103/PhysRevA.101.032330

[36] 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. https://doi.org/10.1145/3168822

[37] Andrew Steane. 1998. Quantum computing. *Rep. Prog. Phys.* 61, 2 (Feb. 1998), 117–173.

[38] 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 [quant-ph]

[39] Simon Thelen, Hila Safi, and Wolfgang Mauerer. 2024. Approximating under the Influence of Quantum Noise and Compute Power. In *Proceedings of WIHPQC@IEEE QCE*. http://arxiv.org/abs/2408.02287

[40] 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. https://doi.org/10.1016/j.tcs.2015.01.052 Fun with Algorithms.

[41] 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. https://doi.org/10.1145/3445814.3446706

[42] 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. https://doi.org/10.3390/electronics12163492