LIMDD: A Decision Diagram for Simulation of Quantum Computing Including Stabilizer States

Lieuwe Vinkhuijzen*
Leiden University, Leiden, The Netherlands

Tim Coopmans*
TU Delft, Delft, The Netherlands

David Elkouss
TU Delft, Delft, The Netherlands

Vedran Dunjko
Leiden University, Leiden, The Netherlands

Alfons Laarman
Leiden University, Leiden, The Netherlands

Abstract

Efficient methods for the representation of relevant quantum states and quantum operations are crucial for the simulation and optimization of quantum circuits. Decision diagrams (DDs), a well-studied data structure originally used to represent Boolean functions, have proven capable of capturing interesting aspects of quantum systems, but their limits are not well understood. In this work, we investigate and bridge the gap between existing DD-based structures and the stabilizer formalism, a well-studied method for simulating quantum circuits in the tractable regime. We first show that although DDs were suggested to succinctly represent important quantum states, they actually require exponential space for a subset of stabilizer states. To remedy this, we introduce a more powerful decision diagram variant, called Local Invertible Map-DD (LIMDD). We prove that the set of quantum states represented by poly-sized LIMDDs strictly contains the union of stabilizer states and other decision diagram variants. We also provide evidence that LIMDD-based simulation is capable of efficiently simulating some circuits for which both stabilizer-based and other DD-based methods require exponential time. By uniting two successful approaches, LIMDDs thus pave the way for fundamentally more powerful solutions for simulation and analysis of quantum computing.

2012 ACM Subject Classification Mathematics of computing → Decision diagrams

Keywords and phrases Simulation of Quantum Computing, Decision Diagrams

1 Introduction

Classical simulation of quantum computing is useful for circuit design [54] and studying noise resilience in the era of Noisy Intermediate-Scale Quantum (NISQ) computers [41]. Moreover, identifying classes of quantum circuits that are classically simulatable, helps in excluding regions where a quantum computational advantage cannot be obtained. For example, circuits containing only Clifford gates (a non-universal quantum gate set), using an all-zero initial state, only compute the so-called 'stabilizer states' and can be simulated in polynomial time [30, 3, 29]. Stabilizer states, and associated formalisms for expressing them, are fundamental to many quantum error correcting codes [29] and play a role in measurement-based quantum computation [42]. In fact, simulation of general quantum circuits is fixed-parameter tractable in the number of non-Clifford gates [13], a principle on which many modern simulators are

*These authors contributed equally.;

based [13, 11, 10, 33, 35, 36].

Another method for simulating universal quantum computation is based on (algebraic) decision diagrams (DDs) [4, 15, 7, 16, 44, 20, 27, 49, 50, 39, 55]. A DD is a directed acyclic graph (DAG) in which each path represents a quantum amplitude, enabling the succinct representation of many quantum states through the combinatorial nature of these paths. Various manipulation operations for DDs exist which implement any quantum gate operation in polynomial time in the size of the DD. Together with other DD operations that can be used for measurement, strong simulation is easily implemented using a DD data structure [39, 55]. Indeed, DD-based simulation was empirically shown to be competitive with state-of-the-art simulators [50, 55] and is used in several simulator implementations [51]. DDs and the stabilizer formalism are introduced in Section 2.

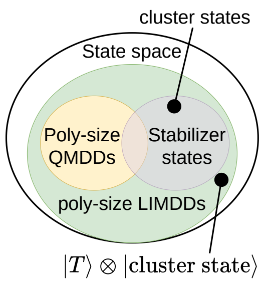


Figure 1 The set of stabilizer states and states represented as: poly-sized LIMDDs and QMDDs.

In this paper, we show that certain stabilizer states, called cluster states [14], yield exponentially large QMDDs, the currently most succinct version of DDs (see Section 4). In order to unite the strengths of DDs and the stabilizer formalism, in Section 3, we propose LIMDD: a new DD for quantum computing simulation using local invertible maps (LIMs). Specifically, LIMDDs eliminate the need to store multiple states which are equivalent up to LIMs, allowing more succinct DD representations. We prove that the set of quantum states that can be *represented* by poly-sized LIMDDs is larger than those that can be expressed in either the stabilizer formalism or a poly-sized QMDD. Figure 1 shows the resulting separation. In Section 5, we give procedures for analyzing and simulating quantum circuits using LIMDDs and conclude in Section 7 with evidence that LIMDD-based simulation can be powerful than modern techniques using low-rank stabilizer decomposition [10].

The workhorse behind LIMDDs is a novel algorithm which merges two DD nodes when they are 'isomorphic:' Two quantum states $|\varphi\rangle$ and $|\psi\rangle$ are isomorphic when there is a series of Pauli operators P_j and a complex nonzero number λ such that $|\varphi\rangle = \lambda P_n \otimes \cdots \otimes P_1 |\psi\rangle$. There is a plethora of work on investigating the effect of similar local operations in the context of stabilizer states [48, 24]; we emphasize that here we consider arbitrary quantum states $|\varphi\rangle, |\psi\rangle$.To find such an isomorphism, we compute (generators of) the stabilizer (sub)group $\text{Stab}(|\psi\rangle)$ of the state $|\psi\rangle$ represented by each DD node, and then we exploit the fact that the set of all such isomorphisms can be expressed as the coset $\pi \cdot \text{Stab}(|\psi\rangle)$ for some isomorphism π. To make the diagram canonical —an important property for realizing efficient manipulation operations [21]— our algorithm then chooses a "lexicographically smallest" element from a Pauli coset.

2 Preliminaries: decision diagrams and stabilizer states

The computational unit of quantum computers are quantum bits or qubits. A single-qubit state is a complex vector $(\alpha_0, \alpha_1)^T \in \mathbb{C}^2$ with norm 1, usually written in Dirac notation as $\alpha_0 |0\rangle + \alpha_1 |1\rangle$. Two quantum states which differ only by a complex multiple are considered equal. The joint state $|\varphi\rangle$ of n quantum bits can be written as

$$\sum_{x_1,x_2,\ldots,x_n\in\{0,1\}} f(x_1,x_2,\ldots,x_n)\,|x_1\rangle\otimes|x_2\rangle\otimes\cdots\otimes|x_n\rangle \tag{1}$$

for a function $f:\{0,1\}^n\to\mathbb{C}$, where $\otimes$ denotes the tensor product. An example two-qubit state is $(|0\rangle\otimes|0\rangle+i\,|1\rangle\otimes|1\rangle)/\sqrt{2}=\frac{1}{\sqrt{2}}(1,0,0,i)^T$. Alternatively to Equation 1, we can recursively describe an n-qubit quantum state $|\varphi\rangle$ for $n>1$ as

$$|\varphi\rangle=\alpha_0\,|0\rangle\otimes|\varphi_0\rangle+\alpha_1\,|1\rangle\otimes|\varphi_1\rangle\,,\text{where } |\varphi_0\rangle\,,|\varphi_1\rangle \text{ are } n\text{-1-qubit states and } \alpha_0,\alpha_1\in\mathbb{C}. \tag{2}$$

An Algebraic Decision Diagram (ADD) represents a function of the form $f\colon\{0,1\}^n\to\mathbb{C}$, and thus also a quantum state via Equation 1, see Figure 2 for an example. An ADD is a rooted directed acyclic graph (DAG), which has a leaf node for each unique value in the image of f, i.e., in $\{f(\vec{x})\mid\vec{x}\in\{0,1\}^n\}$. Each path from the root to a leaf visits nodes representing the variables x_1,x_2,x_3,x_4; one variable at each level of the diagram. The value $f(x_1,\ldots,x_n)$ is found by traversing such a path, following the *low edge* (dashed line) when $x_i=0$, and the *high edge* (solid line) when $x_i=1$; so, e.g., $f(1,1,1,0)=-i$ in Figure 2. Hence every node in an ADD, not only the root node, can be said to represent a function.

A *partial assignment* $(x_1=a_1,\ldots,x_k=a_k)$ to the variables induces a *subfunction* f_a, defined as $f_a(x_{k+1},\ldots,x_n)\triangleq f(a_1,\ldots,a_k,x_{k+1},\ldots,x_n)$. Two ADD nodes representing the same subfunction can be *merged*, i.e., one node is deleted and its incident edges are rerouted to the other. When all eligible nodes have been merged, an ADD is *reduced*. The nodes of a reduced ADD are in one-to-one correspondence with the unique *subfunctions* of f. An ADD is a *canonical* representation: a given function has exactly one reduced ADD. ADDs can represent both states and matrices, and there are algorithms which multiply a matrix with a vector in ADD form. An ADD can represent any quantum state, using exponentially many nodes in the worst case. A given quantum gate can be compiled into an ADD, thus allowing one to simulate any quantum circuit by repeatedly multiplying a gate's matrix with a state.

The Quantum Multi-valued Decision Diagram (QMDD) [39] improves on the ADD representation by also merging two nodes when they represent functions f,g that are related by $f=\lambda\cdot g$ for some $\lambda\in\mathbb{C}^*$. Figure 2 gives an example. Each edge is labelled with a weight. To read a value of f, traverse the QMDD from the root to the leaf just as in an ADD, and multiply the weights of all edges on that path. Like ADDs, QMDDs are a canonical representation and can be used to simulate any quantum circuit.

The single-qubit Pauli operators are 2×2 unitary matrices:

$$\mathbb{I}_2\triangleq\begin{pmatrix}1&0\\0&1\end{pmatrix},X\triangleq\begin{pmatrix}0&1\\1&0\end{pmatrix},Y\triangleq\begin{pmatrix}0&-i\\i&0\end{pmatrix},Z\triangleq\begin{pmatrix}1&0\\0&-1\end{pmatrix}$$

where i is the complex unit. An n-qubit operator of the form $P_n\otimes\cdots\otimes P_1$ is called a *Pauli string* if P_j are single-qubit Pauli operators. The n-qubit Pauli strings generate a nonabelian group $\textsc{Pauli}_n$ (under matrix multiplication), consisting of all operators of the form $\lambda P_n\otimes\cdots\otimes P_1$ with $\lambda\in\{\pm1,\pm i\}$. Note that the high indices are on the left, in keeping with the custom that the least significant (qu)bit is the first qubit, and the leftmost operator P_n acts on the most significant qubit. Pauli operators A,B either commute $(A\cdot B=B\cdot A)$ or anticommute $(A\cdot B=-B\cdot A)$.

In contrast to decision diagrams, the stabilizer formalism forms a subset of quantum computation that is efficiently simulatable. A stabilizer state on n qubits can be prepared from the state $|0\rangle^{\otimes n}$ by repeatedly applying any of the following gates (generators of the Clifford set):

$$H \triangleq \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, S \triangleq \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}, \text{CNOT} \triangleq \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}. \tag{3}$$

There exist $2^{\Theta(n^2)}$ stabilizer states on n qubits [3]; examples are $|00\rangle$ and $(|00\rangle + |11\rangle)/\sqrt{2}$. A strict subset of stabilizer states is the set of graph states [32]. The relationship between graph states and stabilizer states has been extensively investigated in, e.g., [47, 48] For an undirected graph $G = (V, E)$, the graph state $|G\rangle$ is the following state on n qubits, where Z_{jk} denotes the controlled Z-gate between qubits j and k.

$$\frac{1}{\sqrt{2}^n}\prod_{(j,k)\in E} Z_{jk}\left(|0\rangle + |1\rangle\right)^{\otimes n}, \text{ where e.g. } Z_{1,2} = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&-1\end{pmatrix} \text{ for } n = 2. \tag{4}$$

An n-qubit stabilizer state $|\varphi\rangle$ is uniquely specified by the set S of Pauli operators $A \in \text{PAULI}_n$ for which $A|\varphi\rangle = |\varphi\rangle$. This set S is an abelian group of 2^n elements, succinctly represented by n independent generators. Since each Pauli generator takes $\mathcal{O}(n)$ space to represent, an n-qubit stabilizer is represented by $\mathcal{O}(n^2)$ bits. Updating the stabilizer generating set after application of one of the gates from eq. (3) or a single-qubit computational-basis measurement can be done in polynomial time [30]. Also, we note that multiplying two n-qubit Pauli strings can be done in $O(n)$ time by using the property of the tensor product $\otimes$ that $(a \otimes b) \cdot (c \otimes d) = (a \cdot c) \otimes (b \cdot d)$.

Stabilizer-rank based methods [13, 11, 10, 33, 35, 36] extend this approach to families of Clifford circuits with arbitrary input states $|\varphi_n\rangle$, enabling the simulation of universal quantum computation in general [12]. By decomposing $|\varphi_n\rangle$ as linear combination of χ stabilizer states, the measurement outcome probabilities can be computed in time $\mathcal{O}(\chi \cdot \text{poly}(n))$, where the least χ is referred to as the *stabilizer rank*. Therefore, stabilizer-rank based methods are efficient for a family of input states $|\varphi_n\rangle$ with a stabilizer rank polynomially growing in n.

In this work we will also consider stabilizer groups of states which are not stabilizer states. In general, we will refer to an abelian subgroup of PAULI_n, not containing $-\mathbb{I}_2^{\otimes n}$, as an n-qubit *stabilizer subgroup*, which generally has $\leq n$ generators. Such objects are also studied in the context of simulating mixed states [6] and quantum error correction [29]. Examples of stabilizer subgroups are $\{\mathbb{I}_2\}$ for $|0\rangle + e^{i\pi/4}|1\rangle$, $\langle -Z\rangle$ for $|1\rangle$ and $\langle X \otimes X\rangle$ for $(|00\rangle + |11\rangle) + 2(|01\rangle + |10\rangle)$.

Any n-qubit Pauli string can (modulo factor $\in \{\pm 1, \pm i\}$) be written as $(X^{x_n}Z^{z_n}) \otimes \cdots \otimes (X^{x_1}Z^{z_1})$ for bits $x_j, z_j, 1 \leq j \leq n$. We can therefore write an n-qubit Pauli string P as length-$2n$ binary vector

$$(\underbrace{x_n, x_{n-1}, \ldots x_1}_{\text{X block}} \,|\, \underbrace{z_n, z_{n-1}, \ldots, z_1}_{\text{Z block}}),$$

where we added the horizontal bar ($|$) only to guide the eye. We will refer to such vectors as *check vectors*. For example $X \sim (1, 0)$ and $Z \otimes Y \sim (0, 1|1, 1)$ [3]. A set of k Pauli strings thus can be written as $2n \times k$ binary matrix, often called *check matrix*, e.g.

$$\begin{pmatrix} X & \otimes & X & \otimes & X \\ \mathbb{I}_2 & \otimes & Z & \otimes & Y \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 1 & | & 0 & 0 & 0 \\ 0 & 0 & 1 & | & 0 & 1 & 1 \end{pmatrix}.$$

This equivalence induces an ordering on Pauli strings following the lexicographic ordering on bit strings. For example, $X < Y$ because $(1|0) < (1|1)$ and $Z \otimes \mathbb{I}_2 < Z \otimes X$ because $(00|10) < (01|10)$. Furthermore, if P, Q are Pauli strings corresponding to binary vectors $\vec{x}^P, \vec{z}^P$ and $\vec{x}^Q, \vec{z}^Q$, then

$$P \cdot Q \propto \bigotimes_{j=1}^{n} \left(X^{x_j^P} Z^{z_j^P}\right)\left(X^{x_j^Q} Z^{z_j^Q}\right) = \bigotimes_{j=1}^{n} \left(X^{x_j^P \oplus x_j^Q} Z^{z_j^P \oplus z_j^Q}\right)$$

and therefore the group of n-qubit Pauli strings with multiplication (disregarding factors) is group isomorphic to the vector space $\{0,1\}^{2n}$ with bitwise addition (i.e., exclusive or; 'xor'). Consequently, many efficient algorithms for linear-algebra problems carry over to sets of Pauli strings. In particular, if $G = \{g_1, \ldots, g_k\}$ are length$-2n$ binary vectors (/ n-qubit Pauli strings) with $k \leq n$, then we can efficiently perform the following operations.

RREF: bring G into a reduced-row echelon form (RREF) using Gauss-Jordan elimination (both standard linear algebra notions) where each row (in check matrix form) has strictly more leading zeroes than the row above. The RREF is achievable by $O(k^2)$ row additions (/ multiplications modulo factor) and thus $O(k^2 \cdot n)$ time (see [46] for a similar algorithm). In the RREF, the first 1 after the leading zeroes in a row is called a 'pivot'.

Independent Set convert G to a (potentially smaller) independent set by performing the RREF procedure and discarding resulting all-zero rows.

Membership: determining whether a given a vector (/ Pauli string) h has a decomposition in elements of G. This task can be reduced to independence by first getting G^{RREF} by applying RREF, followed by adding h to G and performing the Independent-Set procedure. The result has $|G^{\text{RREF}}|$ rows if $h \in \langle G \rangle$, and $|G^{\text{RREF}}| + 1$ rows otherwise.

Intersection: determine all Pauli strings which, modulo a factor, are contained in both G_A and G_B, where G_A, G_B are generator sets for n-qubit stabilizer subgroups. This can be achieved using the Zassenhaus algorithm [1] in time $O(n^3)$.

Division remainder: given a vector h (/ Pauli string h), determine $h^{\text{rem}} := \min_{g \in \langle G \rangle}\{gh\}$ (minimum in the lexicographic ordering) where $\oplus$ denotes bitwise XOR (/ factor-discarding multiplication). We do so in the check matrix picture by bringing G into RREF, and then making the check vector of h contain as many zeroes as possible by adding rows from G:

```
1: for column index $j = 1$ to $2n$ do
2:     if $h_j = 1$ and $G$ has a row $g_i$ with its pivot at position $j$ then $h := h \oplus g_i$
```

The resulting h is h^{rem}. This algorithm's runtime is dominated by the RREF step; $O(n^3)$.

In this work, we will consider the group of n-qubit Pauli operators $\lambda P_n \otimes \ldots P_n$ for arbitrary $\lambda \in \mathbb{C} - \{0\}$, denoted as PAULILIM$_n$. Since each stabilizer $\lambda P \in$ PAULILIM$_n$ has factor $\lambda = \pm 1$ (follows from $(\lambda P)|\varphi\rangle = (\lambda P)^2 |\varphi\rangle = \lambda^2 \mathbb{I} |\varphi\rangle = |\varphi\rangle$, hence $\lambda^2 = 1$), the stabilizer subgroups in PAULILIM$_n$ are the same as in PAULI$_n$. As extension of the check matrix form to $A \in$ PAULILIM$_n$, we write $A = r \cdot e^{i\theta} \cdot P_n \otimes \ldots \otimes P_1$, for $r \in \mathbb{R}_{>0}$ and $\theta \in [0, 2\pi)$ and represent A by a length-$(2n+2)$ vector where the last entries store r and θ, e.g.:

$$\begin{pmatrix} 3X & \otimes & X & \otimes & X \\ -\frac{1}{2}i\mathbb{I}_2 & \otimes & Z & \otimes & Y \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 1 & | & 0 & 0 & 0 & | & 3 & 0 \\ 0 & 0 & 1 & | & 0 & 1 & 1 & | & \frac{1}{2} & \frac{3\pi}{2} \end{pmatrix}$$

where we used $3 = 3 \cdot e^{i \cdot 0}$ and $-\frac{1}{2}i = \frac{1}{2} \cdot e^{3\pi i/2}$. The ordering on real numbers induces a lexicographic ordering (from left to right) on such extended check vectors, for example $(1, 1, |0, 0|3, \frac{1}{2}) < (1, 1|1, 0|2, 0)$. Let us stress that the factor encoding (r, θ) is less significant than the Pauli string encoding $(x_n, \dots, x_1|z_n, \dots, z_1)$. As a consequence, we can greedily determine the minimum of two Pauli operators.

Finally, we emphasize that the algorithms above rely on row addition, which is a commutative operation. Since conventional (i.e., factor-respecting) multiplication of Pauli operators is not commutative, the algorithms above are not straightforwardly applicable to (nonabelian subgroups of) PAULILIM_n. (For abelian subgroups of PAULILIM_n, such as stabilizer subgroups [3], the algorithms still do work.) Fortunately, since Pauli strings either commute or anti-commute, row addition may only yield an factors up to the $\pm$ sign, not the resulting Pauli strings. This feature, combined with the stipulated order assigning least significance to the factor, enables us to invoke the algorithms above as subroutine, with postprocessing to obtain the correct factor. We will do so in Subsubsection 5.5.2–5.5.3.

3 The LIMDD data structure

Where QMDDs only merge nodes representing the same state up to a constant factor, the LIMDD data structure goes further by also merging nodes that are equivalent up to local operations, called Local Invertible Maps (LIMs) (see Definition 1). As a result, LIMDDs can be exponentially more succinct than QMDDs, including some stabilizer states (see Section 4). We will call nodes which are equivalent under LIMs, *(LIM-) isomorphic*. This definition generalizes SLOCC equivalence; in particular, if we choose the parameter G to be the linear group, then the two notions coincide (Appendix A of [22]) [8, 19].

Domains of variables	
$n, i, j, k \in \mathbb{N}$	indices
$v, w, u \in \text{NODE}$	LIMDD nodes
$e, f \in \text{EDGE}$	LIMDD edges
$L, M \in \text{DD}$	LIMDD(diagrams)
$\|\varphi\rangle, \|\psi\rangle \in \mathbb{C}^{\otimes n}$	n-qubit states
$\alpha, \beta, \lambda, \mu \in \mathbb{C}$	complex factors
$P, Q, R \in \text{PAULI}^{\otimes n}$	PAULI strings
$A, B \in \mathbb{C} \times \text{PAULI}^{\otimes n}$	Pauli LIMs

▶ **Definition 1** (LIM, Isomorphism). *An n-qubit Local Invertible Map (LIM) is an operator $\mathcal{O}$ of the form $\mathcal{O} = \lambda \mathcal{O}_n \otimes \dots \otimes \mathcal{O}_1$, where the matrices $\mathcal{O}_i$ are invertible 2×2 matrices and $\lambda \in \mathbb{C} \setminus \{0\}$. An* isomorphism *between two n-qubit quantum states $|\varphi\rangle, |\psi\rangle$ is a LIM $\mathcal{O}$ such that $\mathcal{O}|\varphi\rangle = |\psi\rangle$. If G is a group of 2×2 invertible matrices and if all $\mathcal{O}_i \in G$, then we say that $\mathcal{O}$ is a G-isomorphism and that $|\varphi\rangle$ is G-isomorphic to $|\psi\rangle$, denoted $|\varphi\rangle \simeq_G |\psi\rangle$.*

Before we give the formal definition of LIMDDs in Definition 2, we give a motivating example in Figure 2 which demonstrates how the use of isomorphisms can yield small diagrams for a four-qubit state. In the state's QMDD, the nodes c_3 and c_4 represent the two-qubit state vectors $|c_3\rangle = [1, 1, 1, -1]^\dagger$ and $|c_4\rangle = [1, -1, \omega, -\omega]^\dagger$. By noticing that these two vectors are related via $|c_4\rangle = (T \otimes Z)|c_3\rangle$, we may discard the node c_4 and store the isomorphism $T \otimes Z$ instead. In fact, all the nodes on the QMDD's third level (i.e., the grandchildren nodes of the root node) are related to each other through such isomorphisms. A similar reduction in size can be achieved at the QMDD's second level, by noticing that nodes c_1 and c_2 are related via $|c_2\rangle = Z \otimes I \otimes Z |c_1\rangle$. The resulting data structure is a LIMDD of only five nodes instead of ten. Section 4 shows that discarding isomorphic nodes sometimes leads to exponentially smaller diagrams, while the additional cost of storing the isomorphisms is only polynomial.

▶ **Definition 2** (G-LIMDD). *An n-G-LIMDD is a rooted, directed acyclic graph (DAG) which represents an n-qubit quantum state (or matrix). Formally, a G-LIMDD is a 6-tuple $(\textsc{Node} \cup \{\mathit{Leaf}\}, \mathsf{idx}, \mathsf{low}, \mathsf{high}, \mathsf{label}, e^r)$, where:*

- *$\textsc{Node}$ is a set of nodes with qubit indices $\mathsf{idx}(v) \in [n]$ for $v \in \textsc{Node}$;*
- *$\mathsf{low}, \mathsf{high} \colon \textsc{Node} \to \textsc{Node} \cup \{\mathit{Leaf}\}$ are the low and high edge functions;*
- *$\mathsf{label} \colon \mathsf{low} \cup \mathsf{high} \to G\text{-}LIM \cup \{0\}$ is a function labeling edges with LIMs or 0;*
- *a root edge e^r without source pointing to root node $r \in \textsc{Node}$;*
- *a unique leaf node Leaf (a sink) with label $\mathsf{idx}(s) = 0$ representing the number 1;*

Depending on context, we interpret $\mathsf{label}(\mathsf{low}(v))$ as a node v_0 or edge (v, v_0), etc. We define the semantics of a (non-leaf) node v and edge $e = (w, v)$ by overloading the Dirac notation:

$$|e\rangle \triangleq \begin{cases} \lambda \cdot (\mathcal{Q}_{\mathsf{idx}(v)} \otimes \cdots \otimes \mathcal{Q}_1) \cdot |v\rangle & \text{if } \mathsf{label}(e) = (\lambda, \mathcal{Q}_1, \ldots, \mathcal{Q}_{\mathsf{idx}(v)}) \\ [0, \ldots, 0]^\dagger & \text{if } \mathsf{label}(e) = 0 \end{cases}$$

$$|v\rangle \triangleq |0\rangle \otimes |\mathsf{low}(v)\rangle + |1\rangle \otimes |\mathsf{high}(v)\rangle$$

The coefficient $\langle x|e\rangle$ for bitstring $x \in \{0,1\}^n$ of a LIMDD with root edge e representing an n-qubit state $|e\rangle$ is read by traversing the LIMDD from top to bottom according to Definition 2 (i.e., pushing down the LIMs). It is best illustrated by example, e.g., reading the amplitude for 1111 in the LIMDD of Figure 2. The LIM on the root edge is the identity, so we can simply follow the 1-edge to get a new root edge with LIM $P_3 \otimes P_2 \otimes P_1 = Z \otimes \mathbb{I} \otimes Z$ to d_1. Since the most significant operator (P_3) is a Z, we multiply the LIM on the 1-edge of d_1 (which has $\mathsf{idx}(d_1) = 3$) with -1 and the remainder of the LIM ($\mathbb{I} \otimes Z$), yielding $-\omega T \otimes \mathbb{I}$, etc. Eventually the leaf is reached, when only a factor remains (= the sought amplitude). If we would encounter an X (or Y) as P_3, we would also have to switch the high (1) and the low (0) edge, thus taking the 0- instead of 1-edge (and multiply by $-i$ for $P_3 = Y$). For the choice $G = \textsc{Pauli}$, this is formalized in the $\textsc{follow}()$ procedure given in Section 5.

Let us now summarize the representation and manipulation capabilities of the LIMDD data structure. A G-LIMDD is exact and universal, i.e., for each n-qubit quantum state $|\varphi\rangle$ there

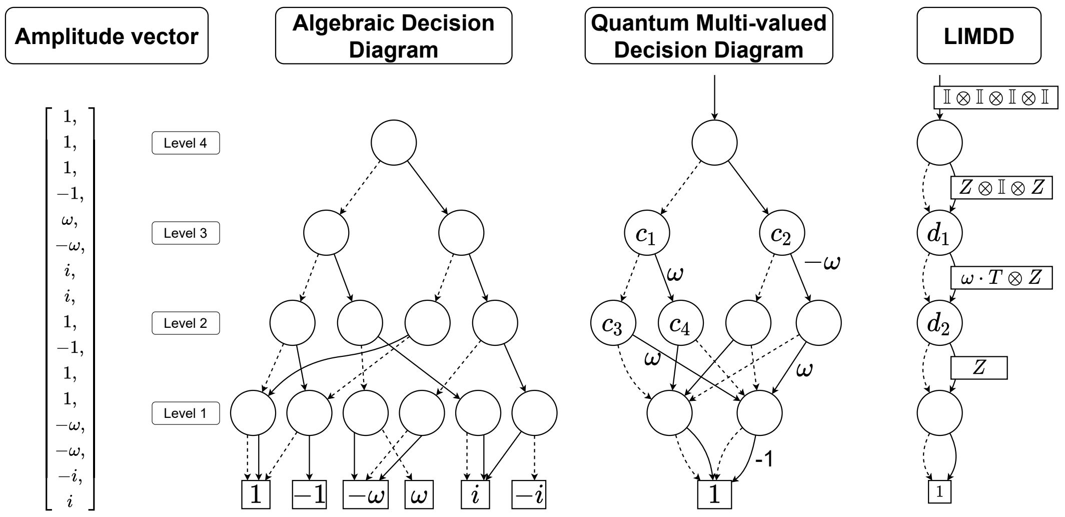

Amplitude vector
Algebraic Decision Diagram
Quantum Multi-valued Decision Diagram
LIMDD
1, 1, 1, −1, ω, −ω, i, i, 1, −1, 1, 1, −ω, −ω, −i, i
Level 4
Level 3
Level 2
Level 1
1
−1
−ω
ω
i
−i
c1
c2
c3
c4
ω
−ω
ω
ω
-1
1
I⊗I⊗I⊗I
Z⊗I⊗Z
d1
ω·T⊗Z
d2
Z
1

Figure 2 A four-qubit quantum state shown as: an amplitude vector (left), an ADD, a QMDD and a LIMDD (right). Diagram nodes are horizontally ordered in 'levels' with qubit indices $4, 3, 2, 1$.

is a LIMDD with root edge e such $|e\rangle = |\varphi\rangle$, for any choice of parameter G. In particular, a G-LIMDD with $G = \{\mathbb{I}\}$ captures all QMDDs by definition. As all groups G contain the identity operator $\mathbb{I}$, the universality of LIMDDs follows from the universality of QMDDs. Furthermore, for the choice $G =$ Pauli, the states that LIMDDs can represent using polynomial space include all stabilizer states, which is a feature that QMDDs do not posses, as shown in Section 4. Finally, Pauli-LIMDDs can also *manipulate* and measure quantum states, thereby enabling the simulation of quantum circuits, as shown in Section 5.4. For many operations, we show that the manipulation is also efficient, i.e., it takes polynomial time in the size of the LIMDD representation of the quantum state/circuit. Specifically, LIMDDs are often faster than QMDDs, and never slower than a multiplicative factor $O(n^3)$.

From here on, we will focus on the choice $G =$ PAULI, omitting the prefix PAULI- in front of LIMDD unless it is clear that we mean otherwise, and hence write $\simeq$ to mean $\simeq_{\text{Pauli}}$.

In general, there are many different PAULI-LIMDDs which represent a given quantum state. By imposing a small number of constraints on the diagram, listed in Definition 3 and visualized in Figure 3, we ensure that every quantum state is represented by a unique '*reduced*' LIMDD. Unique representation, or canonicity, is a useful property of decision diagrams. In the first place, it allows for circuit analysis and simplification [16, 39], by facilitating efficient manipulation operations. In the second place, a reduced diagram is smaller than an unreduced diagram because it merges nodes with the same semantics. For instance, LIMDDs allow all states in the same $\simeq$ equivalence class to be merged. The algorithms for quantum circuit simulation in Section 5 ensure that all intermediate LIMDDs are reduced.

▶ **Definition 3** (Reduced LIMDD). *A PAULI-LIMDD is* reduced *when it satisfies the following constraints. It is* semi-reduced *if it satisfies all constraints except high determinism.*

1. ***Merge:*** *No two nodes are identical: We say two nodes v, w are identical if $\mathsf{low}(v) = \mathsf{low}(w)$, $\mathsf{high}(v) = \mathsf{high}(w)$, $\mathsf{label}(\mathsf{low}(v)) = \mathsf{label}(\mathsf{low}(w))$, $\mathsf{label}(\mathsf{high}(v)) = \mathsf{label}(\mathsf{high}(w))$.*
2. ***(Zero) Edge:*** *Any edge $(v, w) \in \mathsf{high} \cup \mathsf{low}$ has $\mathsf{idx}(v) = \mathsf{idx}(w)+1$, and if $\mathsf{label}(v, w) = 0$, then both edges point to the same node, i.e., $\mathsf{high}(v) = \mathsf{low}(v) = w$.*
3. ***Low Precedence:*** *For each internal node v, we have $\mathsf{low}(v) \preccurlyeq \mathsf{high}(v)$, where $\preccurlyeq$ is a total order on the nodes of the diagram.*
4. ***Low Factoring:*** *The label on every low edge to a node v is the identity $\mathbb{I}_2^{\otimes \mathsf{idx}(v)}$.*
5. ***High Determinism:*** *The label on the high edge of any node v is $B_{\text{high}} = \mathsf{HighLabel}(v)$, where $\mathsf{HighLabel}$ is a function that takes as input a semi-reduced n-PAULI-LIMDD node v, and outputs an $(n-1)$-PAULI-LIM B_{high} satisfying $|v\rangle \simeq_{\text{PAULI}} |0\rangle\,|\mathsf{low}(v)\rangle + |1\rangle \otimes B_{\text{high}}\,|\mathsf{high}(v)\rangle$. Moreover, for any other semi-reduced node w with $|v\rangle \simeq_{\text{PAULI}} |w\rangle$, it returns $\mathsf{HighLabel}(w) = B_{\text{high}}$. In other words, $\mathsf{HighLabel}$ is constant within an isomorphism class.*

A few observations can be made about the above definition:

O1 There is no reduced LIMDD for the 0-vector, because any low edge must be labeled with $\mathbb{I}_2^{\otimes n}$. This is not a problem for us, since the 0-vector is not a quantum state.

O2 To represent a state like $|0\rangle \otimes |\varphi\rangle$, there is a choice between $|0\rangle \otimes |\varphi\rangle$ and $(X \otimes \mathbb{I})\,|1\rangle \otimes |\varphi\rangle$. The low factoring rule forces us to take the $\mathbb{I}$ label on the low edge, so this gives a node of the form $|0\rangle \otimes |\varphi\rangle$. Therefore, the high edge *must* be labeled with 0. Since, by the zero edges rule, the high edge then points to the same node as the low edge the low precedence rule vacuously holds for such states.

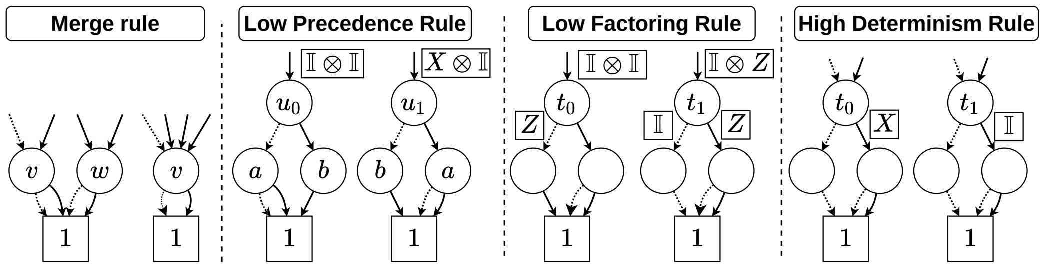


Figure 3 Illustration of the reduction rules in Definition 3. In each case, the left and right LIMDDs represent the same state, but the left LIMDD violates a reduction rule, while the right LIMDD satisfies that rule. The Merge rule regards the merging of two identical nodes v and w. Low Precedence determines which child is the low child, and which is the high child according to $\preccurlyeq$. Low Factoring ensures that the low edge is always labeled with $\mathbb{I}$. High Determinism ensures that the label on high edges is chosen canonically.

O3 The definition of reduced LIMDD cannot always be applied to G-LIMDD when G is a subgroup of the Pauli group; in particular, such a diagram may not be universal. This is because the low precedence rule requires that $v_0 \preccurlyeq v_1$ for every node, so if G is a group which does not contain the X, then no reduced G-LIMDD represents a state $|0\rangle|v_0\rangle + |1\rangle|v_1\rangle$ where $v_1 \preccurlyeq v_0$.

O4 While the literature on other decision diagrams [4, 15, 26] often considers a "redundant test" rule that allows edges to skip multiple levels, we omit this reduction for the sake of simplicity, because it offers only linear improvements in the best case and complicates correctness proofs (see, e.g., [5, Lemma 1]). There is however no fundamental reason which would prevent the addition of a similar redundancy reduction.

Lemma 5 shows that LIMDDs are canonical, in the sense that its nodes uniquely represent equivalence classes under $\simeq$ as expressed in Corollary 4. This does not mean that the root edge of a LIMDD canonically represents a quantum state. We instead show in Section 5.1 how to check whether two root edges represent the same state.

▶ **Corollary 4** (of Lemma 5). *Each equivalence class under $\simeq$, has a unique representative reduced LIMDD node v, which follows from Lemma 5 and the transitivity of $\simeq$.*

▶ **Lemma 5** (Node canonicity). *For each n-qubit state vector $|\varphi\rangle$, there exists a unique reduced Pauli-LIMDD L with root node v_L such that $|v_L\rangle \simeq |\varphi\rangle$.*

Proof. We use induction on on the number of qubits n to show universality (the existence of an isomorphic LIMDD node) and uniqueness (canonicity).

Base case. If $n = 0$, then $|\varphi\rangle$ is a complex number λ. A reduced Pauli-LIMDD for this state is the leaf node representing the scalar 1. To show it is unique, consider that nodes v other than the leaf have an $\mathsf{idx}(v) > 0$, by the edges rule, and hence represent multi-qubit states. Since the leaf node itself is defined to be unique, the merge rule is not needed and canonicity follows. Finally, $|\varphi\rangle$ is represented by root edge $\xrightarrow{\lambda}\textcircled{1}$.

Inductive case. Suppose $n > 0$. We first show existence, and then show uniqueness.

We use the unique expansion of $|\varphi\rangle$ as $|\varphi\rangle = |0\rangle \otimes |\varphi_0\rangle + |1\rangle \otimes |\varphi_1\rangle$ where $|\varphi_0\rangle$ and $|\varphi_1\rangle$ are

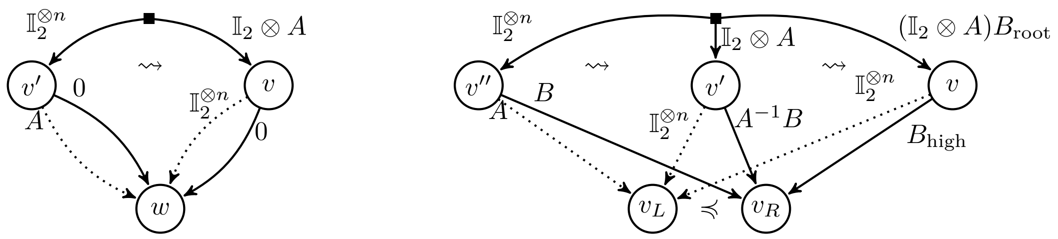


Figure 4 Reduced node construction in case $|\varphi_1\rangle = 0$ (left), and $|\varphi_0\rangle, |\varphi_0\rangle \neq 0$ and $v_L \preccurlyeq v_R$ (right). For cases $|\varphi_0\rangle = 0$ and $v_R \preccurlyeq v_L$, we take instead root edge $X \otimes A$ and swap low/high edges. The black square (■) signifies a unique quantum state (all root edge represent this one state).

either $(n-1)$-qubit state vectors, or the all-zero vector. We distinguish three cases based on whether $|\varphi_0\rangle, |\varphi_1\rangle = 0$.

Case $|\varphi_0\rangle, |\varphi_1\rangle = 0$: This case is ruled out because $|\varphi\rangle \neq 0$.

Case $|\varphi_0\rangle = 0$ or $|\varphi_1\rangle = 0$: In case $|\varphi_0\rangle \neq 0$, by the induction hypothesis, there exists a Pauli-LIMDD with root node w satisfying $|w\rangle \simeq |\varphi_0\rangle$. By definition of $\simeq$, there exists an n-qubit Pauli isomorphism A such that $|\varphi_0\rangle = A|w\rangle$. We construct the following reduced Pauli-LIMDD for $|\varphi\rangle$: $\textcircled{w} \xleftarrow{I} \textcircled{v} \xrightarrow{0} \textcircled{w}$. In case $|\varphi_1\rangle \neq 0$, we do the same for root $|w\rangle \simeq |\varphi_1\rangle = A|w\rangle$. In both cases, it is easy to check that the root node is reduced. Also in both cases, we have $|\varphi\rangle \simeq |v\rangle$ because either $|\varphi\rangle = \mathbb{I}_2 \otimes A|v\rangle$ or $|\varphi\rangle = X \otimes A|v\rangle$ as illustrated in Figure 4 (left).

Case $|\varphi_0\rangle, |\varphi_1\rangle \neq 0$: By the induction hypothesis, there exist Pauli-LIMDDs L and R with root nodes $|v_L\rangle \simeq |\varphi_0\rangle$ and $|v_R\rangle \simeq |\varphi_1\rangle$.[1] By definition of $\simeq$, there exist n-qubit Pauli isomorphisms A and B such that $|\varphi_0\rangle = A|v_L\rangle$ and $|\varphi_1\rangle = B|v_R\rangle$. In case $v_L \preccurlyeq v_R$, we construct the following reduced Pauli-LIMDD for $|\varphi\rangle$: the root node is $\textcircled{v_L} \xleftarrow{\mathbb{I}} \textcircled{v} \xrightarrow{E} \textcircled{v_R}$, where E is the LIM computed by HighLabel($\textcircled{v_L} \xleftarrow{\mathbb{I}} \bigcirc \xrightarrow{A^{-1}B} \textcircled{v_R}$) . Otherwise, if $v_R \preccurlyeq v_L$, then we construct the following reduced Pauli-LIMDD for $|\varphi\rangle$: the root node is $\textcircled{v_R} \xleftarrow{I} \textcircled{v} \xrightarrow{F} \textcircled{v_L}$, where $F =$ HighLabel($\textcircled{v_L} \xleftarrow{\mathbb{I}} \bigcirc \xrightarrow{B^{-1}A} \textcircled{v_R}$). It is straightforward to check that, in both cases, this Pauli-LIMDD is reduced. Moreover, $|v\rangle$ isomorphic to $|\varphi\rangle$ as illustrated in Figure 4 (right).

To show uniqueness, let L and M be reduced LIMDDs (root nodes v_L, v_M) such that $|v_L\rangle \simeq |\varphi\rangle \simeq |v_M\rangle$. Expanding the semantics of v_L and v_M, this implies there exists a Pauli isomorphism $\lambda P_{\text{top}} \otimes P_{\text{rest}} \neq 0$, where P_{top} is a single-qubit Pauli and P_{rest} an $(n-1)$-qubit Pauli isomorphism, such that

$$\lambda P_{\text{top}} \otimes P_{\text{rest}}(|0\rangle \otimes A_L|v_L^0\rangle + |1\rangle \otimes B_L|v_L^1\rangle) = |0\rangle \otimes A_M|v_M^0\rangle + |1\rangle \otimes B_M|v_M^1\rangle. \quad (5)$$

We distinguish two cases from here on: where $P_{\text{top}} \in \{\mathbb{1}, Z\}$ or $P_{\text{top}} \in \{X, Y\}$.

Case $P_{\text{top}} = I, Z$. If $P_{\text{top}} = \left[\begin{smallmatrix} 1 & 0 \\ 0 & z \end{smallmatrix}\right]$ for $z \in \{1, -1\}$, then Equation 5 gives:

$$\lambda P_{\text{rest}} A_L|v_L^0\rangle = A_M|v_M^0\rangle \qquad \text{and} \qquad z\lambda P_{\text{rest}} B_L|v_L^1\rangle = B_M|v_M^1\rangle \quad (6)$$

[1]Note that the induction hypothesis implies a 'local' reduction of LIMDDs L and R, but not automatically a reduction of their union. For instance, L might contain a node w and R a node w such that $v \simeq w$. While the other reduction rules ensure that v and w will be structurally the same, the induction hypothesis only applies the merge rule L and M in isolation, leaving two separate identical nodes v, w. We can solve this by applying merge on the union of nodes in L and M, to merge any equivalent nodes.

By low factoring, we have $A_L = A_M = \mathbb{1}$, so we obtain $\lambda P_{\text{rest}} |v_L^0\rangle = |v_M^0\rangle$. Hence $|v_L^0\rangle$ is isomorphic with $|v_M^0\rangle$, so by induction hypothesis and Corollary 4, we have $v_L^0 = v_M^0$. We now show that also $v_L = v_M$ by considering two cases.

$B_L \neq 0$ and $B_M \neq 0$: then $z\lambda P_{\text{rest}} B_L |v_L^1\rangle = B_M |v_M^1\rangle$, so the nodes v_L^1 and v_M^1 represent isomorphic states, so by the induction hypothesis and Corollary 4 we have $v_L^1 = v_M^1$. We already noticed by the low factoring rule that v_L and v_M have $\mathbb{I}$ as low edge label. By the high edge rule, their high edge labels are HighLabel(v_L) and HighLabel(v_M), and since the reduced LIMDDs L and M also satisfy low precedence and edge rules and $|v_L\rangle \simeq |v_M\rangle$, we have HighLabel$(v_M)$ = HighLabel(v_L) by definition of HighLabel.

$B_L = 0$ or $B_M = 0$: In case $B_L = 0$, we see from Equation 6 that $0 = B_M |v_M^1\rangle$. Since the state vector $|v_M^1\rangle \neq 0$ by O1, it follows that $B_M = 0$. Otherwise, if $B_M = 0$, then Equation 6 yields $z\lambda P_{\text{rest}} B_L |v_L^1\rangle = 0$. We have $z\lambda \neq 0$, $P_{\text{rest}} \neq 0$ by definition, and $|v_L^1\rangle \neq 0$ by O1. Therefore $B_L = 0$. In both cases, $B_L = B_M$.

We conclude that in both cases v_L and v_M have the same children and the same edge labels, so they are identical by the merge rule.

Case $P_{\text{top}} = X, Y$. If $P_{\text{top}} = \left[\begin{smallmatrix} 0 & z^* \\ z & 0 \end{smallmatrix}\right]$ for $z \in \{1, i\}$, then Equation 5 gives:

$$\lambda z P_{\text{rest}} A_L |v_L^0\rangle = B_M |v_M^1\rangle \qquad \text{and} \qquad \lambda z^* P_{\text{rest}} B_L |v_L^1\rangle = A_M |v_M^0\rangle .$$

By low factoring, $A_L = A_M = \mathbb{1}$, so we obtain $z\lambda P_{\text{rest}} |v_L^0\rangle = B_M |v_M^1\rangle$ and $\lambda z^* P_{\text{rest}} B_L |v_L^1\rangle = |v_M^0\rangle$. To show that $v_L = v_M$, we consider two cases.

$B_L \neq 0$ and $B_M \neq 0$: we find $|v_L^0\rangle \simeq |v_M^1\rangle$ and $|v_L^1\rangle \simeq |v_M^0\rangle$, so by the induction hypothesis, $v_L^0 = v_M^1$ and $v_L^1 = v_M^0$. By low precedence, it must be that $v_L^1 = v_M^1 = v_L^0 = v_M^0$. Now use high determinism to infer that $B_L = B_M$ as in the $P_{\text{top}} = I, Z$ case.

$B_L = 0$ or $B_M = 0$: This case leads to a contradiction and thus cannot occur. B_L cannot be zero, because then $|v_M^0\rangle$ is the all-zero vector, which we excluded by O1. The other case: $B_M = 0$, then it must be that P_{rest} is zero, hence $|v_M^0\rangle$ is the all-zero vector, which is again excluded.

We conclude that v_L and v_M have the same children and the same edge labels for all choices of P_{top}, so they are identical by the merge rule. ◀

4 LIMDDs are exponentially more succinct than QMDDs

In this section, we show that LIMDDs can be exponentially more succinct than the union of QMDDs and stabilizer states (Theorem 6). Namely, we show that polynomial-sized G-LIMDDs can represent stabilizer states, using $G = \text{Pauli}$, (Theorem 7) whereas QMDDs require exponential space to represent cluster states (Lemma 8). In Appendix A and Appendix B, we show that LIMDDs retain this exponential advantage even when we use the parameter $G = \langle Z \rangle$ or $G = \langle X \rangle$.[2] We emphasize that this section regards *representation* of quantum states; in Section 5, we show that by using LIMDDs to represent quantum states, we can *simulate* quantum circuits.

[2] Note that the proofs in this section do not rely on the specialized definition of reduced LIMDDs, but only on the parametrized Definition 2.

Figure 5 visualizes the results from this section. The boxes with $\langle Z\rangle$-LIMDD denote the LIMDD with parameter $G = \langle Z\rangle$, i.e., LIMDDs where the labels on edges are all of the form $\lambda P_n \otimes \cdots \otimes P_1$ where $P_j \in \{\mathbb{I}, Z\}$. Although of course every $\langle Z\rangle$-LIMDD is a Pauli-LIMDD, this parameterization is interesting in its own right because it requires only polynomial size to represent any graph state. Similarly, the $\langle X\rangle$-LIMDD can succinctly represent a set of stabilizer states we call "XOR states." Appendix B proves these results, and proves that QMDDs require exponential size for all these states.

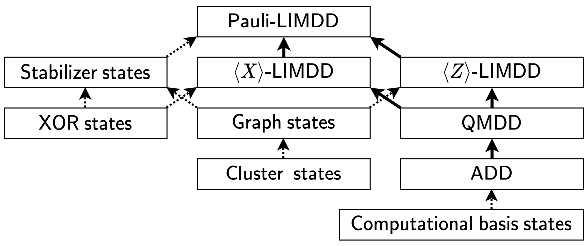


Figure 5 Relations between classes of quantum states and families of polynomial-size decision diagrams. Every arrow denotes a strict inclusion of sets; in particular, each solid arrow $D_1 \to D_2$ denotes an exponential separation between two decision diagram families, i.e., some quantum states have polynomial-size diagrams of type D_2, but have only exponential-size diagrams of type D_1. The dotted arrows represent inclusion.

▶ **Theorem 6** (Exponential separation between Pauli-LIMDD versus QMDD union stabilizer states). *The set of quantum states represented by polynomial-size Pauli-LIMDDs is a strict superset of the union of stabilizer states and polynomial-size QMDDs.*

Proof. Each n-qubit stabilizer state can be represented by a Pauli-LIMDD of n nodes (Theorem 7). Moreover, if a state has a polynomial-size QMDD, it also has a polynomial-size LIMDD, due to our earlier remark that a QMDD can be seen as a G-LIMDD with $G = \{\mathbb{I}\}$, i.e., each label is of the form $\lambda\mathbb{I}$ with $\lambda \in \mathbb{C}$. To show that polynomial-size Pauli-LIMDDs represent strictly more than these two classes of states, consider $|\varphi\rangle := |T\rangle \otimes |G_n\rangle$, where $|T\rangle = |0\rangle + e^{i\pi/4}|1\rangle$, and where $|G_n\rangle$ is the graph state on the $n \times n$ grid. We note that $|\varphi\rangle$ is not a stabilizer state, because each computational-basis coefficient of a stabilizer state is of the form $z \cdot 1/\sqrt{2}^k$ for $z \in \{\pm 1, \pm i\}$ and some integer $k \geq 1$ [48], while $\langle 1| \otimes \langle 0|^{\otimes n} |\varphi\rangle = e^{i\pi/4} \cdot \frac{1}{\sqrt{2}}^n$ is not of this form. Moreover, its canonical QMDD is a root node $v_G \xleftarrow{1} \bigcirc \xrightarrow{e^{i\pi/4}} v_G$ where v_G is the root node of the QMDD for $|G_n\rangle$, which has exponential size (Lemma 8). In contrast, the reduced Pauli-LIMDD for $|G_n\rangle$ (with root node w_G) has n nodes (because $|G_n\rangle$ is a stabilizer state), and hence a polynomial-size Pauli-LIMDD for $|T\rangle \otimes |G_n\rangle$ has root node with $v_G \xleftarrow{\mathbb{I}} \bigcirc \xrightarrow{e^{i\pi/4}\mathbb{I}} v_G$. ◀

Now we give the two statements leading up to Theorem 6: Pauli-LIMDDs represent stabilizer states succinctly (Theorem 7) but QMDDs representing stabilizer states are necessarily large (Lemma 8). In this theorem, by a Tower LIMDD we mean a LIMDD which has a single node on each level.

▶ **Theorem 7** (Tower Pauli-LIMDDs are stabilizer states). *Let $n > 0$. Each n-qubit stabilizer state is represented by a reduced Tower Pauli-LIMDD on n nodes with high edge label factors $\in \{0, \pm 1, \pm i\}$. Conversely, every such LIMDD represents a stabilizer state.*

Proof sketch. We sketch here why each stabilizer state is represented by a reduced Tower Pauli-LIMDD and give a full proof in Theorem 30. Let $|\psi\rangle$ be a stabilizer state. If $|\psi\rangle = |x\rangle |\psi'\rangle$ for some $x \in \{0, 1\}$ and $|\psi'\rangle$, then $|\psi'\rangle$ is a stabilizer state and it is represented by a Tower

Pauli-LIMDD which has a root node with a low edge label $\mathbb{I}$, high edge label 0 and root edge labelled $X^x \otimes \mathbb{I}$, to the root node of the Tower Pauli-LIMDD of $|\psi'\rangle$. Otherwise, $|\psi\rangle \propto |0\rangle|\psi_0\rangle + |1\rangle|\psi_1\rangle$, where both $|\psi_0\rangle$ and $|\psi_1\rangle$ are stabilizer states. Moreover, if $|\psi\rangle$ is a stabilizer state, there is always a set of single-qubit Pauli gates $P_1, \dots, P_n$ and a $\lambda \in \{\pm 1, \pm i\}$ such that $|\psi_1\rangle = \lambda P_n \otimes \dots \otimes P_1 |\psi_0\rangle$. That is, in our terminology, the states $|\psi_0\rangle$ and $|\psi_1\rangle$ are *isomorphic*. Hence $|\psi\rangle$ can be written as

$$|\psi\rangle = |0\rangle|\psi_0\rangle + \lambda|1\rangle \otimes (P_n \otimes \dots \otimes P_1 |\psi_0\rangle) \tag{7}$$

This expression suggests the following representation as a Tower Pauli-LIMDD: the root node represents $|\psi\rangle$, both its outgoing edges point to a node representing $|\psi_0\rangle$, and its high edge is labeled with the isomorphism $\lambda P_n \otimes \dots \otimes P_1$; this strategy is then applied recursively to $|\psi_0\rangle$ and its two subfunctions. This procedure yields a semi-reduced Tower-Pauli-LIMDD, which can be made reduced by making all high labels canonical, from bottom to top. ◀

▶ **Lemma 8.** *Denote by $|G_n\rangle$ the graph state on the $n \times n$ lattice. Each QMDD representing the cluster state $|G_n\rangle$ has at least $2^{\lfloor n/12 \rfloor}$ nodes.*

Proof sketch. Consider a partition of the vertices of the $n \times n$ lattice into two sets S and T of size $\frac{1}{2}n^2$, corresponding to the first $\frac{1}{2}n^2$ qubits under some variable order. Then there are at least $\lfloor n/3 \rfloor$ vertices in S that are adjacent to a vertex in T [37, Th. 11]. Because the degree of the nodes is small, many vertices on this boundary influence the amplitude function independently of one another. From this independence, it follows that, for any variable order, the partial assignments $\vec{a} \in \{0,1\}^{\frac{1}{2}n^2}$ induce $2^{\lfloor n/12 \rfloor}$ different subfunctions $f_{\vec{a}}$, where $f\colon \{0,1\}^{n^2} \to \mathbb{C}$ is the amplitude function of $|G_n\rangle$. The lemma follows by noting that a QMDD has a single node per unique subfunction modulo phase. For details see Appendix A. ◀

Finally, we note an exponential separation between QMDDs and ADDs. Although we do believe this fact may be known in the decision diagram community, to the best of our knowledge this separation does not appear in the literature.

▶ **Theorem 9.** *There is an infinite family of quantum states $\{|\varphi_n\rangle_n\}_n$ such that every ADD needs $\Theta(2^n)$ nodes to store $|\varphi_n\rangle$, but every QMDD needs only $\Theta(n)$ nodes.*

Proof. The family of states is

$$|\varphi_n\rangle = (|0\rangle + e^{i\pi}|1\rangle) \otimes (|0\rangle + e^{i\pi 2^{-1}}|1\rangle) \otimes \dots \otimes (|0\rangle + e^{i\pi 2^{-n+1}}|1\rangle),$$

which is a product state and can thus be represented by a QMDD on n nodes. In contrast, the computational-basis amplitudes are $\langle x|\varphi_n\rangle = e^{i\pi x 2^{-n}}/\sqrt{2^n}$ for $x \in \{0, 1, \dots, 2^n - 1\}$ in binary notation, and therefore no two $|x\rangle$ share the same amplitude, resulting in 2^n leaves of the ADD. ◀

5 Simulating Quantum Circuits with Pauli-LIMDDs

In this section, we give all the algorithms that are necessary to analyze and simulate a quantum circuit with Pauli-LIMDDs (we will simply say LIMDD from now on). We provide algorithms for dedicated gates, but also an algorithm to apply any multi-qubit gate that is represented as a LIMDD. We also give the subroutine MAKEEDGE which is used to keep the diagram reduced throughout the computation, thus preventing the creation of redundant nodes, keeping the diagram small and canonical. The procedure is the counterpart of 'MakeNode' used in other DD definitions for that purpose. Table 1 provides an overview of the LIMDD algorithms and their complexities compared to ADDs and QMDDs (an unequal comparison due to the exponential differences in conciseness shown in Section 4).

Table 1 Complexity of currently *best-known algorithm* for applying specific operations, in terms of the size of the input diagram size m (i.e., the number of nodes in the DD) and the number of qubits n. Although addition of quantum states is not, strictly speaking, a quantum operation, we include it because it is a subroutine of gate application. Although the table seems to suggest that ADD is faster than QMDD and LIMDD, we emphasize that QMDD is never slower than ADD, and LIMDD is never slower than QMDD modulo a multiplicative factor $O(n^3)$ due to calls to MAKEEDGE. The discrepancy appears because runtime is given as a function of decision diagram size, and ADDs/ QMDDs are sometimes exponentially larger than LIMDDs (see Section 4). Note that several of the LIMDD algorithms invoke MAKEEDGE and therefore inherit its cubic complexity (as a factor).

Operation \ input:	**ADD**	**QMDD**	**LIMDD**	**Section**
Checking state equality	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n^3)$	Section 5.1
Single $\|0\rangle$ / $\|1\rangle$-basis measurement	$\mathcal{O}(m)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$	Section 5.2
Single Pauli gate	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	Section 5.3
Single Hadamard gate / ADD()	$\mathcal{O}(m^2)$	$\mathcal{O}(2^n)$ note[3]	$\mathcal{O}(n^3 2^n)$ note[3]	Section 5.3
Clifford gate on stabilizer state	$\mathcal{O}(2^n)$	$\mathcal{O}(2^n)$	$\mathcal{O}(n^4)$	Section 5.3
Multi-qubit gate	$\mathcal{O}(4^n)$ note[4]	$\mathcal{O}(4^n)$ note[4]	$\mathcal{O}(n^3 4^n)$ note[4]	Section 5.4
MakeNode / MAKEEDGE (LIMDD)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n^3)$	Section 5.5

Our algorithms will use the notation in Table 2 to easily navigate and construct diagrams. The notation $\xrightarrow{A}\textcircled{v}$ creates an edge to an existing node v, labeled with a LIM A. If v_0 and v_1 are two existing nodes, then the notation $\textcircled{v_0}\xleftarrow{A}\bigcirc\xrightarrow{B}\textcircled{v_1}$ creates a new (not-necessarily-reduced) node whose left edge is $\xrightarrow{A}\textcircled{v_0}$ and whose right edge is $\xrightarrow{B}\textcircled{v_1}$. LIMs are decomposed using $A = \lambda_A P_n \otimes P'$. Here $\lambda_A \in \mathbb{C}$ is a non-zero scalar, and the matrices satisfy $P_j \in \{\mathbb{I}, X, Z, Y\}$ for every j. Finally, the $\textsc{follow}_b(\xrightarrow{\lambda P_n \otimes P'}\textcircled{v})$ notation allows us to *semantically* follow low and high edges, by applying P_n on qubit $\mathsf{idx}(v) = n$ and returning either $\mathsf{low}(v)$ or $\mathsf{high}(v)$ multiplied by $\lambda_A P'$. Specifically, if e is an edge with $|e\rangle = |0\rangle|\varphi_0\rangle + |1\rangle|\varphi_1\rangle$, then $\textsc{follow}_b(e)$ denotes an edge f satisfying $|f\rangle = |\varphi_b\rangle$. The amplitude of basis state $|1111\rangle$ for the LIMDD root edge e in Figure 2 is computed by taking $|\textsc{follow}_{1111}(e)\rangle = |\xrightarrow{\omega e^{\frac{\pi}{4}i}}\textcircled{1}\rangle = \omega e^{\frac{\pi}{4}i}$.

All LIMDD operations discussed in this section return reduced LIMDDs by creating edges and nodes using a dedicated MAKEEDGE operation. The complexity of computing LIMs to find a canonical representative for each isomorphism equivalence class resides in this function. We will therefore defer the treatment of this function to the end of this section. For now, the reader can assume a trivial (non-reducing) implementation of the functions as given in Algorithm 1. Like other DD structures, LIMDDs require a unique table (a set) to uniquely store canonical representatives.

[3]Only exponential when the ADD representing the same state is already large, i.e., $|\mathsf{ADD}| \in \Omega(2^n)$. See for details Figure 8 and [25, Table 2].

[4]The NP-complete satisfiability problem can be reduced to matrix vector multiplication in (poly-sized) BDDs [38]. In practice however, this rarely poses a problem [18].

Table 2 Notation to navigate and construct LIMDDs.

Type	**Notation**	**Semantics**
(New) EDGE e	$e \xrightarrow{A} \textcircled{v}$	$\lvert e\rangle = \lvert \xrightarrow{A}\textcircled{v}\rangle \triangleq A\lvert v\rangle$
(New) NODE v	$\textcircled{v_0} \xleftarrow{A} \textcircled{v} \xrightarrow{B} \textcircled{v_1}$	$\lvert v\rangle \triangleq \lvert 0\rangle A \lvert v_0\rangle + \lvert 1\rangle B \lvert v_1\rangle$
NODE → EDGE	$B \cdot v$	$\triangleq \xrightarrow{B}\textcircled{v}$
EDGE → EDGE	$B \cdot (\xrightarrow{A}\textcircled{v})$	$\triangleq \xrightarrow{BA}\textcircled{v}$
EDGE → EDGE	$\text{FOLLOW}_b(\xrightarrow{\lambda X^x \left[\begin{smallmatrix} z_1 & 0 \\ 0 & z_2 \end{smallmatrix}\right] \otimes P'} \textcircled{v})$	$\triangleq \begin{cases} \xrightarrow{z_1 \lambda P' B_0} \textcircled{v_0} & \textbf{if } x = b,\ \mathsf{low}(v) = \xrightarrow{B_0}\textcircled{v_0} \\ \xrightarrow{z_2 \lambda P' B_1} \textcircled{v_1} & \textbf{if } x \neq b, \mathsf{high}(v) = \xrightarrow{B_1}\textcircled{v_1} \end{cases}$
EDGE → EDGE	$\text{FOLLOW}_{b_n \ldots b_1}(\xrightarrow{A}\textcircled{v})$	$\triangleq \text{FOLLOW}_{b_1}(\ldots \text{FOLLOW}_{b_N}(\xrightarrow{A}\textcircled{v}))..)$

Algorithm 1 Provisionary algorithm MAKEEDGE for creating nodes modulo reduction.

```
1: procedure MAKEEDGE(EDGE $\xrightarrow{A}\textcircled{v}$, EDGE $\xrightarrow{B}\textcircled{w}$)
2:     return $\textcircled{v} \xleftarrow{A} \bigcirc \xrightarrow{B} \textcircled{w}$
```

In line with other existing efficient decision-diagram algorithms, we use dynamic programming in our algorithms to avoid traversing all paths (possibly exponentially many) in the LIMDD (DAG). To implement this, we use a cache data structure (a lossy set) storing the parameters of the recursive function calls.

5.1 Checking state equality

Contrary to other decision diagrams, the nodes of a LIMDD do not represent a single quantum state but an entire equivalence class of states (see Corollary 4). While reduced nodes are canonical descriptions of these equivalence classes, only the root edge, representing individual quantum states in an equivalence class, is not necessarily canonical.

Given two LIMDDs representing the states $\lvert\varphi\rangle$ and $\lvert\psi\rangle$, we can check in $\mathcal{O}(n^3)$ time whether $\lvert\varphi\rangle = \lvert\psi\rangle$ using Algorithm 2. The algorithm uses the canonicity of LIMDDs, which guarantees that two reduced nodes represent the same state if and only if they are equal, and which guarantees that no two nodes represent isomorphic states (Corollary 4). We have $A\lvert v\rangle = B\lvert w\rangle$ if and only if $\lvert v\rangle = A^{-1}B\lvert w\rangle$, which, due to canonicity, can only hold if $A^{-1}B\lvert w\rangle = \lvert w\rangle$. That is, $A^{-1}B$ is a *stabilizer* of $\lvert w\rangle$ and $A^{-1}B$ is an element of the stabilizer subgroup of $\lvert w\rangle$ (see Section 2). We check whether $A^{-1}B$ is a stabilizer of $\lvert w\rangle$ in two steps, assuming we have computed generators for the stabilizer subgroup of $\lvert w\rangle$ (see Subsubsection 5.5.3): first, we check if $A^{-1}B$ commutes with all stabilizer generators. If not, then $A^{-1}B$ cannot be a stabilizer of $\lvert w\rangle$. If it does, then either $A^{-1}B$ or $-A^{-1}B$ is a stabilizer of $\lvert w\rangle$. To distinguish these two cases, we use the Membership algorithm of Section 2 while keeping track of the scalars. The complexity of Algorithm 2 is in $\mathcal{O}(n^2)$ time. In Section 5.5 we will introduce algorithms for stabilizers more completely.

Algorithm 2 Checks whether two reduced LIMDDs represent the same state. The input is the two root edges, pointing to the root nodes v and w of the two diagrams.

```
1: procedure EQUALITY-CHECK(EDGE $\xrightarrow{A}\!\circled{v}$, EDGE $\xrightarrow{B}\!\circled{w}$ with reduced $v, w$)
2:     return $v = w$ and $(A = B = 0$ or $A^{-1}B \in \text{Stab}(v))$
```

5.2 Performing a measurement in the computational basis

We provide a subroutine that, given a LIMDD representation of a state, can sample the outcome of computational-basis measurements on the state (i.e., weak simulation), and a subroutine to compute the exact probability of measuring a given computational basis state $|x\rangle$ for $x \in \{0,1\}^n$ (i.e., strong simulation). For brevity, we do so only for the top qubit. The general case is described in Appendix C.

Simulating the computational-basis measurement of the top qubit of an n-qubit LIMDD, as part of a larger quantum state $|\varphi\rangle = |0\rangle|\varphi_0\rangle + |1\rangle|\varphi_1\rangle$, consists of two parts: first, obtaining the probability of observing output $m \in \{0,1\}$, which equals

$$p(m) = \langle\varphi| \left(|m\rangle\langle m| \otimes \mathbb{I}_2^{\otimes(n-1)}\right) |\varphi\rangle \,/\, \langle\varphi|\varphi\rangle = \langle\varphi_m|\varphi_m\rangle / \langle\varphi|\varphi\rangle.$$

This allows one to to determine which outcome is observed by throwing a random $p(0)$-biased coin. Second, updating the LIMDD to the (here unnormalized) post-measurement state $\left(|m\rangle\langle m| \otimes \mathbb{I}_2^{\otimes(n-1)}\right)|\varphi\rangle = |\varphi_m\rangle$ after obtaining outcome m. In Algorithm 3, we provide efficient algorithms for both parts in case the measured qubit is the top qubit of a LIMDD.

The runtime of computing the probability of a measurement outcome is dominated by the runtime of the subroutine SQUAREDNORM, which computes the quantity $|\langle e|e\rangle|$ given a LIMDD edge e. By saving the squared norm of each node in cache, the algorithm only needs to visit each node once. Consequently, this algorithm runs in time $\mathcal{O}(m)$ when the diagram has m nodes.

The UPDATEPOSTMEAS algorithm is straightforward: in order to update the state $|e\rangle = |0\rangle|e_0\rangle + |1\rangle|e_1\rangle$ after the top qubit is measured to be m, we simply construct an edge $|m\rangle|e_m\rangle$ using the MAKEEDGE subroutine. The runtime is $\mathcal{O}(n)$, since in this case MAKEEDGE only needs to find a new label for the root edge.

To sample a bitstring as measurement outcome, simply repeat the measurement procedure outlined above n times, i.e., first compute the probability p of observing a 1 for the top qubit, and then throw a p-biased coin, obtaining outcome m, and lastly update the LIMDD according to the outcome, and repeat this process for the second qubit, then the third, etc.

For strong simulation, given a bit-string $x = x_n \ldots x_1$, first compute the probability p_n of observing x_n; then update the LIMDD to outcome x_n, obtaining a new, smaller LIMDD. On this new LIMDD, compute the probability p_{n-1} of observing x_{n-1}, and so forth. Note that p_{n-1} is the probability of observing x_{n-1} given that the top qubit is measured to be x_n. Then the probability of observing the string x is the product $p = p_1 \cdots p_n$.

Algorithm 3 Algorithms MeasurementProbability and UpdatePostMeas for respectively computes the probability of observing outcome 0 when measuring the first qubit of a Pauli LIMDD in the computational basis and converting the LIMDD to the post-measurement state after outcome $m \in \{0, 1\}$. The subroutine SquaredNorm takes as input a Pauli LIMDD edge e, and returns $\langle e|e\rangle$.

```
1: procedure MeasurementProbability(Edge $\xrightarrow{A}\textcircled{v}$ with $A = \lambda P_n \otimes \cdots \otimes P_1$)
2:     $p_0 :=$ SquaredNorm(low($v$))
3:     $p_1 :=$ SquaredNorm(high($v$))
4:     return $p_i/(p_0 + p_1)$ for $i = (P_n \in \{X, Y\})$          ▷ (anti-)diagonal Paulis
5: procedure SquaredNorm(Edge $\xrightarrow{\lambda P}\textcircled{v}$)
6:     if $n = 0$ then return $|\lambda|^2$
7:     if $v \in$ cache then return $|\lambda|^2 \cdot$ cache[$v$]          ▷ Dynamic programming
8:     $s :=$ Add(SquaredNorm(follow$_0$($\xrightarrow{\mathbb{I}}\textcircled{v}$)), SquaredNorm(follow$_1$($\xrightarrow{\mathbb{I}}\textcircled{v}$)))
9:     cache[$v$] $:= s$          ▷ Store in dynamic programming cache
10:    return $|\lambda|^2 s$
11: procedure UpdatePostMeas(Edge $e \xrightarrow{\lambda P_n \otimes P'}\textcircled{v}$, measurement outcome $m \in \{0, 1\}$)
12:    if $m = 0$ then
13:        return MakeEdge(follow$_0$($e$),   $0 \cdot$ follow$_0$($e$))
14:    else
15:        return MakeEdge($0 \cdot$ follow$_1$($e$),   follow$_1$($e$))
```

5.3 Simple Gates

Before we give the algorithm for arbitrary gates in Section 5.4, we first show how to apply several simple gates, most of which efficiently. Figure 6 illustrates some of these gates. In the description below, for brevity we omit the calls to MakeEdge to make the diagram canonical again.

- Applying a **single-qubit Pauli gate** Q to qubit k of a LIMDD can be done in constant time, by updating the diagram's root edge from $\lambda P_n \otimes \cdots \otimes P_1$ to $\lambda P_n \otimes \cdots \otimes P_{k+1} \otimes QP_k \otimes P_{k-1} \otimes \cdots \otimes P_1$.
- Applying the **S gate** $\left[\begin{smallmatrix}1 & 0\\ 0 & i\end{smallmatrix}\right]$ to qubit with index k is also efficient. If $k = n$ (top qubit), then note $SB_{\text{root}}|v_{\text{root}}\rangle = (SB_{\text{root}}S^\dagger)S|v_{\text{root}}\rangle$ where $SB_{\text{root}}S^\dagger$ is an ($O(n)$-computable) Pauli LIM because S is a Clifford gate. Further, applying S to v_{root} yields a multiplication with i of the high edge. If $k < n$, then we note $S_k(|0\rangle \otimes |v_0\rangle + |1\rangle \otimes B_{\text{high}}|v_1\rangle) = (|0\rangle \otimes S_k|v_0\rangle + |1\rangle \otimes \left(S_k B_{\text{high}} S_k^\dagger\right) S_k|v_1\rangle)$, where, again $S_k B_{\text{high}} S_k^\dagger$ is a Pauli LIM.
- The application of a **downward Controlled-Pauli gate** CQ_t^c (Q is a single-qubit Pauli gate, c is the control qubit, t is the target qubit with $t < c$) is similar to the S gate in case c is not the top qubit, since controlled-Pauli gates are Clifford gates. If CQ_t^c is applied to node v with $\mathsf{idx}(v) = c$, then update v's high edge label as $B_{\text{high}} \mapsto Q_t B_{\text{high}}$. This operation takes $\mathcal{O}(m)$ time on a diagram with m nodes. A similar algorithm exists for the application of a **downward Controlled-Pauli string**.
- To apply a **Hadamard gate** to the first qubit, see Algorithm 4. The algorithm first constructs $|a_0\rangle = |\varphi_0\rangle + |\varphi_1\rangle$ and $|a_1\rangle = |\varphi_0\rangle - |\varphi_1\rangle$, and then constructs $H \otimes \mathbb{I}|\varphi\rangle \propto |0\rangle|a_0\rangle + |1\rangle|a_1\rangle$. For QMDDs, it was known that applying a Hadamard gate ($H = \left[\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right]$) to the top qubit of a state requires exponential time, because of the needed point-wise addition [25, Table 2]. However, this only happens, in cases where the ADD version of the diagram is already exponential (otherwise addition for ADDs would not be poly-time,

which it is [25, Table 2]). This behavior remains for LIMDDs, however LIMDDs can be exponentially more succinct than QMDDs as shown in Section 4. Applying a Hadamard gate to a stabilizer state, represented as a LIMDD, can be done in polynomial time, by Theorem 31 in Appendix C.2, which shows that the specific pointwise additions required to implement Hadamard are all $\mathcal{O}(n^4)$.

- Applying a **upward Controlled NOT**, i.e., CX_t^c with $t > c$, can be done in polynomial time using only Hadamards and a downward CNOT because $CX_t^c = (H \otimes H) CX_c^t (H \otimes H)$.

It follows that applying Clifford gates to stabilizer states can be done in polynomial time. In general, all Clifford gates are polynomial-time, except for the Hadamard gate.

Algorithm 4 Applies a Hadamard gate to the first qubit. Specifically, given a LIMDD edge for a state $|\varphi\rangle$, returns a LIMDD edge for the state $|\psi\rangle = H \otimes \mathbb{1}_2^{\otimes n-1} |\varphi\rangle$. ADD is explained in Section 5.4.

```
1: procedure ApplyHadamard(Edge $e \xrightarrow{A} \textcircled{v}$)
2:     Edge $a_0 :=$ Add(follow$_0(e)$, follow$_1(e)$)
3:     Edge $a_1 :=$ Add(follow$_0(e)$, $-$follow$_1(e)$)
4:     return $\frac{1}{\sqrt{2}} \cdot$ MakeEdge$(a_0, a_1)$
```

5.4 Applying a generic multi-qubit gate to a state

In order tot represent quantum gates ($2^n \times 2^n$ unitary matrices) using decision diagrams, we use the standard approach [27]. A natural choice is to let the coefficient of $|a_1, \ldots, a_n, b_1, \ldots, b_n\rangle$, for $a_j, b_j \in \{0,1\}$ be the matrix entry in row a and column b. We slightly adjust this and interleave the row and column variables to facilitate recursive descent on the structure: $u(a_1, b_1, a_2, b_2, \ldots, a_n, b_n)$. Therefore, for $x, y \in \{0,1\}$, the subfunction u_{xy} represents a quadrant of the matrix, namely the submatrix $u_{xy}(a_2, b_2, \ldots, a_n, b_n) \triangleq u(x, y, a_2, b_2, \ldots, a_n, b_n)$, as follows: $u = \overbrace{\begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix}}^{u_{0*}} \Big\} u_{*1}$. Definition 10 formalizes this. Figure 6 shows a few examples of gates represented as LIMDDs.

▶ **Definition 10** (LIMDDs for gates). *A reduced LIMDD edge* $\xrightarrow{A}\textcircled{u}$ *can represent a (unitary)* $2^n \times 2^n$ *matrix* M *iff* idx$(u) = 2n$. *The matrix value of cell* $M_{r,c}$ *is defined as* FOLLOW$_{r_1 c_1 r_2 c_2 \ldots r_n c_n}(\xrightarrow{A}\textcircled{u})$ *where* r, c *are the row and column indices, respectively, with binary representation* $r_1, \ldots, r_n$ *and* $c_1, \ldots, c_n$. *The semantics of a LIMDD edge* u *as a matrix is denoted* $[u] \triangleq M$ *(as opposed to its semantics* $|u\rangle$ *as a vector).*

The procedure ApplyGate (Algorithm 5) applies a gate U to a state $|\varphi\rangle$, represented by LIMDDs e^u and e^v. It outputs is a LIMDD edge representing $U|\varphi\rangle$. It works as follows (see Figure 7 for an illustration). Using the follow$_x(e)$ procedure, we write $|\varphi\rangle$ and U as

$$|\varphi\rangle = |0\rangle|\varphi_0\rangle + |1\rangle|\varphi_1\rangle \tag{8}$$

$$U = |0\rangle\langle 0| \otimes U_{00} + |0\rangle\langle 1| \otimes U_{01} + |1\rangle\langle 0| \otimes U_{10} + |1\rangle\langle 1| \otimes U_{11} \tag{9}$$

Then, on line 8, we compute each of the four terms $U_{rc}|\varphi_c\rangle$ for row/column bits $r, c \in \{0,1\}$. We do this by constructing four LIMDDs $f_{r,c}$ representing the states $|f_{r,c}\rangle = U_{r,c}|\varphi_c\rangle$, using four recursive calls to the ApplyGate algorithm. Next, on lines 9 and 10, the appropriate

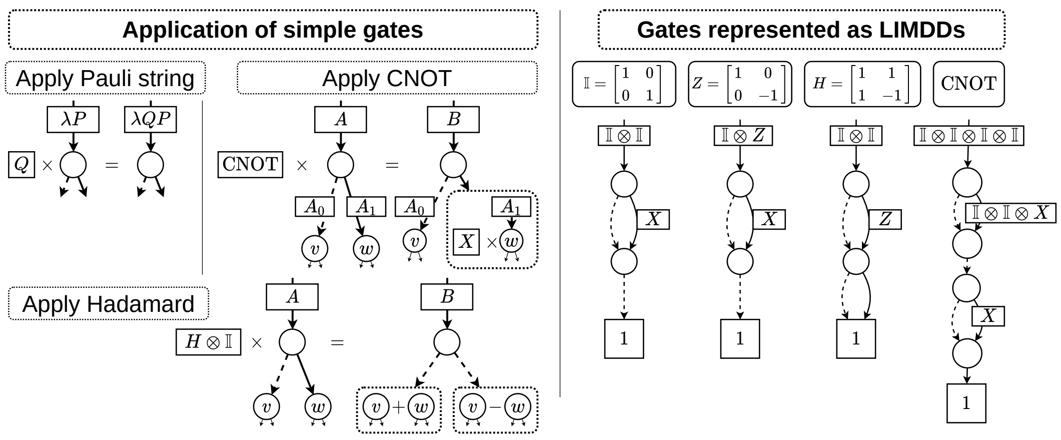

Application of simple gates
Apply Pauli string
Apply CNOT
Apply Hadamard
Gates represented as LIMDDs
λP
λQP
$Q \times$
CNOT ×
A
B
A_0
A_1
$X \times w$
$H \otimes \mathbb{I} \times$
$v + w$
$v - w$
$\mathbb{I} = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$
$Z = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$
$H = \begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$
CNOT
$\mathbb{I}\otimes\mathbb{I}$
$\mathbb{I}\otimes Z$
$\mathbb{I}\otimes\mathbb{I}\otimes\mathbb{I}\otimes\mathbb{I}$
$\mathbb{I}\otimes\mathbb{I}\otimes X$
X
Z
1

Figure 6 **Left half**: illustration of applying several simple gates (some details are omitted). Leftmost: applying a Pauli string Q to an edge entails only updating the label on that edge. Right: To apply a CNOT gate to the top qubit, apply $X \otimes \mathbb{I}^{\otimes n-2}$ to the right child; here this operation is represented inside a dotted box. The incoming edge's label changes from A to $B = \text{CNOT} \cdot A \cdot \text{CNOT}$, which is guaranteed to be a Pauli string. Bottom: Applying a Hadamard gate on the top qubit is done by first making nodes representing $|v\rangle + |w\rangle$ and $|v\rangle - |w\rangle$, here shown in dotted boxes. **Right half**: Examples of four gates, represented as LIMDDs. Left: The identity gate $\mathbb{I}$ is represented by a LIMDD of two levels. The first level indicates the row, and the second level indicates the column. Second from the right: the Hadamard gate; notice that the Z label produces the -1 value in the matrix' bottom right entry. Rightmost: The CNOT gate. Since this gate is on $n = 2$ qubits, it has $2n = 4$ levels. Edges with label $\mathbb{I}$ are drawn without label; edges with label 0 are not drawn.

states are added, using Add (Algorithm 6), producing LIMDDs e_0 and e_1 for the states $|e_0\rangle = U_{00}|\varphi_0\rangle + U_{10}|\varphi_1\rangle$ and for $|e_1\rangle = U_{01}|\varphi_0\rangle + U_{11}|\varphi_1\rangle$. The base case of ApplyGate is the case where $n = 0$, in which case both e^u and e^v are edges which point to the leaf, which means U and $|v\rangle$ are simply scalars.

Here, too, we employ dynamic programming to prevent the algorithm from performing duplicate computations. Namely, when we have computed the edge e^w, we store this result in the cache (line 12). In subsequent calls, this result can be retrieved from the cache (on line 6), recovering the result without performing the computation again.

Specifically, we store a tuple (P', u, Q', v) in the cache. Here $P' = \text{RootLabel}(\xrightarrow{P}\textcircled{u})$ and $Q' = \text{RootLabel}(\xrightarrow{Q}\textcircled{v})$ are canonically chosen LIMs. By "canonically chosen", we mean that $Q'|v\rangle = Q|v\rangle$, and that the procedure chooses the same LIM Q' for every $\xrightarrow{Q''}\textcircled{v}$ such that $Q''|v\rangle = Q|v\rangle$. We do this so that, in a subsequent call to ApplyGate with inputs $(\xrightarrow{\mu_A P''}\textcircled{u}, \xrightarrow{\mu_B Q''}\textcircled{v})$, we will find the correct result in the cache whenever $Q''|v\rangle = Q|v\rangle$, even if $Q'' \neq Q$.

A specific choice for RootLabel is the lexicographic minimum of all possible root labels, following a similar choice for making a canonical choice for the high edge label of a node in Section 5.5. In Algorithm 13 in that section, we will give an $O(n^3)$-time algorithm for computing the lexicographically minimal root label.

Notice also that the scalars λ_A, λ_B are not stored in the cache; they are factored out and multiplied as needed on lines 6 and 13. Using similar reasoning, we also do not propagate these scalars into the recursive calls on line 8.

Algorithm 5 Applies the gate $[e^u]$ to the state $|e^v\rangle$. Here e^u and e^v are LIMDD edges. The output is a LIMDD edge e^w satisfying $|e\rangle = [e^u]\,|e^v\rangle$. It assumes that $2\mathsf{idx}(v) = \mathsf{idx}(u)$.

```
1: procedure ApplyGate(Edge $e^u \xrightarrow{A} (u)$, Edge $e^v \xrightarrow{B} (v)$ with $A = \lambda_A P$, $B = \lambda_B Q$)
2:     if $A = 0$ or $B = 0$ then return 0
3:     if $n = 0$ then return $\xrightarrow{AB} (v)$
4:     $P', Q' :=$ RootLabel($\xrightarrow{P} (u)$), RootLabel($\xrightarrow{Q} (v)$)          ▷ Get canonical root labels
5:     if ($\xrightarrow{P'} (u)$, $\xrightarrow{Q'} (v)$) $\in$ Apply-cache then          ▷ Dynamic programming
6:         return $\lambda_A \lambda_B \cdot$ Apply-cache[$\xrightarrow{P'} (u)$, $\xrightarrow{Q'} (v)$]
7:     for $r, c \in \{0,1\}$ do
8:         Edge $f_{r,c} :=$ ApplyGate(follow$_{rc}$($\xrightarrow{Q} (u)$), follow$_c$($\xrightarrow{P} (v)$))
9:     Edge $e_0 :=$ Add($f_{0,0}, f_{0,1}$)
10:    Edge $e_1 :=$ Add($f_{1,0}, f_{1,1}$)
11:    Edge $e^w :=$ MakeEdge($e_0, e_1$)
12:    Apply-cache[$\xrightarrow{P'} (u)$, $\xrightarrow{Q'} (v)$] $:= e^w$          ▷ Store result in cache
13:    return $\lambda_A \lambda_B \cdot e^w$
```

The subroutine Add (Algorithm 6) adds two quantum states, i.e., given two LIMDDs representing $|e\rangle$ and $|f\rangle$, it returns a LIMDD representing $|e\rangle + |f\rangle$. A natural way to implement this algorithm would have been to use the follow$_x(e)$ procedure to express the states as $|e\rangle = |0\rangle\,|e_0\rangle + |1\rangle\,|e_1\rangle$ and $|f\rangle = |0\rangle\,|f_0\rangle + |1\rangle\,|f_1\rangle$, and then to call the algorithm recursively to construct the states $|e_0\rangle + |f_0\rangle$ and $|e_1\rangle + |f_1\rangle$.

However, we implement the algorithm slightly differently in order to better take advantage of dynamic programming. We remark that we are looking to construct the state $A\,|v\rangle + B\,|w\rangle$, and that this is equal to $A \cdot (|v\rangle + A^{-1}B\,|w\rangle)$. Therefore, letting $|\psi\rangle = A^{-1}B\,|w\rangle$, we use the follow$_x$($\xrightarrow{A^{-1}B} (w)$) procedure to construct the states $|\psi_0\rangle$ and $|\psi_1\rangle$. Then, we construct the states $|a_0\rangle = |v_0\rangle + |\psi_0\rangle$ and $|a_1\rangle = |v_1\rangle + |\psi_1\rangle$, on Line 6 and 7. Lastly, we construct the

Algorithm 6 Given two n-LIMDD edges e, f, constructs a new LIMDD edge a with $|a\rangle = |e\rangle + |f\rangle$.

```
1: procedure Add(Edge $e \xrightarrow{A} (v)$, Edge $f \xrightarrow{B} (w)$ with $A = \lambda P$, $B = \mu Q$, $\mathsf{idx}(v) = \mathsf{idx}(w)$)
2:     if $n = 0$ then return $\xrightarrow{A+B} (1)$          ▷ $A, B \in \mathbb{C}$
3:     if $v \not\preceq w$ then return Add($f, e$)
4:     $C :=$ RootLabel($\xrightarrow{A^{-1}B} (w)$)
5:     if ($v$, $\xrightarrow{C} (w)$) $\in$ Add-cache then return $A \cdot$ Add-cache[$v$, $\xrightarrow{C} (w)$]          ▷ Dynamic programming
6:     Edge $a_0 :=$ Add(follow$_0$($\xrightarrow{\mathbb{I}} (v)$), follow$_0$($\xrightarrow{C} (w)$))
7:     Edge $a_1 :=$ Add(follow$_1$($\xrightarrow{\mathbb{I}} (v)$), follow$_1$($\xrightarrow{C} (w)$))
8:     Edge $a :=$ MakeEdge($a_0, a_1$)
9:     Add-cache[$v$, $\xrightarrow{C} (w)$] $:= a$          ▷ Store in dynamic programming cache
10:    return $A \cdot a$
```

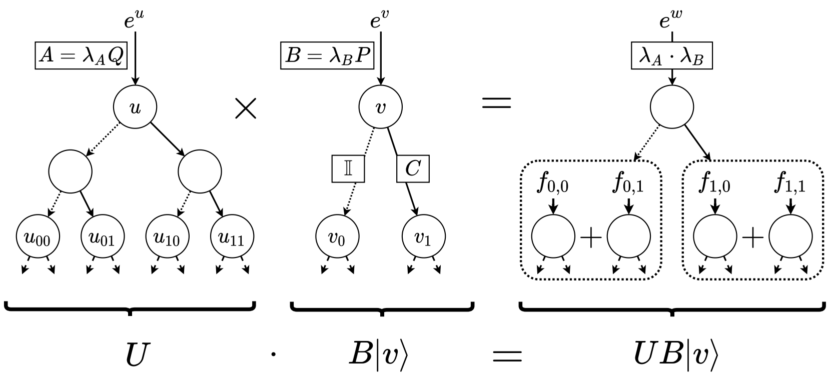


Figure 7 An illustration of ApplyGate (Algorithm 5), where matrix U is applied to state $B\,|v\rangle$, both represented as Pauli-LIMDDs. The edges $f_{0,0}$, $f_{0,1}$, etc. are the edges made on line 8. The dotted box indicates that these states are added, using Add (Algorithm 6) before they are passed to MakeEdge.

state $|a\rangle = |0\rangle\,|a_0\rangle + |1\rangle\,|a_1\rangle$, on line 8. Returning the LIMDD $A \cdot a$ yields the desired result:

$$A\,|a\rangle = A(|0\rangle\,|a_0\rangle + |1\rangle\,|a_1\rangle) = A(|0\rangle\,(|v_0\rangle + |\psi_0\rangle) + |1\rangle\,(|v_1\rangle + |\psi_1\rangle)) \tag{10}$$

$$= A(|v\rangle + |\psi\rangle) = A(|v\rangle + A^{-1}B\,|w\rangle) = A\,|v\rangle + B\,|w\rangle \tag{11}$$

We store a tuple (v, w, C) in the cache, where $C = \text{RootLabel}(\xrightarrow{A^{-1}B}\textcircled{w})$ is a canonically chosen LIM such that $C\,|w\rangle = A^{-1}B\,|w\rangle$. By “canonically chosen”, we mean, again, that the procedure chooses the same C for any input $\xrightarrow{D}\textcircled{w}$, $\xrightarrow{E}\textcircled{w}$ such that $D\,|w\rangle = E\,|w\rangle$.

Finally, on line 3, we use the total order on nodes $\preccurlyeq$, in order to prevent cache misses due to storing v and w in the wrong order.

The worst-case running time of Add is $\mathcal{O}(n^3 2^n)$, if n is the number of qubits. The resulting LIMDD can be exponential in the input sizes (bounded by 2^n), and Add calls MakeEdge has runtime $\mathcal{O}(n^3)$. This exponential result happens already for QMDDs because the addition might remove any common factors that can be factored out as illustrated in Figure 8. This exponential-time worst-case behavior for QMDDs (and related DDs) was identified [25, Table 2] and is inherited by LIMDDs. However, the Add algorithm is polynomial-time when $v = w$ and v is a stabilizer state (Corollary 33)

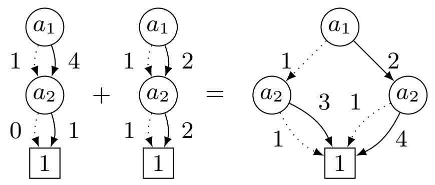
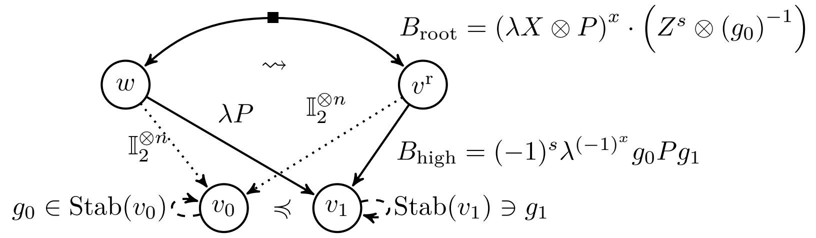


Figure 8 Adding two states $(0, 1, 0, 4)$ and $(1, 2, 2, 4)$ as QMDDs can cause an exponentially larger result QMDD $(1, 3, 2, 8)$ due to the loss of common factors.

5.5 The MakeEdge subroutine

To construct new nodes and edges, our algorithms use the MakeEdge subroutine as discussed above. MakeEdge produces reduced parent nodes for reduced children, so that the LIMDD representation becomes canonical; an important property for efficient manipulation algorithms and analysis techniques. Here we give the algorithm for MakeEdge and show that it runs in time $O(n^3)$ (assuming the input nodes are reduced).


$B_{\text{root}} = (\lambda X \otimes P)^x \cdot \left(Z^s \otimes (g_0)^{-1}\right)$
$\rightsquigarrow$
w
v^{r}
λP
$\mathbb{I}_2^{\otimes n}$
$\mathbb{I}_2^{\otimes n}$
$B_{\text{high}} = (-1)^s \lambda^{(-1)^x} g_0 P g_1$
$g_0 \in \text{Stab}(v_0)$
v_0
$\preccurlyeq$
v_1
$\text{Stab}(v_1) \ni g_1$

Choose $s, x \in \{0,1\}, g_0 \in \text{Stab}(v_0), g_1 \in \text{Stab}(v_1)$ s.t. B_{high} is minimal and $x = 0$ if $v_0 = v_1$.

Figure 9 Illustration of finding a canonical high label for a semi-reduced node w, yielding a reduced node v^{r}. The chosen high label is the minimal element from the set of eligible high labels based on stabilizers g_0, g_1 of v_0, v_1 (drawn as self loops). The minimal element holds a factor $\lambda^{(-1)^x}$ for some $x \in \{0,1\}$. There are two cases: if $v_0 \neq v_1$ or $x = 0$, then the factor is λ and the root edge should be adjusted with an $\mathbb{I}_2$ or Z on the root qubit. The other case, $x = 1$, leads to an additional multiplication with an X on the root qubit.

We call a LIMDD node *semi-reduced*, if it satisfies the low factoring, low precedence, zero edge and merge rules of Definition 3, i.e., if it satisfies all reduction rules except possibly high determinism. Semi-reduction is easy to enforce, as shown in Subsubsection 5.5.1. The bulk of this section discusses how to obtain a canonical LIM on the high edge to realize high determinism. We achieve this in Subsubsection 5.5.2 by selecting a lexicographically minimum LIM from all the candidates. The problem of find all candidates efficiently is reduced to finding all stabilizers of a state $|v\rangle$, i.e., the set of all LIMs A such that $A|v\rangle = |v\rangle$, efficiently represented as a set of generators (see also Section 2). In Subsubsection 5.5.3, we show how to compute these stabilizers in the bottom-up fashion typical for decision diagrams.

5.5.1 Basic reduction in MakeEdge

Algorithm 7 gives the MAKEEDGE subroutine that now constructs canonical nodes through reduction, yielding a new root edge (whose LIM label is not necessarily canonical). It takes as input two edges e_0 and e_1, pointing to already reduced nodes, and outputs an edge e, pointing to a reduced node v, such that $|e\rangle = |0\rangle |e_0\rangle + |1\rangle |e_1\rangle$. The reduction proceeds as follows.

- First it ensures low precedence, switching e_0 and e_1 if necessary at Line 3. This is also done if e_0's label A is 0 to allow for low factoring (avoiding divide by zero).
- Low factoring, i.e., dividing out the LIM A, placing it on the root node, is visualized in Figure 4 for the cases $e_1 = 0 / e_1 \neq 0$, and done in the algorithm at Line 6,7 / 9,11.
- The zero edges rule is enforced in the $B = 0$ branch by taking $v_1 := v_0$.
- The canonical high label B_{high} is computed by GETLABELS, discussed below, for low factored node $\boxed{v_0} \xleftarrow{\mathbb{I}} \boxed{w} \xrightarrow{\hat{A}} \boxed{v_1}$. It satisfies the high determinism rule of Definition 3 with HighLabel$(w) = B_{\text{high}}$ as shown in the next section with Corollary 12.
- Finally, we merge nodes by creating an entry $(v_0, B_{\text{high}}, v_1)$ in a *unique table* [9] at Line 13.

All steps except for GETLABELS have complexity $O(1)$ or $O(n)$ (for checking low precedence, we use nodes' order in the unique table). GETLABELS has a runtime $O(n^3)$ as we show in Subsubsection 5.5.2, yielding an overall complexity $O(n^3)$.

Algorithm 7 Algorithm MAKEEDGE takes two root edges to (already reduced) nodes v_0, v_1, the children of a new node, and returns a reduced node with root edge. It assumes that $\mathsf{idx}(v_0) = \mathsf{idx}(v_1) = n$. We indicate which lines of code are responsible for which reduction rule in Definition 3.

```
1: procedure MAKEEDGE(EDGE $e_0 \xrightarrow{A} \textcircled{$v_0$}$, EDGE $e_1 \xrightarrow{B} \textcircled{$v_1$}$ , where $v_0, v_1$ reduced, with $A \neq 0$ or $B \neq 0$)
2:     if $v_0 \not\preccurlyeq v_1$ or $A = 0$ then                ▷ Enforce low precedence and enable factoring
3:         return $(X \otimes \mathbb{1}_2^{\otimes n}) \cdot \text{MakeEdge}(e_1, e_0)$
4:     if $B = 0$ then
5:         $v_1 := v_0$                                                  ▷ Enforce zero edges
6:         $v := \textcircled{$v_0$} \overset{\mathbb{1}_2^{\otimes n}}{\dashleftarrow} \bigcirc \xrightarrow{0} \textcircled{$v_0$}$                                   ▷ Enforce low factoring
7:         $B_{\text{root}} := \mathbb{1}_2 \otimes A$                    ▷ $B_{\text{root}} |v\rangle = |0\rangle \otimes A |v_0\rangle + |1\rangle \otimes B |v_1\rangle$
8:     else
9:         $\hat{A} := A^{-1}B$                                          ▷ Enforce low factoring
10:        $B_{\text{high}}, B_{\text{root}} := \text{GETLABELS}(\hat{A}, v_0, v_1)$     ▷ Enforce high determinism
11:        $v := \textcircled{$v_0$} \overset{\mathbb{1}_2^{\otimes n}}{\dashleftarrow} \bigcirc \xrightarrow{B_{\text{high}}} \textcircled{$v_1$}$          ▷ $B_{\text{root}} |v\rangle = |0\rangle \otimes |v_0\rangle + |1\rangle \otimes A^{-1}B |v_1\rangle$
12:        $B_{\text{root}} := (\mathbb{I}_2 \otimes A) B_{\text{root}}$     ▷ $(\mathbb{I}_2 \otimes A) B_{\text{root}} |v\rangle = |0\rangle \otimes A |v_0\rangle + |1\rangle \otimes B |v_1\rangle$
13:    $v^r :=$ Find or create unique table entry UNIQUE$[v] = (v_0, B_{\text{high}}, v_1)$      ▷ Enforce merge
14:    return $\xrightarrow{B_{\text{root}}} \textcircled{$v^r$}$
```

5.5.2 Choosing a canonical high-edge label

On line 10, the MAKEEDGE algorithm finds a canonical label B_{high} for the high edge of node v with a call to GETLABELS. It does so by taking the lexicographically minimal candidate for B_{high}, see Section 2. We now first characterize all eligible labels B_{high}, by reducing the problem to finding stabilizer subgroups of the children nodes v_0, v_1 (see Section 2), denoted as $\text{Stab}(v_0)$ and $\text{Stab}(v_1)$. Then, we show that GETLABELS (Algorithm 8) correctly finds the lexicographically minimal eligible LIM (and corresponding root label), and runs in time $O(n^3)$ where n is the number of qubits.

Figure 9 illustrates this process. It shows the status of the MAKEEDGE algorithm on line 10, when it has enough information to construct the semi-reduced node $\textcircled{$v_0$} \overset{\mathbb{I}^{\otimes n}}{\dashleftarrow} \textcircled{w} \xrightarrow{\hat{A}} \textcircled{v_1}$, with $\hat{A} = \lambda P$ on its high edge, shown on the left. The set of eligible high labels is shown, and the lexicographically minimal is chosen as B_{high}, yielding a new node v^r (with 'r' for 'reduced'). This set of labels is decomposed into a choice of v_0, v_1 stabilizer g_0, g_1 and a choice for the most significant PAULI operator on the root LIM $X^x Z^s$. Theorem 11 shows that this captures all possible high edges.

▶ **Theorem 11** (Eligible high-edge labels). *Let $\textcircled{$v_0$} \overset{\mathbb{I}_2^{\otimes n}}{\dashleftarrow} \textcircled{w} \xrightarrow{\lambda P} \textcircled{v_1}$ be a semi-reduced n-qubit node in a Pauli-LIMDD, where v_0, v_1 are reduced, P is a Pauli string and $\lambda \neq 0$. For all nodes $v = \textcircled{$v_0$} \overset{\mathbb{I}_2^{\otimes n}}{\dashleftarrow} \textcircled{v} \xrightarrow{B_{\text{high}}} \textcircled{v_1}$, it holds that $|w\rangle \simeq |v\rangle$ if and only if*

$$B_{\text{high}} = (-1)^s \cdot \lambda^{(-1)^x} g_0 P g_1 \tag{12}$$

for some $g_0 \in \text{Stab}(v_0), g_1 \in \text{Stab}(v_1), s, x \in \{0, 1\}$ and $x = 0$ if $v_0 \neq v_1$. An isomorphism mapping $|w\rangle$ to $|v\rangle$ is

$$B_{\text{root}} = (X \otimes \lambda P)^x \cdot (Z^s \otimes (g_0)^{-1}). \tag{13}$$

Proof. It is straightforward to verify that the isomorphism B_{root} in eq. (13) indeed maps $|w\rangle$ to $|v\rangle$ (as $x = 1$ implies $v_0 = v_1$), which shows that $|w\rangle \simeq |v\rangle$. For the converse direction, suppose there exists an n-qubit Pauli LIM C such that $C\,|w\rangle = |v\rangle$, i.e.

$$C\left(|0\rangle \otimes |v_0\rangle + \lambda\,|1\rangle \otimes P\,|v_1\rangle\right) = |0\rangle \otimes |v_0\rangle + |1\rangle \otimes B_{\text{high}}\,|v_1\rangle\,. \tag{14}$$

We show that if B_{high} satisfies eq. (14), then it has a decomposition as in eq. (12). We write $C = Q_{\text{top}} \otimes C_{\text{rest}}$ where Q_{top} is a single-qubit Pauli operator and C_{rest} is an $(n-1)$-qubit Pauli LIM (or a complex number $\neq 0$ if $n = 1$). We treat the two cases $Q_{\text{top}} \in \{\mathbb{1}_2, Z\}$ and $Q_{\text{top}} \in \{X, Y\}$ separately:

Case $\mathbb{1}_2, Z$. Then $Q_{\text{top}} = \left[\begin{smallmatrix}1 & 0\\ 0 & (-1)^y\end{smallmatrix}\right]$ for $y \in \{0,1\}$. In this case, eq. (14) implies $C_{\text{rest}} \in \text{Stab}(|v_0\rangle)$ and $(-1)^y\lambda C_{\text{rest}}P\,|v_1\rangle = B_{\text{high}}\,|v_1\rangle$, or, equivalently, $(-1)^{-y}\lambda^{-1}P^{-1}C_{\text{rest}}^{-1}B_{\text{high}} \in \text{Stab}(v_1)$. Hence, by choosing $s = y$ and $x = 0$, we compute

$$(-1)^y\lambda^{(-1)^0}\underbrace{C_{\text{rest}}}_{\in\text{Stab}(v_0)}\;P\,\underbrace{(-1)^{-y}\lambda^{-1}P^{-1}C_{\text{rest}}^{-1}B_{\text{high}}}_{\in\text{Stab}(v_1)} = \frac{(-1)^y\lambda^{(-1)^0}}{(-1)^y\lambda}B_{\text{high}} = B_{\text{high}}$$

Case X, Y. Write $Q_{\text{top}} = \begin{pmatrix}0 & z^{-1}\\ z & 0\end{pmatrix}$ where $z \in \{1, i\}$. Now, eq. (14) implies

$$zC_{\text{rest}}\,|v_0\rangle = B_{\text{high}}\,|v_1\rangle \qquad \text{and} \qquad z^{-1}\lambda C_{\text{rest}}P\,|v_1\rangle = |v_0\rangle\,. \tag{15}$$

From eq. (15), we first note that $|v_0\rangle$ and $|v_1\rangle$ are isomorphic, so by Corollary 4, we have $v_0 = v_1$. Consequently, we find from eq. (15) that $z^{-1}C_{\text{rest}}^{-1}B_{\text{high}} \in \text{Stab}(v_0)$ and $z^{-1}\lambda C_{\text{rest}}P \in \text{Stab}(v_1)$. Now choose $x = 1$ and choose s such that $(-1)^s \cdot z^{-2}C_{\text{rest}}^{-1}B_{\text{high}}C_{\text{rest}} = B_{\text{high}}$ (recall that Pauli LIMs either commute or anticommute, so $B_{\text{high}}C_{\text{rest}} = \pm C_{\text{rest}}B_{\text{high}}$). This yields:

$$(-1)^s\lambda^{-1}\cdot\underbrace{z^{-1}C_{\text{rest}}^{-1}B_{\text{high}}}_{\in\text{Stab}(v_0)}\cdot P\cdot\underbrace{z^{-1}\lambda PC_{\text{rest}}}_{\in\text{Stab}(v_1)} = \lambda^{-1}\cdot\lambda\cdot(-1)^s z^{-2}\cdot\left(C_{\text{rest}}^{-1}B_{\text{high}}C_{\text{rest}}\right) = B_{\text{high}}$$

where we used the fact that $P^2 = \mathbb{I}_2^{\otimes(n-1)}$ because P is a Pauli string. ◀

▶ **Corollary 12.** *As a corollary of Theorem 11, we find that taking, as in Figure 9,*

$$\textsf{\textit{HighLabel}}\Big(\begin{tikzcd} v_0 & v \arrow[l, dashed, "\mathbb{I}"'] \arrow[r, "\lambda P"] & v_1 \end{tikzcd}\Big) = \min_{i,s,x\in\{0,1\},g_i\in Stab(v_i)}\left(\left\{(-1)^s\cdot\lambda^{(-1)^x}\cdot g_0\cdot P\cdot g_1 \;\middle|\; x \neq 1 \text{ if } v_0 \neq v_1\right\}\right)$$

yields a proper implementation of ***HighLabel*** *as required by Definition 3, because it considers all possible B_{high} such that $|v\rangle \simeq_{\text{PAULI}} |0\rangle\,|v_0\rangle + |1\rangle \otimes$* ***HighLabel****$(v)\,|v_1\rangle$.*

A naive implementation for GETLABELS would follow the possible decompositions of eligible LIMs (see eq. (12)) and attempt to make this LIM smaller by greedy multiplication, first with stabilizers of $g_0 \in \text{Stab}(v_0)$, and then with stabilizers $g_1 \in \text{Stab}(v_1)$. To see why this does not work, consider the following example: the high edge label is Z and the stabilizer subgroups $\text{Stab}(v_0) = \langle X\rangle$ and $\text{Stab}(v_1) = \langle Y\rangle$. Then the naive algorithm would terminate and return Z because $X, Y > Z$, which is incorrect since the high-edge label $X \cdot Z \cdot Y = -i\mathbb{I}_2$ is smaller than Z.

To overcome this, we consider the group closure of *both* $\text{Stab}(v_0)$ *and* $\text{Stab}(v_1)$. See Algorithm 8 for the $O(n^3)$-algorithm for GETLABELS, which proceeds in two steps. In the first step (Line 3), we use the subroutine ARGLEXMIN for finding the minimal Pauli LIM A such

Algorithm 8 Algorithm for finding the LIMs B_{high} and B_{root} required by MAKEEDGE. The LIM B_{high} is chosen canonically as the lexicographically smallest LIM in the set characterized in Theorem 11. It runs in $O(n^3)$-time (with n the number of qubits), provided `GetStabilizerGenSet` has been computed for the children v_0, v_1.

```
1:  procedure GETLABELS(PauliLim $\lambda P \neq 0$ (current high label), reduced children nodes $v_0, v_1$)
    Output: canonical high label $B_{\text{high}}$ and root label $B_{\text{root}}$
2:      $G_0, G_1 :=$ GetStabilizerGenSet($v_0$), GetStabilizerGenSet($v_1$)
3:      $(g_0, g_1) :=$ ARGLEXMIN$(G_0, G_1, \lambda P)$
4:      if $v_0 = v_1$ then
5:          $(x, s) := \arg\min_{(x,s)\in\{0,1\}^2} (-1)^s \lambda^{(-1)^x} g_0 P g_1$
6:      else
7:          $x := 0$
8:          $s := \arg\min_{s\in\{0,1\}} (-1)^s \lambda g_0 P g_1$
9:      $B_{\text{high}} := (-1)^s \cdot \lambda^{(-1)^x} \cdot g_0 \cdot P \cdot g_1$
10:     $B_{\text{root}} := (X \otimes \lambda P)^x \cdot (Z^s \otimes (g_0)^{-1})$
11:     return $(B_{\text{high}}, B_{\text{root}})$
```

that $A = \lambda P \cdot g_0 \cdot g_1$ for $g_0 \in \text{Stab}(v_0), g_1 \in \text{Stab}(v_1)$. We will explain and prove correctness of this subroutine below in Subsubsection 5.5.4. In the second step (Lines 4-8), we follow eq. (13) by also minimizing over x and s. Finally, the algorithm returns B_{high}, the minimum of all eligible edge labels according to eq. (13), together with a root edge label B_{root} which ensures the represented quantum state remains the same.

Below, we will explain $O(n^3)$-time algorithms for finding generating sets for the stabilizer subgroup of a reduced node and for ARGLEXMIN. Since all other lines in Algorithm 8 can be performed in linear time, its overall runtime is $O(n^3)$. Note that we can amortize `GetStabilizerGenSet` over the MAKEEDGE calls.

5.5.3 Constructing the stabilizer subgroup of a LIMDD node

In this section, we give a recursive subroutine `GetStabilizerGenSet` to construct the stabilizer subgroup $\text{Stab}(|v\rangle) := \{A \in \text{PAULILIM}_n \mid A|v\rangle = |v\rangle\}$ of an n-qubit LIMDD node v (see Section 2). The subroutine is used by the algorithm GETLABELS to select a canonical label for the high edge and root edge. If the stabilizer subgroup of v's children have been computed already, `GetStabilizerGenSet`'s runtime is $O(n^3)$. `GetStabilizerGenSet` returns a generating set for the group $\text{Stab}(|v\rangle)$. Since these stabilizer subgroups are generally exponentially large in the number of qubits n, but they have at most n generators, storing only the generators instead of all elements may save an exponential amount of space. Because any generator set G of size $|G| > n$ can be brought back to at most n generators in time $\mathcal{O}(|G| \cdot n^2)$ (see Section 2), we will in the derivation below show how to obtain generator sets of size linear in n and leave the size reduction implicit. We will also use the notation $A \cdot G$ and $G \cdot A$ to denote the sets $\{A \cdot g | g \in G\}$ and $\{g \cdot A | g \in G\}$, respectively.

We now sketch the derivation of the algorithm. The base case of the algorithm is the Leaf node of the LIMDD, representing the number 1, which has stabilizer group $\{1\}$. For the recursive case, we wish to compute the stabilizer group of a reduced n-qubit node $v = \begin{tikzcd} \mathbb{I} & v \arrow[l, dotted, "v_0"'] \arrow[r, "B_{\text{high}}"] & v_1 \end{tikzcd}$. If $B_{\text{high}} = 0$, then it is straightforward to see that $\lambda P_n \otimes P' |v\rangle = |v\rangle$ implies $P_n \in \{\mathbb{I}_2, Z\}$, and further that $\text{Stab}(|v\rangle) = \langle\{P_n \otimes g \mid g \in G_0, P_n \in \{\mathbb{I}_2, Z\}\}\rangle$, where G_0 is a stabilizer generator set for v_0.

If $B_{\text{high}} \neq 0$, then we expand the stabilizer equation $\lambda P |v\rangle = |v\rangle$:

$$\lambda P_n \otimes P' \left(|0\rangle \otimes |v_0\rangle + |1\rangle \otimes B_{\text{high}} |v_1\rangle\right) = |0\rangle \otimes |v_0\rangle + |1\rangle \otimes B_{\text{high}} |v_1\rangle \text{, which implies:}$$

$$\lambda P' |v_0\rangle = |v_0\rangle \;\wedge\; z\lambda P' B_{\text{high}} |v_1\rangle = B_{\text{high}} |v_1\rangle \quad \textbf{for } P_n = \left[\begin{smallmatrix} 1 & 0 \\ 0 & z \end{smallmatrix}\right], z \in \{1, -1\} \quad (16)$$

$$y^* \lambda P' B_{\text{high}} |v_1\rangle = |v_0\rangle \;\wedge\; \lambda P' |v_0\rangle = y^* B_{\text{high}} |v_1\rangle \quad \textbf{for } P_n = \begin{bmatrix} 0 & y^* \\ y & 0 \end{bmatrix}, y \in \{1, i\} \quad (17)$$

The stabilizers can therefore be computed according to Equation 16 and 17 as follows.

$$\begin{aligned}\text{Stab}(|v\rangle) = &\bigcup_{z = \in\{1,-1\}, y \in \{1,i\}} \left[\begin{smallmatrix} 1 & 0 \\ 0 & z \end{smallmatrix}\right] \otimes (\text{Stab}(|v_0\rangle) \cap z \cdot \text{Stab}(B_{\text{high}} |v_1\rangle)) \\ &\qquad \cup \begin{bmatrix} 0 & y \\ y^* & 0 \end{bmatrix} \otimes \big(\text{Iso}(y^* B_{\text{high}} |v_1\rangle, |v_0\rangle) \cap \text{Iso}(|v_0\rangle, y^* B_{\text{high}} |v_1\rangle)\big) \end{aligned} \quad (18)$$

where $\text{Iso}(v, w)$ denotes the set of Pauli isomorphisms A which map $|v\rangle$ to $|w\rangle$ and we have denoted $\pi \cdot G := \{\pi \cdot g \mid g \in G\}$ for a set G and a single operator π. Lemma 13 shows that such an isomorphism set can be expressed in terms of the stabilizer group of $|v\rangle$.

▶ **Lemma 13.** *Let $|\varphi\rangle$ and $|\psi\rangle$ be quantum states on the same number of qubits. Let π be a Pauli isomorphism mapping $|\varphi\rangle$ to $|\psi\rangle$. Then the set of Pauli isomorphisms mapping $|\varphi\rangle$ to $|\psi\rangle$ is $\mathit{Iso}(|v\rangle, |w\rangle) = \pi \cdot \mathit{Stab}(|\varphi\rangle)$. That is, the set of isomorphisms $|\varphi\rangle \to |\psi\rangle$ is a coset of the stabilizer subgroup of $|\varphi\rangle$.*

Proof. If $P \in \text{Stab}(|\varphi\rangle)$, then $\pi \cdot P$ is an isomorphism since $\pi \cdot P |\varphi\rangle = \pi |\varphi\rangle = |\psi\rangle$. Conversely, if σ is a Pauli isomorphism which maps $|\varphi\rangle$ to $|\psi\rangle$, then $\pi^{-1}\sigma \in \text{Stab}(|\varphi\rangle)$ because $\pi^{-1}\sigma |\varphi\rangle = \pi^{-1} |\psi\rangle = |\varphi\rangle$. Therefore $\sigma = \pi(\pi^{-1}\sigma) \in \pi \cdot \text{Stab}(|\varphi\rangle)$. ◀

With Lemma 13 we can rewrite eq. (18) as

$$\begin{aligned}\text{Stab}(|v\rangle) = &\mathbb{I}_2 \otimes \underbrace{(\text{Stab}(|v_0\rangle) \cap \text{Stab}(B_{\text{high}} |v_1\rangle))}_{\text{stabilizer subgroup}} \\ &\cup Z \otimes \underbrace{(\mathbb{I} \cdot \text{Stab}(|v_0\rangle) \cap -\mathbb{I} \cdot \text{Stab}(B_{\text{high}} |v_1\rangle))}_{\text{isomorphism set}} \\ &\cup \bigcup_{y \in \{1,i\}} \begin{bmatrix} 0 & y \\ y^* & 0 \end{bmatrix} \otimes \underbrace{\big(\pi \cdot \text{Stab}(y^* B_{\text{high}} \cdot |v_1\rangle) \cap \pi^{-1} \cdot \text{Stab}(|v_0\rangle)\big)}_{\text{isomorphism set}} \end{aligned} \quad (19)$$

where π denotes a single isomorphism $y^* B_{\text{high}} |v_1\rangle \to |v_0\rangle$.

Given generating sets for $\text{Stab}(v_0)$ and $\text{Stab}(v_1)$, evaluating eq. (19) requires us to:

- **Compute $\text{Stab}(A|w\rangle)$ from $\text{Stab}(w)$ (as generating sets) for Pauli LIM A and node w.** It is straightforward to check that $\{AgA^\dagger \mid g \in G\}$, with $\langle G\rangle = \text{Stab}(w)$, is a generating set for $\text{Stab}(A|w\rangle)$.

- **Find a single isomorphism between two edges, pointing to reduced nodes.** In a reduced LIMDD, edges represent isomorphic states if and only if they point to the same nodes. This results in a straightforward algorithm, see Algorithm 10.

- **Find the intersection of two stabilizer subgroups, represented as generating sets G_0 and G_1 (Algorithm 11).** First, it is straightforward to show that the intersection of two stabilizer subgroups is again a stabilizer subgroup (it is never empty since $\mathbb{I}$ is a stabilizer of all states). Algorithm 11 will find a generating set G_U for the conjugated intersection of $\langle UG_0U^\dagger\rangle \cap \langle UG_1U^\dagger\rangle$ for a suitably chosen U, followed by returning $U^\dagger G_U U$ as a generating set for the target intersection $\langle G_0\rangle \cap \langle G_1\rangle$. As unitary U, we choose an n-qubit unitary U which maps G_0 to the generating set

$$UG_0U^\dagger = \{Z_1, Z_2, \dots, Z_{|G_0|}\}$$

where Z_k denotes a Z gate on qubit with index k, i.e.,

$$Z_k := \mathbb{I} \otimes \mathbb{I} \otimes \dots \otimes \mathbb{I} \otimes \underbrace{Z}_{\text{position k}} \otimes \mathbb{I} \otimes \dots \otimes \mathbb{I}.$$

Such a unitary always exists and can be found in time $O(n^3)$ using Algorithm 2 from [28]. It is not hard to see that the Pauli string of all LIMs in $\langle UG_0U^\dagger\rangle$ is a Z or $\mathbb{I}$. Therefore, to find the intersection of this group with $\langle UG_1U^\dagger\rangle$, we only need to bring $UG_1U^\dagger$ into RREF form (see Section 2), followed by discarding all generators in the RREF form whose pivot corresponds to an X or an Y, i.e. its pivot is a 1 in the X-block when representing a generator as a check vector (see Section 2). Both the resulting generator set (called H_1 in Algorithm 11) and $UG_0U^\dagger$ are subsets of the group of Pauli LIMs with scalars ± 1 and Pauli strings with only $\mathbb{I}$ and Z. These groups are finite and abelian. We use the Zassenhaus algorithm [1] to find a generating set H' for the intersection of $\langle H_1\rangle \cap \langle UG_0U^\dagger\rangle$ (in particular, the groups $\langle H_1\rangle$ and $\langle UG_0U^\dagger\rangle$ are group isomorphic to Boolean vector spaces, where addition corresponds to XOR-ing. Hence we may think of H_1 and $UG_0U^\dagger$ as bases of linear subspaces. The Zassenhaus algorithm computes a basis for the intersection of the two linear subspaces.) The final step is to perform the inverse conjugation map and return $U^\dagger H'U$. All of the above steps can be performed in $O(n^3)$ time; in particular, the operator U as found by Algorithm 2 from [28] consists of at most $O(n^2)$ Cliffords, each of which can be applied to a check matrix in time $O(n)$, yielding $O(n^3)$ time required for evaluating $G \mapsto UGU^\dagger$. Hence the overall runtime of Algorithm 11 is $O(n^3)$ also.

- `IntersectIsomorphismSets`: **Find the intersection of two isomorphism sets, represented as single isomorphism (π_0, π_1) with a generator set of a stabilizer subgroup (G_0, G_1), see Lemma 13.** This is the *coset intersection problem* for the PauliLIM$_n$ group. Isomorphism sets are coset of stabilizer groups (see Lemma 13) and it is not hard to see that that the intersection of two cosets, given as isomorphisms $\pi_{0/1}$ and generator sets $G_{0/1}$, is either empty, or a coset of $\langle G_0\rangle \cap \langle G_1\rangle$ (computed using Algorithm 11). Therefore, we only need to determine an isomorphism $\pi \in \pi_0\langle G_0\rangle \cap \pi_1\langle G_1\rangle$, or infer that no such isomorphism exists.
We solve this problem in $O(n^3)$ time in two steps (see Algorithm 12 for the full algorithm). First, we note that that $\pi_0\langle G_0\rangle \cap \pi_1\langle G_1\rangle = \pi_0[\langle G_0\rangle \cap (\pi_0^{-1}\pi_1)\langle G_1\rangle]$, so we only need to find an element of the coset $S := \langle G_0\rangle \cap (\pi_0^{-1}\pi_1)\langle G_1\rangle$. Now note that S is nonempty if and only if there exists $g_0 \in \langle G_0\rangle, g_1 \in \langle G_1\rangle$ such that $g_0 = \pi_0^{-1}\pi_1 g_1$, or, equivalently,

$\pi_0^{-1}\pi_1 \cdot g_1 \cdot g_0^{-1} = \mathbb{I}$. We show in Lemma 14 that such g_0, g_1 exist if and only if $\mathbb{I}$ is the smallest element in the set $S\pi_0^{-1}\pi_1 \langle G_1\rangle \cdot \langle G_0\rangle$. Hence, for finding out if S is empty we may invoke the LexMin algorithm we have already used before in GetLabels and we will explain below in Subsubsection 5.5.4. If it is not empty, then we obtain g_0, g_1 as above using ArgLexMin, and output $\pi_0 \cdot g_0$ as an element in the intersection. Since LexMin and ArgLexMin take $O(n^3)$ time, so does Algorithm 12.

▶ **Lemma 14.** *The coset $S := \langle G_0\rangle \cap \pi_1^{-1}\pi_0 \cdot \langle G_1\rangle$ is nonemtpy if and only if the lexicographically smallest element of the set $S = \pi_0^{-1}\pi_1 \langle G_1\rangle \cdot \langle G_0\rangle = \{\pi_0^{-1}\pi_1 g_1 g_0 | g_0 \in G_0, g_1 \in G_1\}$ is $1 \cdot \mathbb{I}$.*

Proof. (Direction $\rightarrow$) Suppose that the set $\langle G_0\rangle \cap \pi_0^{-1}\pi_1 \langle G_1\rangle$ has an element a. Then $a = g_0 = \pi_0^{-1}\pi_1 g_1$ for some $g_0 \in \langle G_0\rangle, g_1 \in \langle G_1\rangle$. We see that $\mathbb{I} = \pi_0^{-1}\pi_1 g_1 g_0^{-1} \in \pi_0^{-1}\pi_1 \langle G_1\rangle \cdot \langle G_0\rangle$, i.e., $\mathbb{I} \in S$. Note that $\mathbb{I}$ is, in particular, the lexicographically smallest element, since its check vector is the all-zero vector $(\vec{0}|\vec{0}|00)$.

(Direction $\leftarrow$) Suppose that $\mathbb{I} \in \pi_0^{-1}\pi_1 \langle G_1\rangle \cdot \langle G_0\rangle$. Then $\mathbb{I} = \pi_0^{-1}\pi_1 g_1 g_0$, for some $g_0 \in \langle G_0\rangle, g_1 \in \langle G_1\rangle$, so we get $g_0^{-1} = \pi_0^{-1}\pi_1 g_1 \in \langle G_0\rangle \cap \pi_0^{-1}\pi_1 \langle G_1\rangle$, as promised. ◀

The four algorithms above allow us to evaluate each of the four individual terms in eq. (19). To finish the evaluation of eq. (19), one would expect that it is also necessary that we find the union of isomorphism sets. However, we note that if πG is an isomorphism set, with π an isomorphism and G an stabilizer subgroup, then $P_n \otimes (\pi g) = (P_n \otimes \pi)(\mathbb{I}_2 \otimes g)$ for all $g \in G$. Therefore, we will evaluate eq. (19), i.e. find (a generating set) for all stabilizers of node v in two steps. First, we construct the generating set for the first term, i.e. $\mathbb{I}_2 \otimes (\text{Stab}(|v_0\rangle) \cap \text{Stab}(B_{\text{high}} |v_1\rangle))$, using the algorithms above. Next, for each of the other three terms $P_n \otimes (\pi G)$, we add only *a single* stabilizer of the form $P_n \otimes \pi$ for each $P_n \in \{X, Y, Z\}$. We give the full algorithm in Algorithm 9 and prove its efficiency below.

▶ **Lemma 15** (Efficiency of function `GetStabilizerGenSet`)**.** *Let v be an n-qubit node. Assume that generator set for the stabilizer subgroups of the children v_0, v_1 are known, e.g. by an earlier call to* `GetStabilizerGenSet`*, followed by caching the result (see Line 27 in Algorithm 9). Then Algorithm 9 (function* `GetStabilizerGenSet`*), applied to v, runs in time $O(n^3)$.*

Proof. If $n = 1$ then Algorithm 9 only evaluates Line 2–4, which run in constant time. For $n > 1$, the algorithm performs a constant number of calls to `GetIsomorphism` (which only multiplies two Pauli LIMs and therefore runs in time $O(n)$) and four calls to `IntersectIsomorphismSets`. Note that the function `IntersectIsomorphismSets` from Algorithm 12 invoke $O(n^3)$-runtime external algorithms (the Zassenhaus algorithm [1], RREF algorithm from Section 2, and Algorithm 2 from [28]), making its overall runtime $O(n^3)$ also. Therefore, `GetStabilizerGenSet` has runtime is $O(n^3)$. ◀

5.5.4 Efficiently finding a minimal LIM by multiplying with stabilizers

Here, we give $O(n^3)$ subroutines solving the following problem: given generators sets G_0, G_1 of stabilizer subgroups on n qubits, and an n-qubit Pauli LIM A, determine $\min_{(g_0,g_1)\in\langle G_0,G_1\rangle} A \cdot g_0 \cdot g_1$, and also find the g_0, g_1 which minimize the expression. We give an algorithm for

Algorithm 9 Algorithm for constructing the Pauli stabilizer subgroup of a Pauli-LIMDD node

```
1: procedure GetStabilizerGenSet(EDGE $e_0 \xrightarrow{\mathbb{1}_2^{\otimes n}} v_0, e_1 \xrightarrow{B_{\text{high}}} v_1$ with $v_0, v_1$ reduced)
2:     if n=1 then
3:         if there exists $P \in \pm 1 \cdot \{X, Y, Z\}$ such that $P |v\rangle = |v\rangle$ then return $P$
4:         else return None
5:     else
6:         if $v \in$ STABCACHE[$v$] then return STABCACHE[$v$]
7:         $G_0 :=$ GetStabilizerGenSet($v_0$)
8:         if $B_{\text{high}} = 0$ then
9:             return $\{\mathbb{I}_2 \otimes g,\ \mathbb{Z} \otimes g \mid g \in G_0\}$
10:        else
11:            $G := \emptyset$                    ▷ Add all automorphisms of the form $\mathbb{1}_2 \otimes \dots$ :
12:            $G_1 := \{A_1^\dagger g A_1 \mid g \in$ GetStabilizerGenSet($v_1$)$\}$
13:            $(\pi, B) :=$ IntersectIsomorphismSets($(\mathbb{1}_2^{\otimes n-1}, G_0), (\mathbb{1}_2^{\otimes n-1}, G_1)$)
14:            $G := G \cup \{\mathbb{I}_2 \otimes g \mid g \in B\}$
15:
16:            $\pi_0, \pi_1 := \mathbb{1}_2^{\otimes n-1}$, GetIsomorphism($e_1, -1 \cdot e_1$)
17:            $(\pi, B) :=$ IntersectIsomorphismSets($(\pi_0, G_0), (\pi_1, G_1)$)
18:            if $\pi \neq$ None then $G := G \cup \{Z \otimes \pi\}$          ▷ Add stabilizer of form $Z \otimes \dots$
19:
20:            $\pi_0, \pi_1 :=$ GetIsomorphism($e_0, e_1$), GetIsomorphism($e_1, e_0$)
21:            $(\pi, B) :=$ IntersectIsomorphismSets($(\pi_0, G_0), (\pi_1, G_1)$)
22:            if $\pi \neq$ None then $G := G \cup \{X \otimes \pi\}$          ▷ Add stabilizer of form $X \otimes \dots$
23:
24:            $\pi_0, \pi_1 :=$ GetIsomorphism($e_0, -i \cdot e_1$), GetIsomorphism($-i \cdot e_1, e_0$)
25:            $(\pi, B) :=$ IntersectIsomorphismSets($(\pi_0, G_0), (\pi_1, G_1)$)
26:            if $\pi \neq$ None then $G := G \cup \{Y \otimes \pi\}$          ▷ Add stabilizer of form $Y \otimes \dots$
27:        STABCACHE[$v$] := $G$
28:        return $G$
```

Algorithm 10 Algorithm for constructing a single isomorphism between two Pauli-LIMDD edges, each pointing to canonical nodes.

```
1: procedure GetIsomorphism(EDGE $\xrightarrow{A} v$, EDGE $\xrightarrow{B} w$ with $v, w$ reduced, $A \neq 0 \vee B \neq 0$)
2:     if $v = w \wedge A, B \neq 0$ then
3:         return $B \cdot A^{-1}$
4:     return None
```

Algorithm 11

```
1: procedure IntersectStabilizerGroups(stabilizer subgroup generating sets $G_0, G_1$)
   Output: a generating set for $\langle G_0\rangle \cap \langle G_1\rangle$
2:     Compute $U$ s.t. $H_0 := UG_0U^\dagger = \{Z_1, Z_2, \ldots, Z_{|G_0|}\}$, using Algorithm 2 from [28]
3:     $H_1 := UG_1U^\dagger$
4:     Bring $H_1$ into RREF form
5:     Discard any generators from $H_1$ whose check vector has a 1 in the $X$ block as pivot ▷ See also Section 2
6:     $H' :=$ generating set for $\langle H_0\rangle \cap \langle H_1\rangle$ ▷ Computed using the Zassenhaus algorithm for finding the intersection of vector subspaces
7:     return $U^\dagger H' U$
```

Algorithm 12 $O(n^3)$ algorithm for computing the intersection of two sets of isomorphisms, each given as single isomorphism with a stabilizer subgroup (see Lemma 13).

```
1: procedure IntersectIsomorphismSets(stabilizer subgroup generating sets $G_0, G_1$ and Pauli-LIMs $\pi_0, \pi_1$)
   Output: a Pauli LIM $\pi$ and a stabilizer subgroup generating set $G$ such that $\pi\langle G\rangle = \pi_0\langle G_0\rangle \cap \pi_1\langle G_1\rangle$
2:     $\pi := LexMin(G_0, G_1, \pi_1^{-1}\pi_0)$
3:     if $\pi = \mathbb{I}$ then
4:         $(g_0, g_1) = ArgLexMin(G_0, G_1, \pi_1^{-1}\pi_0)$
5:         $\pi := \pi_0 \cdot g_0$
6:         $G := IntersectStabilizerGroups(G_0, G_1)$
7:         return $(\pi, G)$
8:     else
9:         return None
```

finding both the minimum (LexMin) and the arguments of the minimum (ArgLexMin) in Algorithm 13. The inuition behind the algorithms are the following two steps: first, the lexicographically minimum Pauli LIM *modulo scalar* can easily be determined using the scalar-ignoring DivisionRemainder algorithm from Section 2. Since in the lexicographic ordering, the scalar is least significant (Section 2), the resulting Pauli LIM has the same Pauli string as the the minimal Pauli LIM *including scalar*. We show below in Lemma 16 that if the scalar-ignoring minimization results in a Pauli LIM λP, then the only other eligible LIM, if it exists, is $-\lambda P$. Hence, in the next step, we only need to determine whether such LIM $-\lambda P$ exists and whether $-\lambda < \lambda$; if so, then $-\lambda P$ is the real minimal Pauli LIM $\in \langle G_0 \cup G_1 \rangle$.

▶ **Lemma 16.** *Let v_0 and v_1 be LIMDD nodes, R a Pauli string and $\nu, \nu' \in \mathbb{C}$. Define $G = Stab(v_0) \cup Stab(v_1)$. If $\nu R, \nu' R \in \langle G \rangle$, then $\nu = \pm \nu'$.*

Proof. We prove $g \in \langle G \rangle \implies \pm ig \notin \langle G \rangle$, which is equivalent to the statement in the lemma because each product of stabilizers from different stabilizer subgroups has scalar ± 1 or $\pm i$ (follows from the facts that stabilizers hold scalar ± 1 and multiplying Pauli strings yields a scalar $\in \{\pm 1, \pm i\}$). To reach a contradiction, assume there exists a $g \in \langle G \rangle$ for which $\pm ig \in \langle G \rangle$ also. Since Pauli LIMs commute or anticommute, we can decompose both as $g = (-1)^x g_0 g_1$ and $\pm ig = (-1)^y h_0 h_1$ for some $x, y \in \{0, 1\}$ and $g_0, h_0 \in \text{Stab}(v_0)$ and $g_1, h_1 \in \text{Stab}(v_1)$. Combining yields $\pm i(-1)^x g_0 g_1 = (-1)^y h_0 h_1$, which we rewrite as $\pm i(-1)^{x+y} \underbrace{g_1 h_1^{-1}}_{\in \text{Stab}(v_1)} = \underbrace{g_0^{-1} h_0}_{\in \text{Stab}(v_0)}$. Squaring both sides yields the contradiction $-1 \cdot \mathbb{I} = \mathbb{I}$ where we used that $(g_1 h_1^{-1})^2 = (g_0^{-1} g_0)^2 = \mathbb{I}$, since stabilizers square to $\mathbb{I}$. ◀

The central procedure in Algorithm 13 is ArgLexMin, which, given a LIM A and sets G_0, G_1 which generate stabilizer groups, finds $g_0 \in \langle G_0 \rangle, g_1 \in \langle G_1 \rangle$ such that $A \cdot g_0 \cdot g_1$ reaches its lexicographic minimum over all choices of g_0, g_1. It first performs the scalar-ignoring minimization (Line 5) to find g_0, g_1 modulo scalar. The algorithm LexMin simply invokes ArgLexMin to get the arguments g_0, g_1 which yield the minimum and uses these to compute the actual minimum.

The subroutine FindOpposite finds an element $g \in G_0$ such that $-g \in G_0$, or infers that no such g exists. It does so in a similar fashion as IntersectStabilizerGroups from Subsubsection 5.5.3: by conjugation with a suitably chosen unitary U, it maps G_1 to $\{Z_1, Z_2, \ldots, Z_{|G_1|}\}$. Analogously to our explanation of IntersectStabilizerGroups, the group generated by $UG_1U^\dagger$ contains precisely all Pauli LIMs which satisfy the following three properties: (i) the scalar is 1; (ii) its Pauli string has an $\mathbb{I}$ or Z at positions $1, 2, \ldots, |G_1|$; (iii) its Pauli string has an $\mathbb{I}$ at positions $|G_1| + 1, \ldots, n$. Therefore, the target g only exists if there is a LIM in $\langle UG_0U^\dagger \rangle$ which (i') has scalar -1 and satisfies properties (ii) and (iii). To find such a g, we put $UG_0U^\dagger$ in RREF form and check all resulting generators for properties (i'), (ii) and (iii). (By definition of RREF, it suffices to check only the generators for this property) If a generator h satisfies these properties, we return $U^\dagger h U$ and `None` otherwise. The algorithm requires $O(n^3)$ time to find U, the conversion $G \mapsto UGU^\dagger$ can be done in time $O(n^3)$, and $O(n)$ time is required for checking each of the $O(n^2)$ generators. Hence the runtime of the overall algorithm is $O(n^3)$.

Algorithm 13 Algorithms LexMin and ArgLexMin for computing the minimal element from the set $A \cdot \langle G_0 \rangle \cdot \langle G_1 \rangle = \{Ag_0g_1 | g_0 \in G_0, g_1 \in G_1\}$, where A is a Pauli LIM and G_0, G_1 are generating sets for stabilizer subgroups. The algorithms make use of a subroutine FindOpposite for finding an element $g \in \langle G_0 \rangle$ such that $-g \in \langle G_1 \rangle$. A canonical choice for the RootLabel (see Section 5.4) of an edge e pointing to a node v is LexMin($G, \{\mathbb{I}\}, \mathsf{label}(e)$) where G is a stabilizer generator group of Stab(v).

```
1: procedure LexMin(stabilizer subgroup generating sets $G_0, G_1$ and Pauli LIM $A$)
   Output: $\min_{(g_0, g_1 \in \langle G_0 \cup G_1 \rangle} A \cdot g_0 \cdot g_1$
2:     $(g_0, g_1) :=$ ArgLexMin($G_0, G_1, A$)
3:     return $A \cdot g_0 \cdot g_1$

4: procedure ArgLexMin(stabilizer subgroup generating sets $G_0, G_1$ and Pauli LIM $A$)
   Output: $\arg\min_{g_0 \in G_0, g_1 \in G_1} A \cdot g_0 \cdot g_1$
5:     $(g_0, g_1) := \arg\min_{(g_0, g_1) \in \langle G_0 \cup G_1 \rangle} \{h \mid h \propto A \cdot g_0 \cdot g_1\}$    ▷ Using the scalar-ignoring DivisionRemainder algorithm from Section 2,
6:     $g' :=$ FindOpposite($G_0, G_1, g_0, g_1$)
7:     if $g'$ is None then
8:         return $(g_0, g_1)$
9:     else
10:        $h_0, h_1 := g_0 \cdot g', (-g') \cdot g_1$    ▷ $g_0 g_1 = -h_0 h_1$
11:        if $A \cdot h_0 \cdot h_1 <_{\text{lex}} A \cdot g_0 \cdot g_1$ then return $(h_0, h_1)$
12:        else return $(g_0, g_1)$

13: procedure FindOpposite(stabilizer subgroup generating sets $G_0, G_1$)
    Output: $g \in G_0$ such that $-g \in G_1$, or None if no such $g$ exists
14:     Compute $U$ s.t. $UG_1U^\dagger = \{Z_1, Z_2, \ldots, Z_{|G_1|}\}$, using Algorithm 2 from [28]    ▷ $Z_j$ is the $Z$ gate applied to qubit with index $j$
15:     $H_0 := UG_0U^\dagger$
16:     $H_0^{RREF} := H_0$ in RREF form
17:     for $h \in H_0^{RREF}$ do
18:         if $h$ satisfies all three of the following: (i) $h$ has scalar $-1$; the Pauli string of $h$ (ii) contains only $\mathbb{I}$ or $Z$ at positions $1, 2, \ldots, |G_1|$, and (iii) only $\mathbb{I}$ at positions $|G_1| + 1, \ldots, n$ then
19:             return $U^\dagger h U$
20:     return None
```

6 A use case for LIMDDs

In this section, we describe a family of quantum circuits we call "Hamming weight-controlled circuits," which can be simulated in polynomial time using LIMDDs. In contrast, we provide numerical evidence that the stabilizer rank of their output states grow rapidly in the number of qubits, indicating their hardness for stabilizer-rank-based simulation methods (see Section 2).

Given an n-qubit Clifford gate C, we define the $2n$-qubit Hamming weight-controlled Clifford gate (or HWC gate) U_w^C on computational-basis states $|x\rangle, |y\rangle$ (both n qubits) as

$$U_w^C(|x\rangle \otimes |y\rangle) = \begin{cases} |x\rangle \otimes C\,|y\rangle & \text{if } |x| = w \\ |x\rangle \otimes |y\rangle & \text{otherwise} \end{cases} \tag{20}$$

where $|x|$ denotes the Hamming weight of bitstring x. By Hamming weight-controlled Clifford (HWC) circuits, we denote a sequence of HWC gates applied to initial state $(H\,|0\rangle)^{\otimes n} \otimes |0\rangle^{\otimes n}$. Figure 10 (a) shows an example. We refer to the output state as a HWC state.

The LIMDD representing a HWC state containing t HWC gates has width $t+1 \leq n+1$, and therefore has $\mathcal{O}(n^2)$ nodes. Figure 10(b) shows an example. First, the division of length-n bitstrings depending on the Hamming weight is a known construction for BDDs [15], yielding a DD for the control register whose width which cannot exceed $n+1$. The target register consists of a series of states $C_w\,|0\rangle^n$, each of which is a stabilizer state because C_w is a Clifford operation. Since Pauli-LIMDDs can represent n-qubit stabilizer states with n nodes (Section 4), the resulting LIMDD for the entire HWC state has polynomially-many nodes. To simulate this circuit, we build a LIMDD for each individual Hamming weight-controlled Clifford gate in the Clifford circuits C_w. These LIMDDs have polynomial size and look similar ot those in Figure 10.

In order to investigate the hardness of HWC circuits for stabilizer rank methods, we employ the heuristic algorithm from [13] (explained in more detail in [17]) for searching for the stabilizer rank of Dicke states $|D_w^n\rangle$, which are equal superpositions of computational basis states with a given Hamming weight:

$$|D_w^n\rangle := \sum_{\substack{x \in \{0,1\}^n, \\ |x| = w}} |x\rangle\,.$$

To see that there exists HWC states $|\varphi\rangle$ which have stabilizer rank $\chi(|\varphi\rangle)$ at least as large as the stabilizer rank $\chi(|D_w^n\rangle)$of Dicke states, consider the HWC circuit consisting of the single gate U_w^C where C is the Pauli X gate on the least significant qubit in the target register. The resulting state is

$$|\varphi\rangle = \sum_{k=0, k\neq w}^{n} |D_k^n\rangle \otimes |0\rangle^{\otimes n} + |D_w^n\rangle\,|0\ldots01\rangle \tag{21}$$

By measuring the target register in the computational basis, the resulting control register state is $|D_w^n\rangle$ upon obtaining output $|0\ldots01\rangle$. Since computational-basis measurements cannot increase the stabilizer rank, we find $\chi(|\varphi\rangle) \geq \chi(|D_w^n\rangle)$.

The heuristic algorithm follows a simulated-annealing approach: on input n, w and χ, it performs a random walk through sets of χ stabilizer states. It starts with a random set V of χ stabilizer states on n qubits. In a single 'step', the algorithm picks one of these states $|\psi\rangle \in V$

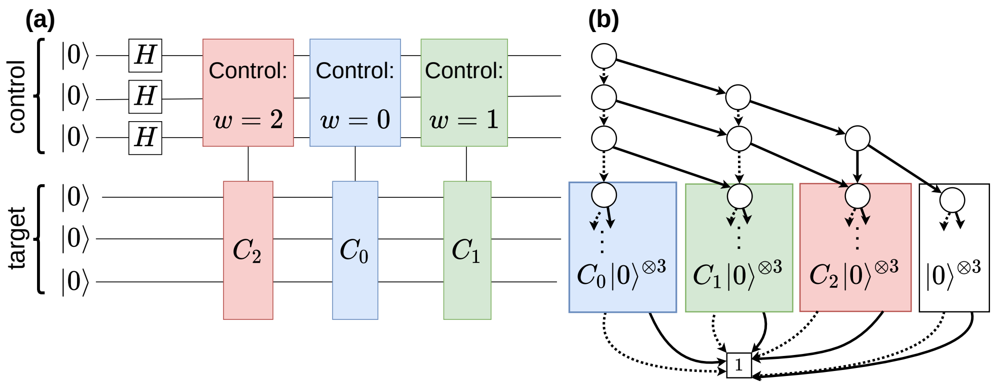


Figure 10 (a) A “Hamming weight-controlled Clifford circuit” on $3+3$ qubits with input state $|0\rangle^{\otimes 3} \otimes |0\rangle^{\otimes 3}$. The depicted circuit consists of three controlled-Clifford gates, where the Clifford C_w is only applied if the Hamming weight of the control register equals w see also eq. (20). **(b)** A Pauli-LIMDD for the output state of the circuit in (a), where the colored blocks represent Pauli-LIMDDs for the states $C_w |0\rangle^{\otimes 3}$. The edges which originate in the control-register-part of the LIMDD point to the top node of three different branches, each representing a constant Hamming weight of the control qubits (edge weights are all $\mathbb{I}$; recall dashed/solid lines represent low/high edges). The branch in the target-register-part corresponding to Hamming weight w represents the state $C_w |0\rangle^{\otimes 3}$, which is a stabilizer state. We note that stabilizer states are represented by a polynomially-large LIMDD (see Section 4) while the width of the control-register-part of the LIMDD can never exceed the possible number of Hamming weights on length-n bitstrings, i.e. $\binom{n}{2} = O(n^2)$. Consequently, Pauli-LIMDDs represent all Hamming weight-controlled Clifford circuits using only polynomially-many nodes.

at random, together with a random n-qubit Pauli operator P, and replaces the state $|\psi\rangle$ with $|\psi'\rangle := c(\mathbb{I} + P)|\psi\rangle$ with c a normalization constant (or repeats if $|\psi'\rangle = 0$), yielding a new set V'. The step is accepted with certainty if $F_V < F_{V'}$, where $F_V := |\langle |D^n_w\rangle | \Pi_V | D^n_w\rangle |$ with Π_V the projector on the subspace of the n-qubit Hilbert space spanned by the stabilizer

	Hamming weight w					
#qubits n	**0**	**1**	**2**	**3**	**4**	**5**
1	1					
2	1	1				
3	1	2				
4	1	2	2			
5	1	3	2			
6	1	3	4	2		
7	1	4	7	4		
8	1	4	8	≤ 11	5	
9						> 10?

Table 3 Heuristically-found upper bounds on the stabilizer rank χ of Dicke states $|D^n_w\rangle$ (eq. (21)) using the heuristic algorithm from Bravyi et al. [13] (see main text for details). Empty cells indicate non-existing or not-investigated states. In particular, we have not investigated $w > \lceil \frac{n}{2} \rceil$ since $\chi(|D^n_w\rangle) = \chi(|D^n_{n-w}\rangle)$ because $X^{\otimes n} |D^n_w\rangle = |D^n_{n-w}\rangle$. For $|D^8_3\rangle$ and $|D^9_5\rangle$, we have run the heuristic algorithm to find sets of stabilizers up to size 11 (theoretical upper bound) and 10, respectively, but the algorithm has not found sets in which these two Dicke states could be decomposed. We emphasize that the algorithm is heuristic, so if there exists a stabilizer decomposition of a given rank, the algorithm might not find it.

states in V. Otherwise, it is accepted with probability $\exp(-\beta(F_{V'} - F_V))$, where β should be interpreted as the inverse temperature. The algorithm terminates if it finds $F_V = 1$, implying that $|D^n_w\rangle$ can be written as linear combination of V, outputting the number χ as (upper bound to) the stabilizer rank. For a fixed χ, we use identical values to Bravyi et al. [13] and vary β from 1 to 4000 in 100 steps, performing 1000 steps at each value of β. See [2] for our open-source implementation.

We provide the results of our numerical runs in Table 3. Since our approach is based on using heuristic algorithms to find a good upper bound on the stabilizer rank, and not a lower bound, by construction we cannot guarantee any statement on the scaling of the rank itself. Our approach could only have provided evidence that Dicke states do not have a rapidly scaling rank, thereby providing evidence that stabilizer-rank methods can simulate HWC states efficiently. However, this is not what we observe, and our findings are consistent with the possibility that Dicke states have a superpolynomial rank. Although further research is needed for a definitive answer, these numerics strengthen our confidence that LIMDDs can simulate HWC circuits faster than stabilizer-rank based methods.

7 Discussion

We have introduced LIMDD, a novel decision diagram-based method to simulate quantum circuits, which enables polynomial-size representation of a strict superset of stabilizer states and of the states represented by polynomially-large QMDDs. To prove this, we have shown the first lower bounds on the size of QMDDs for stabilizer states: they are exponential-size for certain families of stabilizer states. LIMDDs achieve a more succinct representation by representing states up to local invertible maps which uses single qubit (local) operations from a group G. We have investigated the choices $G = \textsc{Pauli}$, $G = \langle Z \rangle$ and $G = \langle X \rangle$, and found that any choice suffices for an exponential advantage over QMDDs; notably, the choice $G = \textsc{Pauli}$ allows us to succinctly represent stabilizer states. We also defined reduction rules for Pauli-LIMDD and showed that for each set of quantum states which are equivalent under local Pauli operations, modulo normalization factor, has a unique reduced Pauli-LIMDD as representative.

Furthermore, we showed how to simulate arbitrary quantum circuits, encoded as Pauli-LIMDDs. The resulting algorithms are often faster than for QMDDs. In contrast to QMDDs, Clifford circuits (initialized to $|0\rangle$) can be simulated by Pauli-LIMDDs in polynomial time. This in itself is not an interesting feat, since efficient simulation methods for the stabilizer regime have been known since the Gottesman-Knill theorem. However, we also showed that Pauli-LIMDDs can efficiently simulate a circuit family we call Hamming weight-controlled Clifford circuits. And we provide empirical evidence that these circuits are hard for stabilizer-rank based simulation methods.

An obvious next step is to investigate other choices for G. Of interest are both the representational capabilities of such diagrams (do they represent interesting states?), and the algorithmic capabilities (can we still find efficient algorithms which make use of these diagrams?). In this vein, an important question is what the relationship is between G-LIMDDs (for various choices of G) and existing formalisms for the classical simulation of quantum circuits, such as those based on match-gates [45, 34, 31] and tensor networks [40]. It would also be interesting to compare LIMDDs to graphical depictions of quantum computation, following similar work for QMDDs [52]. We leave empirical evaluations into the consequences of LIMDD methods for efficient quantum circuit simulation, for future work. This would obviously include a comparison with stabilizer rank simulation [10].

Finally, we note that the current definition of LIMDD imposes a strict total order over the qubits along every path from root to leaf. It is known that the chosen order can greatly influence the size of the DD [43, 53], making it interesting to investigate variants of LIMDDs with a flexible ordering.

References

1 Some algorithms for nilpotent permutation groups. *Journal of Symbolic Computation*, 23(4):335–354, 1997. URL: `https://www.sciencedirect.com/science/article/pii/S0747717196900929`, `doi:https://doi.org/10.1006/jsco.1996.0092`.

2 Stabranksearcher: code for finding (upper bounds to) the stabilizer rank of a quantum state. `https://github.com/timcp/StabRankSearcher`, 2021.

3 Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), nov 2004. URL: `https://doi.org/10.1103%2Fphysreva.70.052328`, `doi:10.1103/physreva.70.052328`.

4 Sheldon B. Akers. Binary decision diagrams. *IEEE Computer Architecture Letters*, 27(06):509–516, 1978.

5 Henrik Reif Andersen. An introduction to binary decision diagrams. *Lecture notes, available online, IT University of Copenhagen*, page 5, 1997.

6 Koenraad M R Audenaert and Martin B Plenio. Entanglement on mixed stabilizer states: normal forms and reduction procedures. *New Journal of Physics*, 7(1):170, 2005. URL: `http://stacks.iop.org/1367-2630/7/i=1/a=170`.

7 R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 188–191, 1993.

8 Charles H Bennett, Herbert J Bernstein, Sandu Popescu, and Benjamin Schumacher. Concentrating partial entanglement by local operations. *Physical Review A*, 53(4):2046, 1996.

9 Karl S Brace, Richard L Rudell, and Randal E Bryant. Efficient implementation of a bdd package. In *27th ACM/IEEE design automation conference*, pages 40–45. IEEE, 1990.

10 Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum*, 3:181, September 2019. `doi:10.22331/q-2019-09-02-181`.

11 Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by clifford gates. *Phys. Rev. Lett.*, 116:250501, Jun 2016. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.116.250501`, `doi:10.1103/PhysRevLett.116.250501`.

12 Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Phys. Rev. A*, 71:022316, Feb 2005. URL: `https://link.aps.org/doi/10.1103/PhysRevA.71.022316`, `doi:10.1103/PhysRevA.71.022316`.

13 Sergey Bravyi, Graeme Smith, and John A. Smolin. Trading classical and quantum computational resources. *Phys. Rev. X*, 6:021043, Jun 2016. URL: `https://link.aps.org/doi/10.1103/PhysRevX.6.021043`, `doi:10.1103/PhysRevX.6.021043`.

14 Hans J. Briegel and Robert Raussendorf. Persistent entanglement in arrays of interacting particles. *Phys. Rev. Lett.*, 86:910–913, Jan 2001. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.86.910`, `doi:10.1103/PhysRevLett.86.910`.

15 Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

16 Yirng-An Chen Randal E Bryant. Verification of arithmetic circuits with binary moment diagrams. In *32nd Design Automation Conference*, pages 535–541. IEEE, 1995.

17 Padraic Calpin. *Exploring Quantum Computation Through the Lens of Classical Simulation*. PhD thesis, UCL (University College London), 2020.

18 Sagar Chaki and Arie Gurfinkel. Bdd-based symbolic model checking. In *Handbook of Model Checking*, pages 219–245. Springer, 2018.

19 Eric Chitambar, Debbie Leung, Laura Mančinska, Maris Ozols, and Andreas Winter. Everything you always wanted to know about locc (but were afraid to ask). *Communications in Mathematical Physics*, 328(1):303–326, 2014.

20 E. M. Clarke, K. L. McMillan, X Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, page 54?60, New York, NY, USA, 1993. Association for Computing Machinery. `doi:10.1145/157485.164569`.

21 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

22 Wolfgang Dür, Guifre Vidal, and J Ignacio Cirac. Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6):062314, 2000.

23 Pavol Ďuriš, Juraj Hromkovič, Stasys Jukna, Martin Sauerhoff, and Georg Schnitger. On multi-partition communication complexity. *Information and computation*, 194(1):49–75, 2004.

24 Matthias Englbrecht and Barbara Kraus. Symmetries and entanglement of stabilizer states. *Phys. Rev. A*, 101:062302, Jun 2020. URL: `https://link.aps.org/doi/10.1103/PhysRevA.101.062302`, `doi:10.1103/PhysRevA.101.062302`.

25 Hélene Fargier, Pierre Marquis, Alexandre Niveau, and Nicolas Schmidt. A knowledge compilation map for ordered real-valued decision diagrams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

26 David Y Feinstein and Mitchell A Thornton. On the skipped variables of quantum multiple-valued decision diagrams. In *2011 41st IEEE International Symposium on Multiple-Valued Logic*, pages 164–169. IEEE, 2011.

27 Masahiro Fujita, Patrick C. McGeer, and JC-Y Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal methods in system design*, 10(2-3):149–169, 1997.

28 Hector J Garcia, Igor L Markov, and Andrew W Cross. Efficient inner-product algorithm for stabilizer states. *arXiv preprint arXiv:1210.6646*, 2012.

29 Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv:quant-ph/9705052*, 1997.

30 Daniel Gottesman. The Heisenberg representation of quantum computers. *arXiv:quant-ph/9807006v1*, 1998.

31 Martin Hebenstreit, Richard Jozsa, Barbara Kraus, and Sergii Strelchuk. Computational power of matchgates with supplementary resources. *Physical Review A*, 102(5):052604, 2020.

32 Marc Hein, Wolfgang Dür, Jens Eisert, Robert Raussendorf, M Nest, and H-J Briegel. Entanglement in graph states and its applications. *arXiv:0602096*, 2006.

33 Yifei Huang and Peter Love. Approximate stabilizer rank and improved weak simulation of clifford-dominated circuits for qudits. *Phys. Rev. A*, 99:052307, May 2019. URL: `https://link.aps.org/doi/10.1103/PhysRevA.99.052307`, `doi:10.1103/PhysRevA.99.052307`.

34 Richard Jozsa and Akimasa Miyake. Matchgates and classical simulation of quantum circuits. *Proceedings: Mathematical, Physical and Engineering Sciences*, pages 3089–3106, 2008.

35 Lucas Kocia and Peter Love. Stationary phase method in discrete wigner functions and classical simulation of quantum circuits. *arXiv:1810.03622*, 2018.

36 Lucas Kocia and Mohan Sarovar. Improved simulation of quantum circuits by fewer gaussian eliminations. *arXiv:2003.01130*, 2020.

37 Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM journal on numerical analysis*, 16(2):346–358, 1979.

38 K.L. McMillan. *Symbolic model checking: an approach to the state explosion problem*. PhD thesis, 1992. UMI No. GAX92-24209.

39 D Michael Miller and Mitchell A Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. In *36th International Symposium on Multiple-Valued Logic (ISMVL'06)*, pages 30–30. IEEE, 2006.

40 Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014. URL: `https://www.sciencedirect.com/science/article/pii/S0003491614001596`, `doi:https://doi.org/10.1016/j.aop.2014.06.013`.

41 John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, 2018.

42 Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86:5188–5191, May 2001. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.86.5188`, `doi:10.1103/PhysRevLett.86.5188`.

43 Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 42–47. IEEE, 1993.

44 Scott Sanner and David McAllester. Affine algebraic decision diagrams (aadds) and their application to structured probabilistic inference. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1384–1390, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

45 Barbara M. Terhal and David P. DiVincenzo. Classical simulation of noninteracting-fermion quantum circuits. *Phys. Rev. A*, 65:032325, Mar 2002. URL: `https://link.aps.org/doi/10.1103/PhysRevA.65.032325`, `doi:10.1103/PhysRevA.65.032325`.

46 Ewout van den Berg and Kristan Temme. Circuit optimization of hamiltonian simulation by simultaneous diagonalization of pauli clusters. *Quantum*, 4:322, 2020.

47 Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Graphical description of the action of local clifford transformations on graph states. *Physical Review A*, 69(2):022316, 2004.

48 Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Local unitary versus local clifford equivalence of stabilizer states. *Phys. Rev. A*, 71:062323, Jun 2005. URL: `https://link.aps.org/doi/10.1103/PhysRevA.71.062323`, `doi:10.1103/PhysRevA.71.062323`.

49 George F Viamontes, Igor L Markov, and John P Hayes. Improving gate-level simulation of quantum circuits. *Quantum Information Processing*, 2(5):347–380, 2003.

50 George F Viamontes, Igor L Markov, and John P Hayes. High-performance quidd-based simulation of quantum circuits. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 1354–1355. IEEE, 2004.

51 George F Viamontes, Igor L Markov, and John P Hayes. *Quantum circuit simulation*. Springer Science & Business Media, 2009.

52 Renaud Vilmart. Quantum multiple-valued decision diagrams in graphical calculi. *arXiv:2107.01186*, 2021.

53 Ingo Wegener. *Branching programs and binary decision diagrams: theory and applications*. SIAM, 2000.

54 Alwin Zulehner and Robert Wille. One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(5):996–1008, 2017.

55 Alwin Zulehner and Robert Wille. Advanced simulation of quantum computations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(5):848–859, 2018.

A Proof that cluster states need exponentially-large QMDDs

In this appendix, we formally prove that cluster states have exponential size in QMDDs Lemma 8. We first fix notation and definitions, after which we prove the theorem using two lemmas.

Let G be an undirected graph with vertices $V_G = \{v_1, \ldots, v_n\}$ and edge set $E_G \subseteq V_G \times V_G$. For a subset of vertices $S \subseteq V_G$, the S-induced subgraph of G has vertices S and edge set $(S \times S) \cap E$. Given G, its graph state $|G\rangle$ is given in Equation 4 in Section 2 and can equivalently be expressed as

$$|G\rangle = \sum_{\vec{x} \in \{0,1\}^n} (-1)^{f_G(\vec{x})} |\vec{x}\rangle$$

where $f_G(\vec{x})$ is the number of edges in the S-induced subgraph of G.

For a function $f : \{0,1\}^n \to \mathbb{C}$ and bit string $\vec{a} = a_1 \cdots a_k \in \{0,1\}^k$, we denote by $f_{\vec{a}}$ the subfunction of f restricted to $\vec{a}$:

$$f_{\vec{a}}(x_{k+1}, \ldots, x_n) := f(a_1, \ldots, a_k, x_{k+1}, \ldots, x_n) \tag{22}$$

We also say that $f_{\vec{a}}$ is a subfunction of f of *order* $|\vec{a}| = k$.

We will also need the notions of boundary and strong matching.

▶ **Definition 17** (Boundary). *For a set $S \subseteq V_G$ of vertices in G, the* boundary *of S is the set of vertices in S adjacent to a vertex outside of S.*

▶ **Definition 18** (Strong Matching). *Let $G = (V, E)$ be an undirected graph. A* strong matching *is a subset of edges $M \subseteq E$ that do not share any vertices (i.e., it is a matching) and no two edges of M are incident to the same edge of G, i.e., an edge in $E \backslash M$. Alternatively, a strong matching is a matching M s.t. $G[V(M)] = M$. We say that M is an (S, T)-strong matching for two sets of vertices $S, T \subset V$ if $M \subseteq S \times T$. For a strong matching M and a vertex $v \in V(M)$, we let $M(v)$ denote the unique vertex to which v is matched by M.*

Using these definitions and notation, we prove Lemma 8.

Proof of Lemma 8. Let $G = \text{lattice}(n, n)$ be the undirected graph of the $n \times n$ lattice, with vertex set $V = \{v_1, \ldots, v_{n^2}\}$. Let $\sigma = v_1 v_2 \cdots v_{n^2}$ be a variable order, and let $S = \{v_1, v_2, \ldots, v_{\frac{1}{2}n^2}\} \subset V$ be the first $\frac{1}{2}n^2$ vertices in this order.

The proof proceeds broadly as follows. First, in Lemma 19, we show that any $(S, \overline{S})$-strong matching M effects $2^{|M|}$ different subfunctions of f_G. Second, Lemma 20 shows that the lattice contains a large $(S, \overline{S})$-strong matching for any choice of S. Put together, this will prove the lower bound on the number of QMDD nodes as in Lemma 8 by the fact that a QMDD for the grid graph state G has a node per unique subfunction of the function f_G. Figure 11 illustrates this setup for the 5×5 lattice.

▶ **Lemma 19.** *Let M be a non-empty $(S, \overline{S})$-strong matching for the vertex set S chosen above. If $\sigma = v_1 v_2 \cdots v_{n^2}$ is a variable order where all vertices in S appear before all vertices in $\overline{S}$, then $f_G(x_1, \ldots, x_{n^2})$ has $2^{|M|}$ different subfunctions of order $|S|$.*

Proof. Let $S_M := S \cap V(M)$ and $\overline{S}_M := \overline{S} \cap M$ be the sets of vertices that are involved in the strong matching. Write $\chi(x_1, \ldots, x_n)$ for the indicator function for vertices: $\chi(x_1, \ldots, x_n) := \{v_i \mid x_i = 1, i \in [n]\}$. Choose two different subsets $A, B \subseteq S_M$ and let $\vec{a} = \chi^{-1}(A)$ and $\vec{b} = \chi^{-1}(B)$ be the corresponding length-$|S|$ bit strings. These two strings induce the two subfunctions $f_{G,\vec{a}}$ and $f_{G,\vec{b}}$. We will show that these subfunctions differ in at least one point.

First, if $f_{G,\vec{a}}(0, \ldots, 0) \neq f_{G,\vec{b}}(0, \ldots, 0)$, then we are done. Otherwise, take a vertex $s \in A \oplus B$ and say w.l.o.g. that $s \in A \setminus B$. Let $t = M(s)$ be its partner in the strong matching. Then we have, $|E[A \cup \{t\}]| = |E[A]| + 1$ but $|E[B \cup \{t\}]| = |E[B]|$. Therefore we have

$$f_{G,\vec{a}}(0, \ldots, 0, x_t = 0, 0, \ldots, 0) \neq f_{G,\vec{a}}(0, \ldots, 0, x_t = 1, 0, \ldots, 0) \quad (23)$$
$$f_{G,\vec{b}}(0, \ldots, 0, x_t = 0, 0, \ldots, 0) = f_{G,\vec{b}}(0, \ldots, 0, x_t = 1, 0, \ldots, 0) \quad (24)$$

We see that each subset of S_M corresponds to a different subfunction of f_G. Since there are $2^{|M|}$ subsets of M, f_G has at least that many subfunctions. ◀

We now show that the $n \times n$ lattice contains a large enough strong matching.

▶ **Lemma 20.** *Let $S = \{v_1, \ldots, v_{\frac{1}{2}n^2}\}$ be a set of $\frac{1}{2}n^2$ vertices of the $n \times n$ lattice, as above. Then the graph contains a $(S, \overline{S})$-strong matching of size at least $\lfloor \frac{1}{12} n \rfloor$.*

Proof. Consider the boundary B_S of S. This set contains at least $n/3$ vertices, by Theorem 11 in [37]. Each vertex of the boundary of S has degree at most 4. It follows that there is a set of $\lfloor \frac{1}{4}|B_S| \rfloor$ vertices which share no neighbors. In particular, there is a set of $\lfloor \frac{1}{4}|B_S| \rfloor \geq \lfloor \frac{1}{12} n \rfloor$ vertices in B_S which share no neighbors in $\overline{S}$. ◀

Put together, every choice of half the vertices in the lattice yields a set with a boundary of at least $n/3$ nodes, which yields a strong matching of at least $\lfloor \frac{1}{12} n \rfloor$ edges, which shows that f_G has at least $2^{\lfloor \frac{1}{12} n \rfloor}$ subfunctions of order $\frac{1}{2}n^2$. ◀

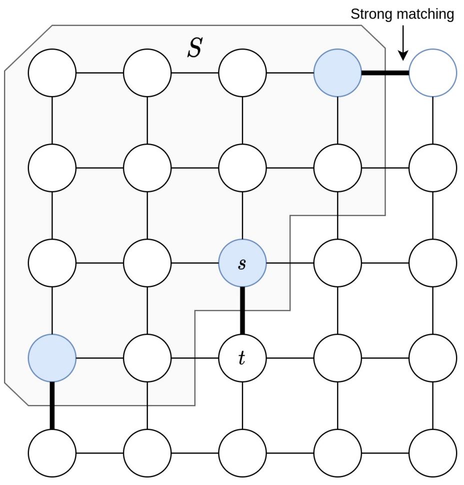


Figure 11 The 5×5 grid graph, partitioned in a vertex set S and its complement $\overline{S}$. A strong matching between S and $\overline{S}$ is indicated by black edges.

A.1 Proof that XOR states need exponentially large QMDDs

We now show that QMDDs which represent so-called XOR states are exponentially large in the worst case. On the other hand, in Appendix B, we will show that these states can be represented using only $\mathcal{O}(n)$ nodes by $\langle X\rangle$-LIMDDs, showing that they are exponentially more succinct than QMDDs. We define XOR states as uniform superpositions over vectors spaces, as follows.

▶ **Definition 21** (XOR-state). *Let $V \subseteq \{0,1\}^n$ be a vector space, i.e., for every $x, y \in V$ it holds that $x \oplus y \in V$. Then the XOR-state $|V\rangle$ is the uniform superposition over the elements of V, i.e.,*

$$|V\rangle = \frac{1}{\sqrt{|V|}} \sum_{x\in V} |x\rangle \tag{25}$$

We will use the following result by Ďuriš et al. on binary decision diagrams (BDDs), which are ADDs with codomain $\{0,1\}$.

▶ **Theorem 22** (Ďuriš et al.[23]). *The characteristic function $f_V : \{0,1\}^n \to \{0,1\}$ of a randomly chosen vector space V in $\{0,1\}^n$, defined as $f_V(x) = 1$ if $x \in V$ and 0 otherwise, needs a BDD of size $2^{\Omega(n)}/(2n)$ with high probability.*

Our result follows by noting that if f is as above, or indeed if f is any function with codomain $\{0, \lambda\}$ for any scalar λ, then the QMDD of the state $|f\rangle = \sum_x f(x) |x\rangle$ has the same structure as the BDD of f. That is to say, in this case, the BDD and QMDD are graphs with the same number of nodes.

▶ **Corollary 23.** *For a random vector space $V \subseteq \{0,1\}^n$, the XOR-state $|V\rangle$ requires QMDDs of size $2^{\Omega(n)}/(2n)$ with high probability.*

Proof. A BDD encodes a function $f\colon \{0,1\}^n \to \{0,1\}$. In this case, the BDD encodes f_V, the characteristic function of V. A BDD is a graph which contains one node for each subfunction of f.

Similarly, a QMDD representing a state $|\varphi\rangle = \sum_x f(x) |x\rangle$ can be said to represent the function $f\colon \{0,1\}^n \to \mathbb{C}$, and contains one node for each subfunction of f modulo scalars. In this case, it represents the function $1/\sqrt{|V|} f_V \colon \{0,1\}^n \to \{0, 1/\sqrt{|V|}\}$. However, in the case of f_V, two distinct subfunctions are not equal up to a scalar, because the codomain is $\{0,1\}$. To this end, let $f_{V,a}, f_{V,b}$ be distinct subfunctions of f_V induced by partial assignments $a, b \in \{0,1\}^k$. We will show that there is no $\lambda \in \mathbb{C}^*$ such that $f_{V,a} = \lambda f_{V,b}$. To see this, say that the two subfunctions differ in the point $x \in \{0,1\}^{n-k}$, i.e., $f_{V,a}(x) \neq f_{V,b}(x)$. Say without loss of generality that $f_{V,a}(x) = 0$ and $f_{V,b}(x) = 1$. Then, since $\lambda \neq 0$, we have $\lambda = \lambda f_{S,b}(x) \neq f_{V,a}(x) = 0$, so $f_{V,a} \neq \lambda f_{B,b}$.

Because distinct subfunctions of f_V are not equal up to a scalar, the QMDD for $|V\rangle$ contains a node for every subfunction of f_V. We conclude that, since by Theorem 22 with high probability the BDD representing f_V has exponentially many nodes, so does the QMDD representing $|V\rangle$. ◀

B How to write graph states, XOR-states and stabilizer states as Tower-LIMDDs

In this appendix, we prove that the families of $\langle Z\rangle$-, $\langle X\rangle$-, and Pauli-Tower-LIMDDs correspond to graph states, XOR states (see Definition 21 below), and stabilizer states, respectively, in Proposition 24, Proposition 26 and Theorem 30 below. Definition 3 for reduced PAULI-LIMDDs also holds when exchanging PAULI with $\langle X\rangle$. However, it does not work for $\langle Z\rangle$ by O3. Note however, that our proofs do not rely on the reduced definition, but on Definition 2.

We recall that a Tower-LIMDD representing an n-qubit state is a LIMDD which has, besides the leaf, n nodes.

▶ **Proposition 24** (Graph states are $\langle Z\rangle$-Tower-LIMDDs). *Let $n \geq 1$. Denote by $\mathcal{G}_n$ the set of n-qubit graph states and write $\mathcal{Z}_n$ for the set of n-qubit quantum states which are represented by Tower-LIMDDs which low-edge-labels $\mathbb{I}$ and high-edge labels $\lambda \bigotimes_j P_j$ with $P_j \in \{\mathbb{I}_2, Z\}$ and $\lambda \in \{0, 1\}$. Then $\mathcal{G}_n = \mathcal{Z}_n$.*

Proof. We establish $\mathcal{G}_n \subseteq \mathcal{Z}_n$ by providing a procedure to convert any graph state in $\mathcal{G}_n$ to a reduced Tower-LIMDD in $\mathcal{Z}_n$. See Figure 12 for an example of a 4-qubit graph state.

Base case: $n = 1$. We note that there is only one single-qubit graph state by definition (see Equation 4), which is $|+\rangle := (|0\rangle + |1\rangle)/\sqrt{2}$ and can be represented as LIMDD by a single node (in addition to the leaf node): see Figure 12(a).

Induction case. For the inductive step, we consider an $(n + 1)$-qubit graph state $|G\rangle$ corresponding to the graph G. We isolate the $(n + 1)$-th qubit by decomposing the full state definition from Equation 4 according to Equation 2:

$$|G\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |G_{1..n}\rangle + |1\rangle \otimes \underbrace{\left[\bigotimes_{(n+1,j)\in E} Z_j\right]}_{\text{Isomorphism B}} |G_{1..n}\rangle\right) \tag{26}$$

where E is the edge set of G and $G_{1..n}$ is the induced subgraph of G on vertices 1 to n. Thus, $|G_{1..n}\rangle$ is an n-qubit graph state on qubits 1 to n. Since $|G_{1..n}\rangle$ is a graph state on n qubits, by the induction hypothesis, we have a procedure to convert it to a Tower-LIMDD $\in \mathcal{Z}_{n-1}$. Now we construct a Tower-LIMDD for $|G\rangle$ as follows. The root node has two outgoing edges, both going to the node representing $|G_{1..n}\rangle$. The node's low edge is labeled with $\mathbb{I}$, and the node's high edge is labeled $B = 0$ if the $(n + 1)$-th qubit is isolated, and otherwise with

$$B = \bigotimes_{(n+1,j)\in E} Z_j \tag{27}$$

Thus the root node represents the state $|0\rangle |G_{1..n}\rangle + |1\rangle B |G_{1..n}\rangle$, satisfying Equation 26.

To prove $\mathcal{Z}_n \subseteq \mathcal{G}_n$, consider the fact that the reverse algorithm is simply the interpretation of the resulting root node $|v\rangle$ (see semantics in Definition 2). A simple counting argument based on the above construction shows that $|\mathcal{Z}_n| = |\mathcal{G}_n| = 2^{\binom{n}{2}}$, so the conversion is indeed a bijection. ◀

We now define "XOR-states" and prove that they are represented exactly by Tower-$\langle X\rangle$-LIMDDs.

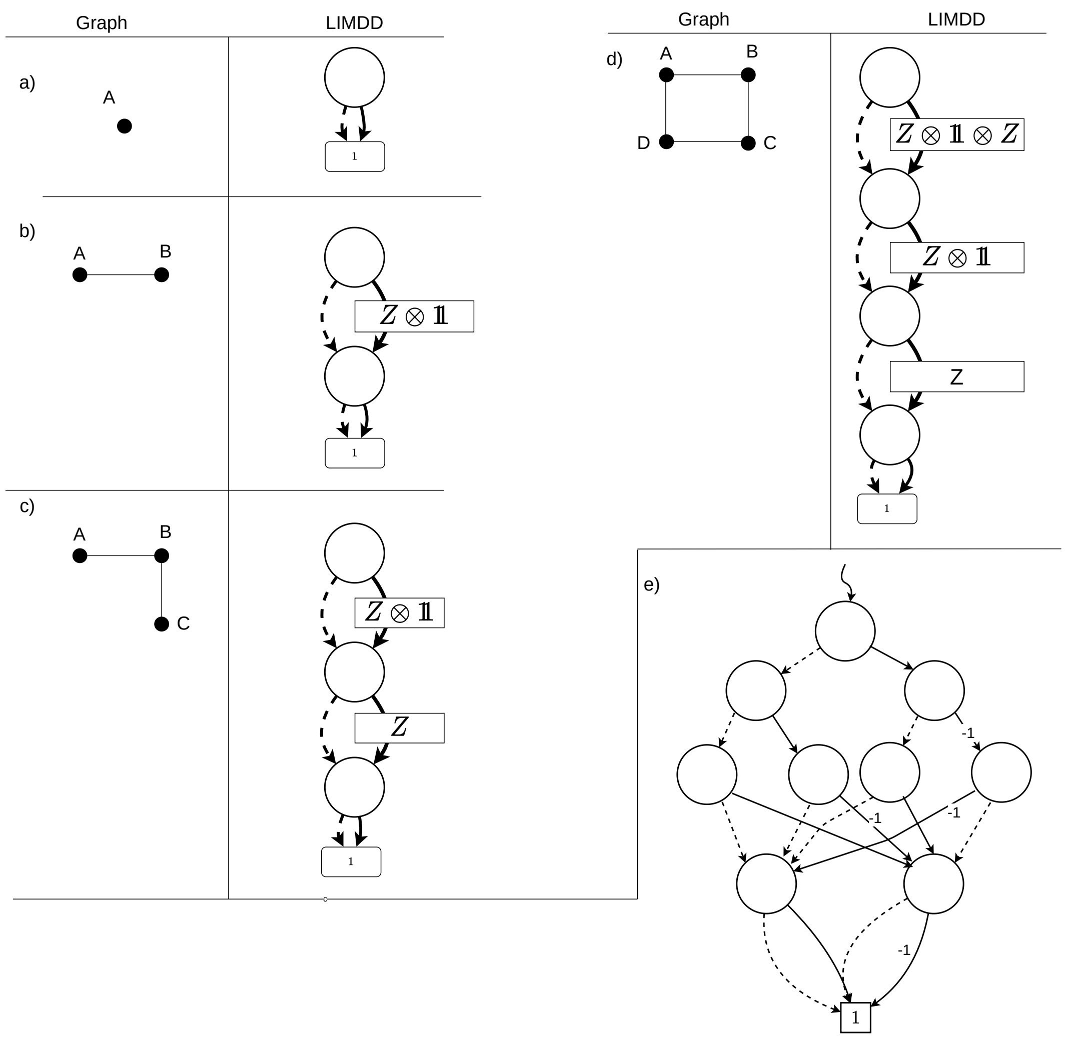


Figure 12 Construction of the Tower-LIMDD for the 4-qubit cluster graph state, by iterating over the number of vertices in the graph. (a) First, we consider the single-qubit graph state, which corresponds to a the subgraph containing only vertex A. (b) Then, we add vertex B, which is connected to A by an edge. The resulting LIMDD is constructed from the LIMDD from (a) by adding a new root node and an isomorphism node. The isomorphism is $\mathbb{1}_A \otimes Z_B$, since vertex C is connected to vertex B (yielding the Z operator) but not to A (yielding the identity operator $\mathbb{1}$). This process is repeated for a third vertex C (c) until we reach the LIMDD of the full 4-qubit cluster graph state (d). For comparison, (e) depicts a QMDD for the same graph state, which has width 4 instead of 1 for the LIMDD.

▶ **Definition 25** (XOR state). *Let $V \subseteq \{0,1\}^n$ be a vector space over $\{0,1\}$, i.e., V is not empty and it holds that $u, v \in V$ implies $u \oplus v \in V$. Then the following quantum state is the corresponding XOR state,*

$$|S\rangle = \frac{1}{\sqrt{|S|}} \sum_{x \in S} |x\rangle \tag{28}$$

▶ **Proposition 26** (XOR-states are $\langle X\rangle$-Tower-LIMDDs). *Let $n \geq 1$. Denote by $\mathcal{V}_n$ the set of n-qubit XOR states and write $\mathcal{X}_n$ for the set of n-qubit quantum states which are represented by Tower-LIMDDs with low edge labels $\mathbb{I}$ and high edge labels $\lambda \bigotimes_j P_j$ with $P_j \in \{\mathbb{I}, X\}$ and $\lambda \in \{0,1\}$. Then $\mathcal{V}_n = \mathcal{X}_n$.*

Proof. We prove $\mathcal{V}_n \subseteq \mathcal{X}_n$ by providing a procedure for constructing a Tower-LIMDD for an XOR-state. The procedure is recursive on the number of qubits.

Base case: $n = 1$. In this case, there are two XOR states: $|0\rangle$ and $(|0\rangle + |1\rangle)/\sqrt{2}$, which are represented by a single node which has a low and high edge pointing to the leaf node with low/high edge labels 1/0 and 1/1, respectively.

Induction case. Now consider an $(n+1)$-qubit XOR state $|S\rangle$ for a vector space $S \subseteq \{0,1\}^{n+1}$ for some $n \geq 1$ and assume we have a procedure to convert any n-qubit XOR state into a Tower-LIMDD in $\mathcal{X}_n$. We consider two cases, depending on whether the first bit of each element of S is zero:

(a) The first bit of each element of S is 0. Thus, we can write $S = \{0x \mid x \in S_0\}$ for some set $S_0 \subseteq \{0,1\}^n$. Then $0a, 0b \in S \implies 0a \oplus 0b \in S$ implies $a, b \in S_0 \implies a \oplus b \in S_0$ and thus S_0 is an length-n bit string vector space. Thus by assumption, we have a procedure to convert it to a Tower-LIMDD in $\mathcal{X}_n$. Convert it into a Tower-LIMDD in $\mathcal{X}_{n+1}$ for $|S\rangle$ by adding a fresh node on top with low edge label $\mathbb{I}_2^{\otimes n}$ and high edge label 0, both pointing to the the root S.

(b) There is some length-n bit string u such that $1u \in S$. Write S as the union of the sets $\{0x \mid x \in S_0\}$ and $\{1x \mid x \in S_1\}$ for sets $S_0, S_1 \subseteq \{0,1\}^n$. Since S is closed under element-wise XOR, we have $1u \oplus 1x = 0(u \oplus x) \in S$ for each $x \in S_1$ and therefore $u \oplus x \in S_0$ for each $x \in S_1$. This implies that $S_1 = \{u \oplus x \mid x \in S_0\}$ and thus S is the union of $\{0x \mid x \in S_0\}$ and $\{1u \oplus 0x \mid x \in S_0\}$. By similar reasoning as in case (a), we can show that S_0 is a vector space on length-n bit strings.
We build a Tower-LIMDD for $|S\rangle$ as follows. By the induction hypothesis, there is a Tower-LIMDD with root node v which represents $|v\rangle = |S_0\rangle$. We construct a new node whose two outgoing edges both go to this node v. Its low edge has label $\mathbb{I}_2^{\otimes n}$ and its high edge has label $P = P_n \otimes \cdots \otimes P_1$ where $P_j = X$ if $u_j = 1$ and $P_j = \mathbb{I}$ if $u_j = 0$.

We now show $\mathcal{V}_n \subseteq \mathcal{X}_n$, also by induction.

Base case: $n = 1$. There are only two Tower-LIMDDs on 1 qubit satisfying the description above, namely

(1) A node whose two edges point to the leaf. Its low edge has label 1, and its high edge has label 0. This node represents the XOR state $|0\rangle$, corresponding to the vector space $V = \{0\} \subseteq \{0,1\}^1$.

(2) A node whose two edges point to the leaf. Its low edge has label 1 and its high edge also has label 1. This node represents the XOR state $|0\rangle + |1\rangle$, corresponding to the vector space $V = \{0,1\}$.

Induction case. Let v be the root node of a Tower-LIMDD as described above. We distinguish two cases, depending on whether v's high edge has label 0 or not.

(a) The high edge has label 0. Then $|v\rangle = |0\rangle |v_0\rangle$ for a node v_0, which represents a XOR state $|v_0\rangle$ by the induction hypothesis.

(b) the high edge has label $\pi = P_n \otimes \cdots \otimes P_1$ with $P_j \in \{\mathbb{I}_2, X\}$. Then $|v\rangle = |0\rangle |v_0\rangle + |1\rangle \otimes \pi |v_0\rangle$. By the observations above, this is a XOR state, corresponding to the vector space $V = \{0x | x \in V_0\} \cup \{1(ux) | x \in V_0\}$ where $u_j = 1$ if $P_j = X$ and $u_j = 0$ if $P_j = \mathbb{I}_2$, and V_0 is the vector space corresponding to the XOR state $|v_0\rangle$.

◀

Lastly, we prove the stabilizer-state case. Specifically, we now define Stabilizer LIMDDs, and then we show that they represent exactly the set of stabilizer states. For this, we first need Lemma 28 and Lemma 29, which state that, if one applies a Clifford gate to a Stabilizer LIMDD, the resulting state is another Stabilizer LIMDD.

▶ **Definition 27** (Stabilizer LIMDD). *A Stabilizer-LIMDD is a Tower Pauli-LIMDD (i.e., each node has exactly one unique child), in which each node is semi-reduced, and each node's high edge's label is of the form λP with $\lambda \in \{0, \pm 1, \pm i\}$ and P is a Pauli string.*

▶ **Lemma 28.** *Let $|\varphi\rangle$ be an n-qubit state which is represented by a Stabilizer LIMDD. Let U be either a Hadamard gate or S gate on the top qubit (n-th qubit), or a CNOT with the top qubit as control. Then $U|\varphi\rangle$ is still represented by a semi-reduced Pauli-Tower-LIMDD.*

Proof. We use induction on the number of qubits n.

Base case: $n = 1$. There are six states represented by Stabilizer LIMDDs on 1 qubit, corresponding to the states $|0\rangle, |1\rangle$, and $|0\rangle + \alpha|1\rangle$ with $\alpha \in \{\pm 1, \pm i\}$. The gates H and S permute these six states, therefore $U|\varphi\rangle$ is represented by a Stabilizer LIMDD.

Induction case. We first consider $U = S$ and U =CNOT: if $U = S$, then the high edge of the top node is multiplied with i, while a downward CNOT (target qubit with index k) updates the high edge label $A \mapsto X_k A$. Both yield a Stabilizer LIMDD. Finally, for the Hadamard, we decompose $|\varphi\rangle = |0\rangle \otimes |\psi\rangle + \alpha |1\rangle \otimes P|\psi\rangle$ for some $(n-1)$-qubit stabilizer state $|\psi\rangle$, $\alpha \in \{0, \pm 1, \pm i\}$ and P is an $(n-1)$-qubit Pauli string. Now we note that $H|\varphi\rangle \propto |0\rangle \otimes |\psi_0\rangle + |1\rangle \otimes |\psi_1\rangle$ where $|\psi_x\rangle := (\mathbb{I} + (-1)^x \alpha P)|\psi\rangle$ with $x \in \{0, 1\}$. Now we consider two cases:

- there exist a stabilizer g of $|\psi\rangle$ which anticommutes with P. We note two things. First, $\langle\psi|P|\psi\rangle = \langle\psi|Pg|\psi\rangle = \langle\psi|g \cdot (-P)|\psi\rangle = -\langle\psi|P|\psi\rangle$, hence $\langle\psi|P|\psi\rangle = 0$. It follows by Lemma 15 of [28] that $|\psi_x\rangle$ is a stabilizer state, so by the induction hypothesis it can be written as a Stabilizer LIMDD. We denote the root node of this LIMDD by v. Next, we note that $g|\psi_0\rangle = g(\mathbb{I}_2 + \alpha P)|\psi\rangle = (\mathbb{I}_2 - \alpha P)g|\psi\rangle = |\psi_1\rangle$. Hence, $\begin{tikzcd} v & \bigcirc \arrow[l, dotted, "\mathbb{I}"'] \arrow[r, "g"] & v \end{tikzcd}$ is the root node of a Stabilizer LIMDD for $H|\varphi\rangle$.
- all stabilizers of $|\psi\rangle$ commute with P. Then $(-1)^y P$ is a stabilizer of $|\psi\rangle$ for either $y = 0$ or $y = 1$. Hence, $|\psi_x\rangle = (\mathbb{I} + (-1)^x \alpha P)|\psi\rangle = (1 + (-1)^{x+y}\alpha)|\psi\rangle$. Therefore, $|\varphi\rangle = |a\rangle \otimes |\psi\rangle$ where $|a\rangle := \big((1 + (-1)^y \alpha)|0\rangle + (1 + (-1)^{y+1}\alpha|1\rangle)\big)$. It is not hard to see that $|a\rangle$ is a stabilizer state for all choices of $\alpha \in \{0, \pm 1, \pm i\}$. By the induction hypothesis, both $|a\rangle$ and $|\psi\rangle$ can be represented as Stabilizer LIMDDs. We construct a Stabilizer LIMDD for $H|\varphi\rangle$ by replacing the leaf of the LIMDD of $|a\rangle$ by the root node of

the LIMDD of $|\psi\rangle$, and propagating the root edge label of $|\psi\rangle$ upwards. Specifically, if the root edge of $|a\rangle$ is $\xrightarrow{A}\textcircled{v}$ with $v = \textcircled{1}\overset{1}{\dashleftarrow}\bigcirc\xrightarrow{\beta}\textcircled{1}$, and if the root edge of $|\psi\rangle$ is $\xrightarrow{B}\textcircled{w}$, then a Stabilizer LIMDD for $H|\varphi\rangle$ has root node $\textcircled{$\mathbb{I}$}\overset{w}{\dashleftarrow}\bigcirc\xrightarrow{\beta\mathbb{I}}\textcircled{w}$ and has root edge label $A \otimes B$.

◀

▶ **Lemma 29.** *Let $|\varphi\rangle$ be an n-qubit state state represented by a Stabilizer LIMDD, and let U be either a Hadamard gate, an S gate or a CNOT gate. Then $U|\varphi\rangle$ is a state which is also represented by a Stabilizer LIMDD.*

Proof. The proof is by induction on n. The case $n = 1$ is covered by Lemma 28. Suppose that the induction hypothesis holds, and let $|\varphi\rangle$ be an $n+1$-qubit state represented by a Stabilizer LIMDD. First, we note that a CNOT gate CX_c^t can be written as $CX_c^t = (H \otimes H)CX_t^c(H \otimes H)$, so wlog we may assume that $c > t$. We treat two cases, depending on whether U affects the top qubit or not.

- U affects the top qubit. Then $U|\varphi\rangle$ is represented by a Stabilizer LIMDD, according to Lemma 28.
- U does not affect the top qubit. Suppose $|\varphi\rangle = |0\rangle \otimes |\varphi_0\rangle + |1\rangle \otimes \alpha P|\varphi_0\rangle$ (with P a Pauli string and $\alpha \in \{0, \pm 1, \pm i\}$). Then $U|\varphi\rangle = |0\rangle \otimes U|\varphi_0\rangle + |1\rangle \otimes (\alpha UPU^\dagger)U|\varphi_0\rangle$. Since U is either a Hadamard, S gate or CNOT, and $|\varphi_0\rangle$ is an n-qubit state, the induction hypothesis states that the state $U|\varphi_0\rangle$ is represented by a Stabilizer LIMDD. Let $\xrightarrow{A}\textcircled{v}$ be the root edge of this Stabilizer LIMDD, representing $U|\varphi_0\rangle$. Then $U|\varphi\rangle$ is represented by the root edge $\xrightarrow{\mathbb{I}\otimes A}\textcircled{w}$, where w is the node $\textcircled{v}\overset{\mathbb{I}}{\dashleftarrow}\bigcirc\xrightarrow{\alpha A^{-1}UPU^\dagger A}\textcircled{v}$. The label $\alpha A^{-1}UPU^\dagger A$ is a Pauli string, and may therefore be used as the label on the high edge of w.

◀

Finally, we show that stabilizer states are precisely the Pauli-Tower-LIMDDs.

▶ **Theorem 30** (Stabilizer states are Stabilizer LIMDDs)**.** *Let $n \geq 1$. Each n-qubit stabilizer state is represented by Stabilizer LIMDD with n nodes. Conversely, every Stabilizer LIMDD represents a stabilizer state.*

Proof. We first prove that each stabilizer state is represented by a Pauli-Tower-LIMDD. We recall that each stabilizer state can be obtained as the output state of a Clifford circuit on input state $|0\rangle^{\otimes n}$. Each Clifford circuit can be decomposed into solely the gates H, S and CNOT. The state $|0\rangle^{\otimes n}$ is represented by a Stabilizer LIMDD. According to Lemma 29, applying an H, S or CNOT gate to a Stabilizer LIMDD results a state represented by another Stabilizer LIMDD. One can therefore apply the gates of a Clifford circuit to the initial state $|0\rangle$, and obtain a Stabilizer LIMDD for every intermediate state, including the output state. Therefore, every stabilizer state is represented by a Stabilizer LIMDD.

For the converse direction, the proof is by induction on n. We only need to note that a state represented by a Pauli-Tower-LIMDD can be written as $|\varphi\rangle = |0\rangle \otimes |\varphi_0\rangle + |1\rangle \otimes \alpha P|\varphi_0\rangle = C(P)(|0\rangle + \alpha|1\rangle) \otimes |\varphi_0\rangle$ where $C(P) := |0\rangle\langle 0| \otimes \mathbb{I} + |1\rangle\langle 1| \otimes P$ is the controlled-(P) gate. Using the relations $Z = HXH$ and $Y = SXS^\dagger$, we can decompose $C(P)$ as CNOT, H and S, hence $C(P)$ is a Clifford gate. Since both $|0\rangle + \alpha|1\rangle$ and $|\varphi_0\rangle$ can be written as

Algorithm 14 Compute the probability of observing $|y\rangle$ when measuring the k-th qubit of the state $|e\rangle$. Here e is given as LIMDD on n qubits, y is given as a bit, and k is an integer index. For example, to measure the top-most qubit, one calls MEASURE$(e, 0, n)$. The procedure SQUAREDNORM(e, y, k) computes the scalar $\langle e| (\mathbb{I} \otimes |y\rangle \langle y| \otimes \mathbb{I}) |e\rangle$, i.e., computes the squared norm of the state $|e\rangle$ after the k-th qubit is projected to $|y\rangle$. We omit dynamic programming here only for readability.

```
1:  procedure MEASUREMENTPROBABILITY(EDGE $e \xrightarrow{\lambda P_n \otimes P'} (v)$, $y \in \{0,1\}$, $k \in Z_{\geq 1}$)
2:      if $n = k$ then
3:          $p_0 :=$ SQUAREDNORM(FOLLOW$_0(e)$)
4:          $p_1 :=$ SQUAREDNORM(FOLLOW$_1(e)$)
5:          return $p_j/(p_0 + p_1)$ where $j = 0$ if $P_n \in \{\mathbb{I}, Z\}$ and $j = 1$ if $P_n \in \{X, Y\}$
6:      else
7:          $p_0 :=$ SQUAREDNORMPROJECTED(FOLLOW$_0(e), y, k$)
8:          $p_1 :=$ SQUAREDNORMPROJECTED(FOLLOW$_1(e), y, k$)
9:          return $(p_0 + p_1)/$SQUAREDNORM$(e)$
10: procedure SQUAREDNORM(EDGE $\xrightarrow{\lambda P} (v)$)
11:     if $n = 0$ then return $|\lambda|^2$
12:     if $v \in$ NORM-CACHE then
13:     $s :=$ ADD(SQUAREDNORM(FOLLOW$_0(\xrightarrow{\mathbb{I}} (v))$), SQUAREDNORM(FOLLOW$_1(\xrightarrow{\mathbb{I}} (v))$))
14:     return $|\lambda|^2 s$
15: procedure SQUAREDNORMPROJECTED(EDGE $e \xrightarrow{\lambda P_n \otimes P'} (v)$, $y \in \{0,1\}$, $k \in \mathbb{Z}_{\geq 1}$)
16:     $b := (P_n \in \{X, Y\})$                    ▷ i.e., $b = 1$ iff $P_n$ is Anti-diagonal
17:     if $n = 0$ then
18:         return $|\lambda|^2$
19:     else if $n = k$ then
20:         return SQUAREDNORM(FOLLOW$_{b \oplus y}(e)$)
21:     else
22:         $\alpha_0 :=$ SQUAREDNORMPROJECTED(FOLLOW$_0(\xrightarrow{\mathbb{I}} (v)), b \oplus y, k$)
23:         $\alpha_1 :=$ SQUAREDNORMPROJECTED(FOLLOW$_1(\xrightarrow{\mathbb{I}} (v)), b \oplus y, k$)
24:         return $|\lambda|^2 \cdot (\alpha_0 + \alpha_1)$
```

Pauli-Tower-LIMDDs, they are stabilizer states by the induction hypothesis. Combined with the fact that $C(P)$ is a Clifford gate, we infer that $|\varphi\rangle$ is a stabilizer state. ◀

C Advanced algorithms

C.1 Measuring an arbitrary qubit

Algorithm 14 allows one to measure a given qubit. Specifically, given a qubit index k and an outcome $y \in \{0, 1\}$, it computes the probability of observing $|y\rangle$ when measuring the k-th significant qubit. The algorithm proceeds by traversing the LIMDD with root edge e at Line 7. Like Algorithm 3 which measured the top qubit, this algorithm computes a squared norm. The case that is added, relative to Algorithm 3, is the case when $n > k$, in which case it calls the procedure SQUAREDNORMPROJECTED. On input e, y, k, the procedure SQUAREDNORMPROJECTED outputs the squared norm of $\Pi_k^y |e\rangle$, where Π_k^y is the projector which projects the k-th qubit onto $|y\rangle$.

After measurement of a qubit k, a quantum state is typically projected to $|0\rangle$ or $|1\rangle$ ($b = 0$ or $b = 1$) on that qubit, depending on the outcome. Algorithm 15 realizes this. It does so by traversing the LIMDD until a node v with $\mathsf{idx}(v) = k$ is reached. It then returns an edge to a new node by calling $\textsc{MakeEdge}(\mathsf{low}(v), 0)$ for projection to $|0\rangle$ and $\textsc{MakeEdge}(0, \mathsf{high}(v))$ on Line 5, recreating nodes level k in the backtrack on Line 7. The projection operator $O \triangleq (\mathbb{I}_{n-k} \otimes |b\rangle\langle b| \otimes \mathbb{I}_{k-1})$ commutes with any LIM A with diagonal operator P_k ($\mathbb{I}_2$ or Z). For the anti-diagonal Pauli operators, we have $O \cdot A|v\rangle = A \cdot (\mathbb{I}_{n-k} \otimes |1-b\rangle\langle 1-b| \otimes \mathbb{I}_{k-1})|v\rangle$. The algorithm corrects for this on Line 2.

Algorithm 15 Project LIMDD $\xrightarrow{A}\textcircled{v}$ to $|b\rangle$ for qubit k, i.e., $(\mathbb{I}_{n-k} \otimes |b\rangle\langle b| \otimes \mathbb{I}_{k-1}) \cdot |Av\rangle$.

```
1: procedure UpdatePostMeas(Edge $\xrightarrow{\lambda P_n \otimes .. \otimes P_1}\textcircled{v}$, $k \leq n = \mathsf{idx}(v)$, $b \in \{0,1\}$)
2:     $b' := x \oplus b$ where $x = 0$ if $P_k \in \{\mathbb{I}, Z\}$ and $x = 1$ if $P_k \in \{X, Y\}$    ▷ flip $b$ if $P_k$ is anti-diagonal
3:     if $(v, k, b') \in$ Cache then return Cache$[v, k, b']$
4:     if $n = k$ then
5:         $e :=$ MakeEdge$((1-b') \cdot \mathsf{low}(v),\ b' \cdot \mathsf{high}(v))$   ▷ Project $|v\rangle$ to $|b'\rangle\langle b'| \otimes \mathbb{I}_2^{\otimes n-1}$
6:     else                                                        ▷ $n \neq k$:
7:         $e :=$ MakeEdge(UpdatePostMeas$(\mathsf{low}(v), k, b')$, UpdatePostMeas$(\mathsf{high}(v), k, b')$)
8:     Cache$[v, k, b'] := e$
9:     return $e$
```

C.2 Hadamards on stabilizer states in polynomial time

We show that, using the algorithms that we have given, a Hadamard can be applied to a stabilizer state in polynomial time. We emphasize that our algorithms do not invoke existing algorithms dedicated to applying a Hadamard to a stabilizer state; instead, the LIMDD algorithms are inherently polynomial-time for this use case.

▶ **Theorem 31.** *Algorithm 4, which applies a Hadamard gate to a stabilizer state represented as a Pauli-LIMDD, runs in polynomial time.*

Proof. Let $|\varphi\rangle$ be a stabilizer state, represented by a LIMDD with root edge $\xrightarrow{P}\textcircled{v}$. By Theorem 30, this LIMDD is a Tower Pauli-LIMDD. Algorithm 4 makes two calls to Add; both are of the form $\textsc{Add}(\xrightarrow{\lambda P}\textcircled{v}, \xrightarrow{\lambda Q v}\textcircled{})$ where $\lambda \in \mathbb{C}$ is a scalar, P and Q are Pauli strings, and v is a node representing a stabilizer state. Corollary 33 tells us that at most $8n$ recursive calls to Add are made. Each recursive call to Add may invoke the MakeEdge procedure, which runs in time $\mathcal{O}(n^3)$, yielding a total worst-case running time of $\mathcal{O}(n^4)$. ◀

▶ **Lemma 32.** *Let v be a node in a Tower Pauli-LIMDD representing a stabilizer state, and P a Pauli string. Then a call to* $\textsc{Add}(\xrightarrow{\mathbb{I}}\textcircled{v}, \xrightarrow{P}\textcircled{v})$ *invokes only $\mathcal{O}(n)$ recursive calls to* Add.

Proof. Let $v_0, \ldots, v_n$ be the nodes in the Tower Pauli-limdd, with $v = v_n$ the top node and v_0 the Leaf node. Let Q^k be the label on the high edge from v_{k+1} to v_k, for $k = 1 \ldots n$. Let $P = P_n \otimes \cdots \otimes P_1$ and write $P^{(k)} = P_k \otimes \cdots \otimes P_1$, so that, e.g., $P = P_n \otimes P_{n-1} \otimes P^{(n-2)}$.

We will show that, upon calling $\textsc{Add}(\xrightarrow{\mathbb{I}}\textcircled{v}, \xrightarrow{P}\textcircled{v})$, the recursive calls to Add are all of the form either $\textsc{Add}(\xrightarrow{\mathbb{I}}\textcircled{v_k}, \xrightarrow{\lambda P^{(k)}}\textcircled{v_k})$, or $\textsc{Add}(\xrightarrow{Q^k}\textcircled{v_k}, \xrightarrow{\lambda P^{(k)} Q^k}\textcircled{v_k})$ with $\lambda \in \{\pm 1, \pm i\}$. Let us first note how the lemma would follow from this fact. If these are the only ways in which the algorithm is called, then there are only $8n$ different parameters with which the algorithm is invoked. Because the algorithm uses a cache in which it stores the result of each computation, it will not descend into recursive calls after a cache hit. Therefore, only at most $8n$ recursive calls to Add are made after the initial call.

First, suppose the algorithm is invoked as $\textsc{Add}(\xrightarrow{\mathbb{I}}\textcircled{v_k}, \xrightarrow{\lambda P^{(k)}}\textcircled{v_k})$. If this yields a cache hit, the algorithm halts without recursing; otherwise, the following two recursive calls to Add are made,

$$\textsc{Add}(\textsc{follow}_0(\xrightarrow{\mathbb{I}}\textcircled{v_k}), \textsc{follow}_0(\xrightarrow{P^{(k)}}\textcircled{v_k})) \tag{29}$$

$$\text{and } \textsc{Add}(\textsc{follow}_1(\xrightarrow{\mathbb{I}}\textcircled{v_k}), \textsc{follow}_1(\xrightarrow{P^{(k)}}\textcircled{v_k})) \tag{30}$$

We note that $\textsc{follow}_0(\xrightarrow{\mathbb{I}}\textcircled{v_k}) = \xrightarrow{\mathbb{I}}\textcircled{v_{k-1}}$ and $\textsc{follow}_1(\xrightarrow{\mathbb{I}}\textcircled{v_k}) = \xrightarrow{Q^{k-1}}\textcircled{v_{k-1}}$. The value of $\textsc{follow}_b(\xrightarrow{P^{(k)}}\textcircled{v_k})$ depends on the value of P_k, so we distinguish four cases.

(a) **Case** $P_n = \mathbb{I}$. Then $P\,|v_k\rangle = |0\rangle\, P^{(k-1)}\,|v_{k-1}\rangle + |1\rangle\, P^{(k-1)}\,|v_{k-1}\rangle$, so we have $\textsc{follow}_0(\xrightarrow{P}\textcircled{v}) = P^{(k-1)}\,|v_{k-1}\rangle$ and $\textsc{follow}_1(\xrightarrow{P^{(k)}}\textcircled{v}) = P^{(k-1)}\,|v_{k-1}\rangle$, as above.

(b) **Case** $P_n = X$. Then $P\,|v_k\rangle = |0\rangle\, P^{(k-1)}Q^n\,|v_{k-1}\rangle + |1\rangle\, P^{(k-1)}\,|v_{k-1}\rangle$, so we have $\textsc{follow}_0(\xrightarrow{P^{(k)}}\textcircled{v_k}) = \xrightarrow{P^{(k-1)}Q^k}\textcircled{v_{k-1}}$ and $\textsc{follow}_1(\xrightarrow{P^{(k)}}\textcircled{v_k}) = \xrightarrow{P^{(k-1)}}\textcircled{v_{k-1}}$.

(c) **Case** $P_n = Y$. Then $P\,|v_n\rangle = -i\,|0\rangle\, P^{k-1}\,|v_{k-1}\rangle + i\,|1\rangle\, P^{(k-1)}Q^{k-1}\,|v_{k-1}\rangle$, so we have $\textsc{follow}_0(\xrightarrow{P^{(k)}}\textcircled{v_k}) = \xrightarrow{-iP^{(k-1)}}\textcircled{v_{k-1}}$ and $\textsc{follow}_1(\xrightarrow{P^{(k)}}\textcircled{v_k}) = \xrightarrow{iP^{(k-1)}Q^k}\textcircled{v_{k-1}}$.

(d) **Case** $P_n = Z$. Then $P\,|v_k\rangle = |0\rangle\, P^{(k-1)}\,|v_{k-1}\rangle - |1\rangle\, P^{(k-1)}Q^{k-1}\,|v_{k-1}\rangle$, so we have $\textsc{follow}_0(\xrightarrow{P^{(k)}}\textcircled{v_k}) = \xrightarrow{P^{(k-1)}}\textcircled{v_{k-1}}$ and $\textsc{follow}_1(\xrightarrow{P^{(k)}}\textcircled{v_k}) = \xrightarrow{-P^{(k-1)}Q^{k-1}}\textcircled{v_{k-1}}$.

In each case, indeed the two recursive calls to Add of Equations 29 and 30 are of the form indicated.

Next, suppose that the algorithm is called as $\textsc{Add}(\xrightarrow{Q^k}\textcircled{v_k}, \xrightarrow{\lambda P^{(k)}Q^k}\textcircled{v_k})$. Then, on line 4 of the Add algorithm, a LIM C is built such that $C\,|v_k\rangle = \lambda Q^{k,-1}P^{(k)}Q^k\,|v_k\rangle$. Then the algorithm proceeds as though it was called with $\textsc{Add}(\xrightarrow{\mathbb{I}}\textcircled{v_k}, \xrightarrow{C}\textcircled{v_k})$. The LIM C satisfies $C\,|v_k\rangle = \pm\lambda P^{(k)}\,|v_k\rangle$. By the observations above, this case, too, will yield only recursive calls to Add of the specified form. ◀

▶ **Corollary 33.** *Let v be a node in a Tower Pauli-LIMDD representing a stabilizer state, P, Q Pauli strings, and $\lambda \in \mathbb{C}$ a scalar. Then a call to $\textsc{Add}(\xrightarrow{\lambda P}\textcircled{v}, \xrightarrow{\lambda Q}\textcircled{v})$ invokes only $\mathcal{O}(n)$ recursive calls to Add.*

Proof. If $P = \mathbb{I}$, then this is the same as Lemma 32. Otherwise, the Add procedure factors out the LIM λP and proceeds as though it were called as $\textsc{Add}(\xrightarrow{\mathbb{I}}\textcircled{v}, \xrightarrow{PQ}\textcircled{v})$, which is the case treated by Lemma 32. ◀