Data structures for quantum circuit verification and how to compare them

Vinkhuijzen, L.T.

Chapter 5

A Quantum Knowledge Compilation Map

Quantum computing is finding promising applications in optimization, machine learning and physics, leading to the development of various models for representing quantum information. Because these representations are often studied in different contexts (many-body physics, machine learning, formal verification, simulation), little is known about fundamental trade-offs between their succinctness and the runtime of operations to update them. We therefore analytically investigate three widely-used quantum state representations: matrix product states (MPS), decision diagrams (DDs), and restricted Boltzmann machines (RBMs). We map the relative succinctness of these data structures and provide the complexity for relevant query and manipulation operations. Further, to chart the balance between succinctness and operation efficiency, we extend the concept of rapidity with support for non-canonical data structures, so that we can study several popular such data structures in this chapter. In particular, we show that MPS is at least as rapid as some DDs.

The aim of this chapter is to contribute to Research Question 2:

> **Research question 2.** *How can we analytically compare the relative strengths of data structures which represent quantum states?*

This chapter contributes to this question by articulating what we believe to be the most informative criteria on which such data structures should be compared: rapidity, succinctness, and tractability, *in that order*. It presents an exposition of such a comparison by systematically comparing several popular data structures on these criteria. In particular, we provide an easy-to-use framework for answering which of two data structures has better *rapidity*. By providing a knowledge compilation map for quantum state representation, this chapter contributes to our understanding of the inherent time and space efficiency trade-offs in this area.

5.1 Introduction

Quantum computing is showing great potential in various artificial intelligence endeavors, spanning tasks such as information retrieval [343], machine learning [234,358] and satisfiability [280]. In the other direction, AI is also crucial in quantum circuit compilation [57], quantum error correction [238,307] and quantum state learning [358]. Hence, we need good classical models for quantum information.

Various classical data structures have been proposed for representing quantum information, including decision diagrams (DDs; [227, 331]), tensor networks (TNs; [284]), matrix product states (MPS; [248]) also called tensor trains —a specialization of TNs— and restricted Boltzmann machines (RBMs; [104]) with complex values, a specialization of neural quantum states [78]. Recently, we have seen new applications for RBM [246], MPS [335], and also combinations of TNs and DDs [163], MPS and decision diagrams [72], TNs and probabilistic graphical models [133], and DDs with quantum circuits (LIMDDs; [337], introduced in Chapter 3).

However, fundamental differences between these data structures have not yet been studied in detail. This choice between different structures introduces an important trade-off between *size* and *speed*, i.e., how much space the data structure uses, versus how fast certain operations, such as measurement, can be performed. This trade-off plays a crucial role in other areas as well, and has already been illuminated for the domain of explainable AI [23] and logic [97,113], but not yet for quantum information.

In this chapter, we analytically compare for the first time several data structures for representing and manipulating quantum states, motivated by the following three applications. First, *simulating quantum circuits*, the building block of quantum com-

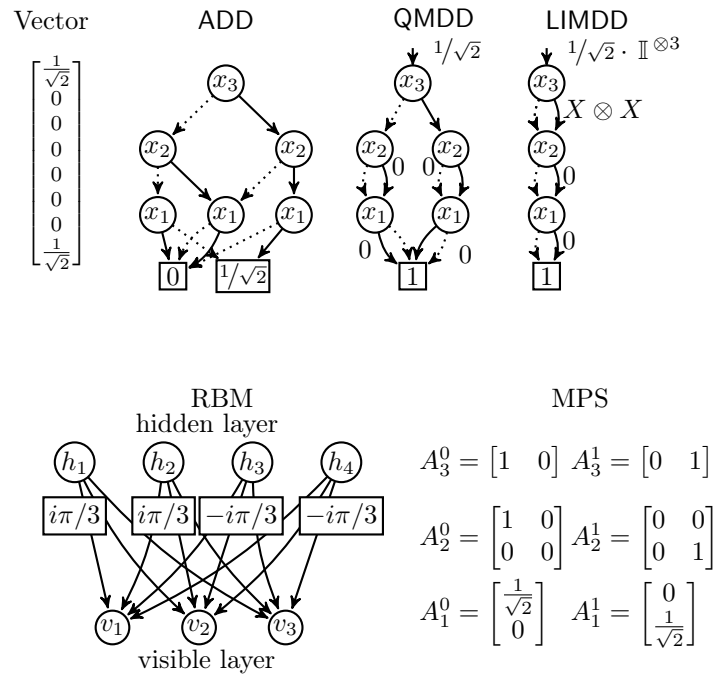


Figure 5.1: The 3-qubit GHZ state $1/\sqrt{2}(|000\rangle + |111\rangle)$, displayed using different data structures. The unlabelled edges for ADD, QMDD, LIMDD have resp. label 1, 1, $\mathbb{I}$. In the RBM, the weights of edges incident to h_1, h_2 (h_3, h_4) are all $i\pi/3$ ($-i\pi/3$); the hidden node biases $(\beta_{h_1}, \beta_{h_2}, \beta_{h_3}, \beta_{h_4}) = i\pi \cdot (1/3, 2/3, -1/3, -2/3)$; the visible node biases $\alpha_{v_1} = \alpha_{v_2} = \alpha_{v_3} = 0$.

putations, is a crucial tool for predicting the performance and scaling behavior of experimental devices with various error sources, thereby guiding hardware development [312, 374]. Next, *variational methods* are the core of *quantum machine learning* [37, 106] and solve quantum physics questions such as finding the lowest energy of a system of quantum particles [78, 122]. Finally, *verifying* if two quantum circuits are equivalent is crucial for checking if a (synthesized or optimized) quantum circuit satisfies its specification [18, 75, 127].

We focus on data structures developed for the above applications. In particular, we focus on: algebraic decision diagrams (ADD), semiring-labeled decision diagrams (QMDD, also called QMDD), local invertible map decision diagrams (LIMDD), matrix product states (MPS) and restricted Boltzmann machines (RBM). For those, we study *succinctness*, *tractability* and *rapidity*:

Succinctness & Tractability. In Section 5.3, we find that the succinctness of MPS, RBM and LIMDD are incomparable. We also find that MPS is strictly more succinct than QMDD, whereas previous research had suggested that these were incomparable [72]. In Section 5.4, we give states which QMDD and LIMDD can compactly represent, but which take exponential time to apply Hadamard and swap gates. Finally, we prove that computing fidelity and inner product in RBM and LIMDD is intractable assuming the exponential time hypothesis.
Rapidity. Considering tractability and succinctness separately can deceive; for instance, when representing a quantum state as an amplitude vector, most operations are polynomial time (i.e., tractable), but this is only because this vector has exponential size in the number of qubits. To mend this deficiency, [194] introduced the notion of *rapidity*, which reflects the idea that exponential operations may be preferable if a representation can be exponentially more succinct.

In Section 5.5, we generalize the definition of rapidity for non-canonical data structures, which allows us to study rapidity of MPS and RBM. We also give a simple sufficient condition for data structures to be more rapid than others (for all operations). We use it to settle several rapidity relations, showing surprisingly that MPS is strictly more rapid than QMDD. Since we are unaware of a previous comparison, our knowledge was consistent with them being incomparable.

5.2 Data structures for quantum states

This section gives the preliminaries that are necessary to understand this chapter. We present a more elaborate introduction to quantum computing in Section 2.2 and an introduction to decision diagrams in Section 2.3 – we note that those texts do not treat matrix product states or restricted Boltzmann machines.

5.2.1 Quantum information

A quantum state φ on n quantum bits or qubits, written in Dirac notation as $|\varphi\rangle$, can be represented as a vector of 2^n complex *amplitudes* $a_k \in \mathbb{C}$ such that the vector has unit norm, i.e. $\sum_{k=1}^{2^n} |a_k|^2 = 1$, where $|z| = \sqrt{a^2 + b^2}$ is the modulus of a complex number $z = a + b \cdot i$ with $a, b \in \mathbb{R}$. We also write $a_k = \langle k|\varphi\rangle$. Two quantum states $|\varphi\rangle, |\psi\rangle$ are equivalent if $|\varphi\rangle = \lambda |\psi\rangle$ for some complex λ. The inner product between

two quantum states $|\varphi\rangle, |\psi\rangle$ is $\langle\varphi|\psi\rangle = \sum_{k=1}^{2^n} (\langle k|\varphi\rangle)^* \cdot \langle k|\psi\rangle$ where $z^* = a - bi$ is the complex conjugate of the complex number $z = a + b \cdot i$. The fidelity $|\langle\varphi|\psi\rangle|^2 \in [0, 1]$ is a measure of closeness, with fidelity equalling 1 if and only if $|\varphi\rangle$ and $|\psi\rangle$ are equivalent.

A quantum circuit consists of gates and measurements. An n-qubit *gate* is a $2^n \times 2^n$ unitary matrix which updates the n-qubit state vector by matrix-vector multiplication. A k-local gate is a gate which effectively only acts on a subregister of $k \leq n$ qubits. Often-used gates go by names Hadamard (H), T, Swap, and the Pauli gates X, Y, Z. Together with the two-qubit gate controlled-Z, the H and T gate form a universal gate set (i.e. any quantum gate can be arbitrarily well approximated by circuits of these gates only). Next, a (computational-basis) measurement is an operation which samples n bits from (a probability distribution defined by) the quantum state, *while also updating the state.*

We now elaborate on the application domains from Section 5.1. *Simulating a circuit* using the data structures in this chapter starts wlog by constructing a data structure for some (simple) initial state $|\varphi_0\rangle$, followed by manipulating the data structure by applying the gates and measurements $A, B, ...$ of the circuit one by one, i.e.: $|\varphi_1\rangle = A|\varphi_0\rangle$, $|\varphi_2\rangle = B|\varphi_1\rangle$, etc. The data structure supports strong (weak) simulation, if, for each measurement gate, it can produce (a sample of) the probability function (as a side result of the state update). Next, the quantum circuit of *variational methods* consist of a quantum circuit; then the output state $|\varphi\rangle$ is used to compute $\langle\varphi|O|\varphi\rangle$ for some linear operator O (an *observable*). This computation reduces to $\langle\varphi|\psi\rangle$ with $|\psi\rangle := O|\varphi\rangle$ (i.e., computing simulation and fidelity). Last, *circuit verification* relies on checking approximate or exact equivalence of quantum states. This extends to unitary matrices (gates), which all data structures from this chapter can represent also, but which we will not treat for simplicity.

5.2.2 Data structures

We now define the data structures for representing quantum states considered in this chapter, with examples in Figure 5.1.

Definition 5.1 (Inspired by [113])**.** A *quantum-state representation (language)* is a tuple $(D, n, |.\rangle, |.|)$ where D is a set of data structures. Given $\alpha \in D$, $|\alpha\rangle$ is the (possibly unnormalized) quantum state it represents (i.e., the interpretation of α), $|\alpha|$ is the size of the data structure, and $n(\alpha)$, or n in short, is the number of qubits of $|\alpha\rangle$.

Finally, each quantum state should be expressible in the language.

We will often refer to a representation as a data structure. We define $D^\varphi \triangleq \{\alpha \in D \mid |\alpha\rangle = |\varphi\rangle\}$, i.e., the set of all data structures in language D representing state $|\varphi\rangle$. In line with quantum information (see Sec. 5.2.1), we say that data structures α, β are *equivalent* if $|\alpha\rangle = \lambda\,|\beta\rangle$ for some $\lambda \in \mathbb{C}$.

Vector. A state vector α is a length-2^n array of complex entries $a_k \in \mathbb{C}^{2^n}$ satisfying $\sum_{k=1}^{2^n} |a_k|^2 = 1$, and interpretation $|\alpha\rangle = \sum_j \alpha_j\,|j\rangle$. Despite its size, the vector representation is used in many simulators [173]. We mainly include the vector representation to show that considering operation tractability alone can be deceiving.

Matrix Product States (MPS). An MPS M is a series of $2n$ matrices $A_k^x \in \mathbb{C}^{D_k \times D_{k-1}}$ where $x \in \{0,1\}$, $k \in [n]$ and D_k is the row dimension of the k-th matrix with $D_0 = D_n = 1$. The interpretation $|M\rangle$ is determined as $\langle \vec{y}|M\rangle = A_n^{x_n} \cdots A_2^{x_2} A_1^{x_1}$ for $\vec{y} \in \{0,1\}^n$. The size of M is the total number of matrix elements, i.e., $|M| = 2 \cdot \sum_{k=1}^{n} D_k \cdot D_{k-1}$. We will speak of $\max_{j \in [0\ldots n]} D_j$ as 'the' bond dimension.

Restricted Boltzmann Machine (RBM). An n-qubit RBM is a tuple $\mathcal{M} = (\vec{\alpha}, \vec{\beta}, W, m)$, where $\vec{\alpha} \in \mathbb{C}^n, \vec{\beta} \in \mathbb{C}^m$ for $m \in \mathbb{N}_+$ are *bias vectors* and $W \in \mathbb{C}^{n \times m}$ is a *weight matrix*. An RBM $\mathcal{M}$ represents the state $|\mathcal{M}\rangle$ as follows.

$$\langle \vec{x}|\mathcal{M}\rangle = e^{\vec{x}^T \cdot \vec{\alpha}} \cdot \prod_{j=1}^{m} (1 + e^{\beta_j + \vec{x}^T \cdot \vec{W}_j}) \tag{5.1}$$

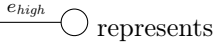
where $\vec{W}_j$ is the j-th column of W, β_j is the j-th entry of β and where we write $\vec{x}^T \cdot W_j$ to denote the inner product of the row vector $\vec{x}^T$ and the column vector $\vec{W}_j$ [80]. The size of $\mathcal{M}$ is $|\mathcal{M}| = n + m + n \cdot m$. We say this RBM has n *visible nodes* and m *hidden nodes*. A weight $W_{v,j}$ is an edge from the v-th visible node to the j-th hidden node. The j-th hidden node is said to contribute the multiplicative term $(1 + e^{\beta_j + \vec{x}^T \cdot \vec{W}_j})$.

Quantum Decision Diagrams (QDD). We define a Quantum Decision Diagram to represent a quantum state, based on the Valued Decision Diagram [113] as instantiated on a domain of binary variables (qu**bit**s) and a co-domain of complex values (amplitudes). A QDD α is a finite, rooted, directed acyclic graph (V, E), where each node v is labeled with an index $\mathsf{idx}(v) \in [n]$ and leaves have index 0. In addition, a 'root edge' e_R (without a source node) points to the root node. Each node has two outgoing edges, one labeled 0 (the low edge) and one labeled 1 (the high edge). In

Table 5.1: Various decision diagrams (DDs) treated by the literature. A DD represents a function $f\colon \{0,1\}^n \to R$ where the column Range specifies the set R. Here S is an arbitrary algebraic structure. The column *Merging strategy* lists the conditions under which two nodes v, w, representing subfunctions $f, g\colon \{0,1\}^k \to \mathbb{R}$ are merged. Here $z, a \in \mathbb{C}$ are complex constants, P_i are Pauli gates and $f + a$ denotes the function defined by $f(x) + a$ for all x. The DDs in **bold underlined text** are treated in depth in this thesis. See Chapter 7 for a definition of *variable tree*; see Gergov and Meinel [129] for a definition of FBDD type. (This table is identical to Table 2.2).

Architecture	**(Quantum) decision diagrams (and variants)**	Range	**Node merging strategy**
variable order	Decision Tree	(any)	(no merging)
	BDD, [67] ZDD, [229] TBDD, [328] CBDD, [68] KFBDD, [103] DSDBDD, [46, 264] CCDD, [195] Partitioned ROBDD, [237] Mod-2-OBDD, [130] ROBDD$[\wedge_i]_{\mathcal{C}}$, [194] CFLOBDD [293]	$\{0,1\}$	$f = g$
	MTBDD [87], **QuIDD** [331]	$\mathbb{C}$	$f = g$
	ADD [27]	S	$f = g$
	SLDD$_\times$ [114, 352], **QMDD** [227, 374], TDD, [163] WCFLOBDD [296]	$\mathbb{C}$	$f = z \cdot g$
	SLDD$_+$ [114], EVBDD [193]	$\mathbb{C}$	$f = g + a$
	AADD [281], FEVBDD [308]	$\mathbb{C}$	$f = z \cdot g + a$
	LIMDD [337]	$\mathbb{C}$	$f = zP_1 \otimes ... \otimes P_n \cdot g$
	DDMF	$U(2)$	$M = \mathbb{I} \otimes \cdots \otimes id \otimes U \cdot R$
variable tree	**SDD**, [96] ZSDD, [245] TSDD, [111] VS-SDD [235]	$\{0,1\}$	$f = g$
	PSDD [180]	$\mathbb{R}$	$f = g$
FBDD type	FBDD [129], Partitioned ROBDD [237]	$\{0,1\}$	$f = g$

addition, edge $e = vw$ pointing to a node w with index k has a label $label(e) \in \mathcal{E}_k$ for some edge label set $\mathcal{E}_k$; in this chapter, $\mathcal{E}_k$ is a group (for QMDD and LIMDD below with 0 added). Also, each leaf node v has a label $\mathsf{label}(v) \in \mathcal{L}$. The size of a QDD is $|\alpha| = |V| + |E| + \sum_{v \in V} |\mathsf{label}(v)| + \sum_{e \in E \cup \{e_R\}} |\mathsf{label}(e)|$. For simplicity, we require that no nodes are skipped, i.e. $\forall vw \in E\colon id[v] = id[w] + 1$.* The semantics are:

- A leaf node v represents the value $\mathsf{label}(v)$.
- A non-leaf node 

represents $|v\rangle = |0\rangle \otimes |e_{low}\rangle + |1\rangle \otimes |e_{high}\rangle$.
- An edge 

represents $|e\rangle = \mathsf{label}(e) \cdot |v\rangle$.

Consequently, any node v at *level* k, i.e., with $\mathsf{idx}(v) = k$, is k edges away from a leaf (along all paths) and therefore represents a k-qubit quantum state (or a complex vector).

In this chapter, we consider the following types of QDDs. We emphasize that an ADD can be seen as a special case of QMDD, which is a special case of LIMDD.

ADD: $\forall k\colon \mathcal{E}_k = \{1\}\,,\ \mathcal{L} = \mathbb{C}$ [27].

QMDD: $\forall k\colon \mathcal{E}_k = \mathbb{C},\ \mathcal{L} = \{1\}$ [352].

LIMDD: $\forall k\colon \mathcal{E}_k = \textsc{PauliLIM}_k \cup \{0\},\ \mathcal{L} = \{1\}$, where $\textsc{PauliLIM}_k \triangleq \{\lambda P \mid \lambda \in \mathbb{C} \setminus \{0\}, P \in \{\mathbb{I}, X, Y, Z\}^{\otimes k}\}$, i.e., the group generated by k-fold tensor products of the single-qubit Pauli matrices [337]:

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

Two isomorphic QDD nodes (Definition 5.2) v, w with $|w\rangle = \ell\,|v\rangle$ can be *merged* by removing w and rerouting all edges $uw \in E$ incoming to w to v, updating their edge labels accordingly (i.e., $\mathsf{label}(uv) := \mathsf{label}(uw) \cdot \ell$). Table 5.1 summarizes merging strategy for the above QDDs and related versions. If all isomorphic nodes are merged, we call a QDD *reduced*. We may assume a QDD is reduced (and even canonical), since it can be reduced in polynomial time and manipulation algorithms keep the QDD reduced using a MakeNode operation [67, 114, 227, 337].

*Asymptotic analysis is not affected by disallowing node skipping as it yields linear-size reductions at best [183].

Definition 5.2 (Isomorphic nodes)**.** QDD node v is isomorphic to node w, if there exists an edge label $\ell \in \mathcal{E}_{\mathsf{idx}(v)}, \ell \neq 0$ such that $\ell \cdot |v\rangle = |w\rangle$. Since $\mathcal{E}_{\mathsf{idx}(v)}$ is a group, $\ell \in \mathcal{E}_{\mathsf{idx}(v)}$ implies $\ell^{-1} \in \mathcal{E}_{\mathsf{idx}(v)}$ so $\ell^{-1}|w\rangle = |v\rangle$. Hence isomorphism is an equivalence relation.

It is well known that the variable order greatly influences QDD sizes [56, 58, 96]. Our results here assume any variable order: For instance, when we say that some structure is strictly more succinct than a QDD, it means that there is no variable order for which the QDD is more succinct than the other structure (for representing a certain worst-case state).

5.3 Succinctness of quantum representations

In this section, we compare the succinctness of the data structures introduced in Section 5.2. A language L_1 is as succinct as L_2 if L_2 uses at least as much space to represent any quantum state as L_1, up to polynomial factors. Thus, a more succinct diagram is more expressive when we constrain ourselves to polynomial (or asymptotically less) space. We define a language L_1 to be *at least as succinct* as L_2 (written $L_1 \preceq_s L_2$), if there exists a polynomial p such that for all $\beta \in L_2$, there exists $\alpha \in L_1$ such that $|\alpha\rangle = |\beta\rangle$ and $|\alpha| \leq p(|\beta|)$. We say that L_1 is more succinct ($L_1 \prec_s L_2$) if

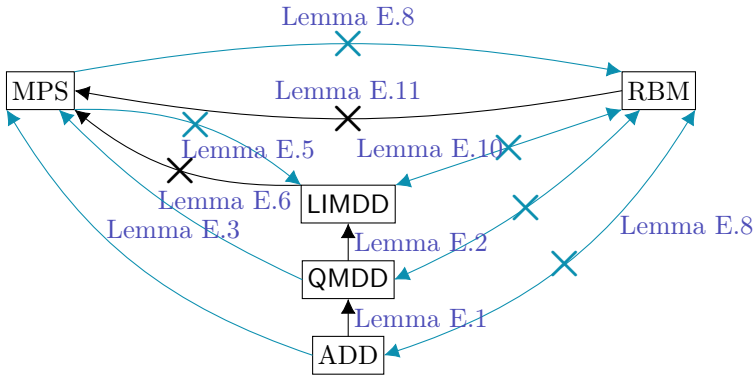


Figure 5.2: Succinctness relations between various classical data structures for representing quantum states. Solid arrows $A \to B$ denote $B \prec_s A$, i.e., B is strictly more succinct than A. Crossed arrows $A \not\to B$ denote a separation $B \not\preceq_s A$; a bidirectional crossed arrow implies incomparability. Blue arrows indicate novel relations that we identified.

L_1 is at least as succinct as L_2 but not vice versa.

The results of this section are summarized by Theorem 5.1 (Figure 5.2) and proved in App. E. We now highlight our novel results. First, we show that MPS $\prec_s$ QMDD, by (i) describing a simple and efficient way to write a QMDD as an MPS, and (ii) finding a family of states, which we call $|\text{Sum}\rangle$, which have small MPS but which require exponentially large QMDD. It follows that MPS $\prec_s$ ADD. Second, we strengthen (ii) to show that the same state also requires exponentially large LIMDDs, thus we establish LIMDD $\not\preceq_s$ MPS. The reverse here also holds because MPS cannot efficiently represent certain stabilizer states, while LIMDD is small for all stabilizer states, as shown in [337]. Lastly, we show that RBMs can efficiently represent $|\text{Sum}\rangle$. Since it is well-known that RBM explodes for parity functions, which are small for QDD [214], we establish incomparability of RBM with all three QDDs.

Theorem 5.1. The succinctness results in Figure 5.2 hold.

5.4 Tractability of quantum operations

In this section, we investigate for each data structure (DS) the tractability of the main relevant queries and manipulation operations for the different applications discussed Section 5.1.

By a *manipulation operation*, we mean a map $D^c \to D$ and by a *query operation* a map $D^c \to \mathbb{C}$, where D is a class of data structures and $c \in \mathbb{N}_{\geq 1}$ the number of operands. We say that a class of data structures D *supports* a (query or manipulation) operation $OP(D)$, if there exists an algorithm implementing OP whose runtime is polynomial in the size of the operands, i.e., $|\varphi_1| + ... + |\varphi_c|$.

The operations whose tractability we investigate are:

- **Sample**: Given a DS representing $|\varphi\rangle$, sample the measurement outcomes $\vec{x} \in \{0,1\}^n$ from measuring all qubits of $|\varphi\rangle$ in the computational basis.
- **Measure**: Given a DS representing $|\varphi\rangle$ and $\vec{x} \in \{0,1\}^n$, compute the probability of obtaining $\vec{x}$ when measuring $|\varphi\rangle$.
- **Gates** (**Hadamard**, Pauli **X,Y,Z**, Controlled-Z **CZ**, **Swap**, **T**): Given a DS representing $|\varphi\rangle$ and a one- or two-qubit gate U, construct a DS representing $U|\varphi\rangle$. This gate set is universal [63] and is used by many algorithms [329].

Table 5.2: Tractability of queries and manipulations on the data structures analyzed in this chapter (single application of the operation). A ✓ means the data structure supports the operation in polytime, a ✓' means supported in randomized polytime, and ✖ means the data structure does not support the operation in polytime. A ○ means the operation is not supported in polytime unless $P = NP$. ? means unknown. The table only considers deterministic algorithms (for some ? a probabilistic algorithm exists, e.g., for **InnerProd** on RBM). Novel results are blue and underlined.

		Queries					Manipulation operations						
		Sample	**Measure**	**Equal**	**InnerProd**	**Fidelity**	**Addition**	**Hadamard**	**X,Y,Z**	**CZ**	**Swap**	**Local**	**T-gate**
Vector		✓'	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ADD	Section F.1	✓'	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
QMDD	Section F.2	✓'	✓	✓	✓	✓	✖	✖	✓	✓	✖	✖	✓
LIMDD	Section F.3	✓'	✓	✓	○	○	✖	✖	✓	✓	✖	✖	✓
MPS	Section F.4	✓'	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RBM	Section F.5	✓'	?	?	○	○	?	?	✓	✓	✓	?	✓

- **Local** (Local gates) as general case of Gates: Given a DS representing $|\varphi\rangle$, a constant $k \in \mathbb{N}_{\geq 1}$ and a k-local gate U, construct a DS representing the state $U \cdot |\varphi\rangle$.
- **Addition**: Given DSs for states $|\varphi\rangle, |\psi\rangle$, construct a DS representing the state $|\varphi\rangle + |\psi\rangle$.
- **InnerProd** (inner product) and **Fidelity**: Given DSs for states $|\psi\rangle, |\psi\rangle$, compute $\langle\varphi|\psi\rangle$ and $|\langle\varphi|\psi\rangle|^2$.
- **Equal** (Equality): Decide whether the states represented by two given data structures are equivalent.

We are motivated to study these operations by their applicability in the three application domains from Section 5.1. First, classical simulation of quantum circuits includes **Gates**, as well as **Measure** (strong simulation) and **Sample** (weak simulation). Although **Addition** is not, technically speaking, a quantum operation, we include it because addition of quantum states can happen due to quantum operations. For example, if we apply first a Hadamard gate, and then a measurement, to the state $|0\rangle|\varphi\rangle + |1\rangle|\psi\rangle$, we may obtain the state $|0\rangle \otimes (|\varphi\rangle + |\psi\rangle)$. For a data structure, this resulting state is at least as difficult to represent as the sum $|\varphi\rangle + |\psi\rangle$. Therefore, it is instructive to check which data structures support **Addition**: it allows us to explain

why certain gates are not tractable for a data structure (such as the swap and general local gates). In particular, all decision diagram implementations support an explicit addition subroutine [86, 227]. In addition, **InnerProd** and **Fidelity** (and **Local**) are required in variational methods and quantum circuit verification involves **Eq**.

For the operations listed above and the languages from Section 5.2, we present an overview of existing and novel tractability results in Theorem 5.2 (Table 5.2), with proofs for all entries in App. F. The novel results in this work are the hardness results (denoted ∘) for **InnerProd** and **Fidelity** on LIMDD and RBM (Theorem 5.3 and for **InnerProd** we can reduce from **Fidelity**) and unsupported manipulations (denoted ✖) on QMDD and LIMDD. More generally, our proof shows that computing **Fidelity** is hard for any data structure which succinctly represents both all graph states and the Dicke state, such as RBM and LIMDD. A fidelity algorithm for QMDD was mentioned by [71], but, to the best of our knowledge, had not previously been described or analyzed.

Theorem 5.2. The tractability results in Table 5.2 hold.

Theorem 5.3. Assuming the exponential time hypothesis, the fidelity of two states represented as LIMDDs or RBMs cannot be computed in polynomial time. The proof uses a reduction from the #EVEN SUBGRAPHS problem [169].

5.5 Rapidity of data structures

The tractability criterion, studied in Section 5.4, sometimes gives a skewed picture of efficiency. For example, looking naively at Table 5.2, it seems that ADD is faster than QMDD when applying a Hadamard gate. Yet there is no state for which applying the Hadamard gate on its QMDD representation is slower than on its ADD representation. So succinctness actually consistently mitigates the worst-case runtime behavior. To remedy this shortcoming, [194] introduced the notion of *rapidity* for canonical data structures.

In Definition 5.3, we generalize rapidity to support non-canonical data structures, such as MPS, RBM, d-DNNF [95] and CCDD [195]. To achieve this, Definition 5.3 requires that for a fixed input φ to ALG_1, among all equivalent inputs to ALG_2, there is one on which ALG_2 is at least as fast as ALG_1. It may seem reasonable to require, instead, that ALG_2 is at least as fast as ALG_1 on *all* equivalent inputs; however, in general,

there may be no upper bound on the size of the (infinitely many) such inputs; thus, such a requirement would always be vacuously false.

In the following, we write $time(A, x)$ for the runtime of algorithm A on input x.

Definition 5.3 (Rapidity for non-canonical data structures)**.** Let D_1, D_2 be two data structures and consider some c-ary operation OP on these data structures. In the below, ALG_1 (ALG_2) is an algorithm implementing OP for D_1 (D_2).

(a) We say that ALG_1 is *at most as rapid as* ALG_2 iff there exists a polynomial p such that for each input $\varphi = (\varphi_1, \ldots, \varphi_c)$ there exists an equivalent input $\psi = (\psi_1, \ldots, \psi_c)$, i.e., with $|\varphi_j\rangle = |\psi_j\rangle$ for $j = 1 \ldots c$, for which $time(ALG_2, \psi) \leq p\,(time(ALG_1, \varphi))$. We say that ALG_2 is *at least as rapid as* ALG_1.

(b) We say that $OP(D_1)$ is *at most as rapid as* $OP(D_2)$ if for each algorithm ALG_1 performing $OP(D_1)$, there is an algorithm ALG_2 performing $OP(D_2)$ such that ALG_1 is at most as rapid as ALG_2.

We remark that, when applied to canonical data structures, Definition 5.3 reduces to the definition by Lai et al. (Lemma G.1), except that Lai et al. allow the input to be fully read by the algorithm: $time(ALG_1, x_1) \leq \mathsf{poly}(time(ALG_2, x_2)\underline{+|x_2|})$ (difference underlined). We omit this to achieve transitivity. Rapidity indeed has the desirable property that it is a preorder, i.e., it is reflexive and transitive, as Theorem 5.4 shows. App. G provides a proof.

Theorem 5.4. Rapidity is a preorder over data structures.

5.5.1 A sufficient condition for Rapidity

We now introduce a simple sufficient condition for rapidity in Theorem 5.5, allowing researchers to easily establish that one data structure is more rapid than another *for many relevant operations simultaneously.* Previously, such proofs were done for each operation individually [194]. We use this sufficient condition to establish rapidity relations between many of the data structures studied in this work. By a *transformation* f from data structure D_1 to D_2 we mean a map such that $|f(x_1)\rangle = |x_1\rangle$ for all $x_1 \in D_1$. We also need the notions of a weakly minimizing transformation (Definition 5.4) and a runtime monotonic algorithm (Definition 5.5).

Definition 5.4 (Weakly minimizing transformation)**.** Let D_1, D_2 be data structures. A transformation $f : D_1 \to D_2$ is *weakly minimizing* if f always outputs an instance which is polynomially close to minimum-size, i.e., there exists a polynomial p such that for all $x_1 \in D_1, x_2 \in D_2$ with $|x_1\rangle = |x_2\rangle$, we have $|f(x_1)| \leq p(|x_2|)$.

Definition 5.5 (Runtime monotonic algorithm)**.** An algortihm ALG implementing some operation on data structure D is *runtime monotonic* if for each polynomial s there is a polynomial t such that for each state $|\varphi\rangle$ and each $x, y \in D^\varphi$, if $|x| \leq s(|y|)$, then $time(ALG, x) \leq t(time(ALG, y))$.

Theorem 5.5 (A sufficient condition for rapidity)**.** Let D_1, D_2 be data structures with $D_1 \preceq_s D_2$ and OP a c-ary operation. Suppose that,

A1 $OP(D_2)$ requires time $\Omega(m)$ where m is the sum of the sizes of the operands; and

A2 for each algorithm ALG implementing $OP(D_2)$, there is a runtime monotonic algorithm ALG^{rm}, implementing the same operation $OP(D_2)$, which is at least as rapid as ALG; and

A3 there exists a transformation from D_1 to D_2 which is (i) weakly minimizing and (ii) runs in time polynomial in the output size (i.e, in time $\mathsf{poly}(|\psi|)$ for transformation output $\psi \in D_2$); and

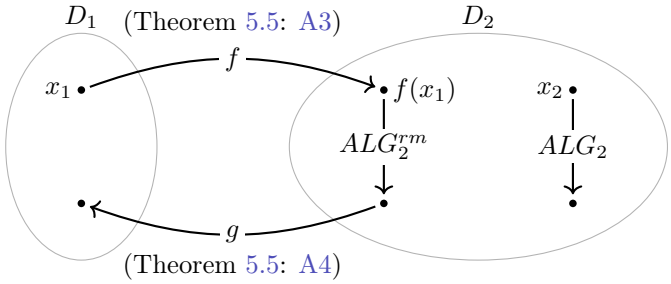


Figure 5.3: Visualization of the proof of Theorem 5.5 in case OP is a transformation operation: Given runtime monotonic algorithm ALG_2^{rm} implementing OP on language D_2, the composed algorithm $ALG_1 \triangleq g \circ ALG_2^{rm} \circ f$ for $OP(D_1)$ is at least as rapid as ALG_2. To prove this, we consider $x_2 \in D_2$ and an equivalent and at most only polynomially larger than $x_1 \in D_1$ and show that ALG_1 takes at most polynomially more time on x_1 than ALG_2 on x_2. ALG_1 is also runtime monotonic. Horizontally-aligned instances of data structures are equivalent, i.e. represent the same quantum state. App. G provides a proof.

A4 if OP is a manipulation operation (as opposed to a query), then there also exists a polynomial time transformation from D_2 to D_1 (polynomial time in the input size, i.e, in $|\rho|$ for transformation input $\rho \in D_2$).

Then D_1 is at least as rapid as D_2 for operation OP.

Figure 5.3 provides a proof outline that illustrates the need for (polynomial-time) transformations in order to execute operations on states represented in data structure D_1 on their D_2 counterpart. The operation on $f(x_1)$ is at most polynomially slower than on its counterpart x_2 because f produces a small instance (because f is weakly minimizing), and because ALG_2^{rm} is not much slower on such instances (because it is runtime monotonic). We opted for weakly minimizing transformations (rather than *strictly* minimizing transformations), because a minimum structure might be hard to compute and is not needed in the proof. Runtime monotonic algorithms are ubiquitous, for instance, most operations on MPS scale polynomially in the bond dimension and number of qubits [335]. Finally, we emphasize that for canonical data structures D, each algorithm is runtime monotonic and any transformation $D_1 \to D$ is weakly minimizing.

5.5.2 Rapidity between quantum representations

We now capitalize on the sufficient condition of Theorem 5.5 by revealing the rapidity relations between data structures for all operations satisfying A1 and A2. Theorem 5.6 shows our findings. We highlight the result that MPS is at least as rapid as QMDD in Theorem 5.7. Its proof provides the required transformations from MPS to QMDD and back.

Theorem 5.6. The rapidity relations in Figure 5.4 hold.

Theorem 5.7. MPS is at least as rapid as QMDD for all operations satisfying A1 and A2.

Proof sketch. Since QMDD is canonical, the runtime monotonicty (A2 of Theorem 5.5, see Definition 5.5) and weakly-minimizing (A3) requirements are satisfied automatically. Hence we only need to provide efficient transformations in both directions (A3-A4): We transform QMDD to MPS by choosing its matrices A_k^x to be the weighted

bipartite adjacency matrices of the x-edges (low / high edges) between level k and $k-1$ nodes of the QMDD. In the other direction, an MPS can be efficiently transformed into an QMDD by contracting the first open index with $|0\rangle$ and then $|1\rangle$ to find the coefficients in the $|0\rangle$ and $|1\rangle$ parts of the state's Shannon decomposition; and repeating this recursively for all possible partial assignments. Dynamic programming using the fidelity operation (efficient for MPS) ensures that only a polynomial number of recursive calls are made. □

The relations not involving MPS involve QDDs. QDDs are canonical data structures as explained in Section 5.2, so that runtime monotonicity and weak minimization of transformations are automatically ensured. In Section G.5, we give detailed transformations between QDDs. Transformations between different QDDs can be realized using the well-known MakeNode procedure (see Section 5.2 and Section 2.3), in linear time in the resulting QDD size (using dynamic programming).

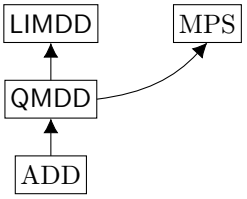


Figure 5.4: Rapidity relations between data structures considered here. A solid arrow $D_1 \to D_2$ means D_2 is at least as rapid as D_1 for all operations satisfying A1 and A2 of Theorem 5.5.

5.6 Related work

Darwiche and Marquis [97] pioneered the knowledge compilation map approach followed here. Fargier et al. [113] employ the same method to compare decision diagrams for real-valued functions. In order to mend a deficiency in the notion of tractability, Lai et al. [194] contributed the notion of *rapidity*, which we extend in Section 5.5

Alternative ways to deal with non-canonical data structures includes finding a canonical variant, as has been done for MPS [262], although it is not canonical for all states. For other structures, such as RBM, d-DNNF and CCDD (mentioned before) such canonical versions do not exist as far as we know, and might not be tractable.

Often the differences between DD variants in Table 2.2 are subtle differences in the constraints used for obtaining canonicity. While perhaps irrelevant for the (asymptotic) analysis, we emphasize that these definitions have great practical relevance. For instance, canonicity makes dynamic programming efficient, which enables fast implementations of QDD operations, while it also reduces the diagram size. Moreover, in practice the behavior of floating point calculations interacts with the canonicity constraints [369], significantly affecting performance. Like Fargier et al. [113], we do not analyze numerical stability and focus on asymptotic behavior.

The affine algebraic decision diagram (AADD), introduced by Sanner and McAllister [281], augments the QMDD as shown in Table 2.2. Their work proves that the worst-case time and space performance of AADD is within a multiplicative factor of that of ADD. The concept of rapidity makes explicit that this is the case only when equivalent inputs are considered for both structures. We note that AADD would be able to represent $|\mathrm{Sum}\rangle$ (a hard case for all QDDs studied here), so its succinctness relation with respect to RBM is still open. We omit Affine ADD (AADD) [281], since to the best of our knowledge these have not been applied to quantum computing yet. The LIMDD data structure is implemented and compared against QMDD in Chapter 4 (which is based on [338]).

Burgholzer et al. [72] compare tensor networks, including MPS, to decision diagrams, on slightly different criteria, such as abstraction level and ease of distributing the computational workload on a supercomputer. Hong et al. introduce the Tensor Decision Diagram, which extends the QMDD so that it is able to represent tensors [163] and their contraction. Context-Free-Language Ordered Binary Decision Diagrams (CFLOBDDs) [293, 295, 296] achieve a similar goal by extending BDDs with insights from visibly pushdown automata [10].

The stabilizer formalism is a tractable but non-universal method to represent and manipulate quantum states [3]. Its universal counterpart is the stabilizer decomposition-based method, introduced by Bravyi et al. [64], which relies on a linear combination of representations in the stabilizer formalism that is exponential only in the number of T gates in the circuit. Although this is a highly versatile technique [62], no super-polynomial lower bounds on their size are known at the moment, making it less useful to include them in a knowledge compilation map.

Similarly, we expect the relation between tree tensor networks [248] and SDDs [96] to be similar the relation between QMDD and MPS, since SDD also extend the linear

QDD variable order with a 'variable tree.'

Ablayev et al. [5] introduce the Quantum Branching Program, a branching program whose state is a superposition of the nodes on a given level, some of which are labeled *accepting*. An input string $x \in \{0,1\}^n$ determines how the superposition evolves. A string $x \in \{0,1\}^n$ is said to be accepted by the automaton if, after evolving according to x, a measurement causes the superposition to collapse to an accepting state with probability greater than $\frac{1}{2}$. Thus, the automaton accepts a finite language $L \subseteq \{0,1\}^n$. Since this diagram accepts a language, rather than represents and manipulates a quantum state, we cannot properly fit it into the present knowledge compilation map.

5.7 Discussion

We have compared several classical data structures for representing quantum information on their succinctness, tractability of relevant operations and rapidity. We catalogued all relevant operations required to implement simulation of quantum circuits, variational quantum algorithms (e.g., quantum machine learning) and quantum circuit verification. We have followed the approach of Darwiche and Marquis [97] and Fargier et al. [113] to map the succinctness and tractability of the data structures. In order to mend a deficiency in the notion of tractability that was noticed by several researchers, we additionally adopt and develop the framework of *rapidity*. We contribute a general-purpose method by which data structures, whether canonical or not, may be analytically compared on their rapidity.

Common knowledge says that there is a trade-off between the succinctness, and the speed of a data structure. In contrast, we find that, the more succinct data structures are often also more rapid. For example, we find that algebraic decision diagrams, QMDD, and matrix product states are each successively more succinct and more rapid. However, in practice, the QMDD has achieved striking performance on realistic benchmarks [374] due to a successful sustained effort to optimize the software implementation. But we emphasize that e.g. MPS was not developed with the intention for circuit simulation but for quantum simulation, and QMDD is vice versa. Therefore, more research is needed to compare the relative performance of these data structures on the various application domains in practice.

In future work, we could also consider unbounded operations (multiple applications of

the same gate) and other data structures, e.g., tree tensor networks [248] and affine algebraic decision diagrams [281].