Data structures for quantum circuit verification and how to compare them

Vinkhuijzen, L.T.

Chapter 8

Conclusions and outlook

In this dissertation, we set out to develop methods for analysis and particularly the verification of quantum algorithms. We have seen that verification can be done in many ways, and have contributed to the approach which attacks this problem using decision diagrams. We now reflect on our findings by revisiting the research questions posed in Chapter 1. Finally, we conclude by posing questions for future research.

8.1 Research questions, revisited

Quantum algorithms can be verified automatically using several methods, which we briefly surveyed in Section 2.4, and using a variety of data structures, which we examined in Chapter 5. The principal bottleneck in these methods is the large amount of computer memory required to store the state vectors of the quantum states that are encountered during simulation (or to store the unitary matrix of a quantum circuit). To this end, these data structures are used to losslessly compress both these states vectors and the unitary matrices of quantum circuits. We have argued that, therefore, powerful data structures are the key to effective methods for verification and indeed for quantum algorithm analysis more broadly. Concretely, a bottleneck for the approaches we studied was that decision diagrams cannot efficiently analyze stabilizer circuits, an important class of quantum circuits. Specifically, existing decision diagrams require exponential amounts of space and time for this use case (Theorem 3.2). In light of this significant limitation, we asked:

> **Research question 1.** *Can we unite the strengths of decision diagrams and the stabilizer formalism?*

To combat this memory bottleneck, in Chapter 3 we introduced a new data structure for the analysis and verification of quantum circuits: the Local Invertible Map Decision Diagram (LIMDD), which unites the strengths of existing decision diagram-based approaches and the stabilizer formalism. We compared the LIMDD to several existing data structures in Chapter 3, 4 and 5 and found that LIMDDs are exponentially faster (in a qualitative sense: they are more *rapid*) and more succinct than QMDDs, more expressive than the stabilizer formalism; and incomparable to matrix product states, restricted Boltzmann Machines and the extended stabilizer formalism.* These results were obtained in a mathematically rigorous way: we provided examples of quantum states which LIMDDs can represent efficiently but which require exponentially scaling resources when using the data structures above and vice versa. The Venn diagram in Figure 8.1 summarizes these results.

We showed empirically that LIMDDs can analyze an important quantum subroutine called the Quantum Fourier Transform, in Chapter 4. To this end, we simulated the circuits using random stabilizer states as the inputs. Due to their better asymptotic scaling, LIMDDs outperform QMDDs on this task: they are faster on circuits that contain 19 qubits or more, and are about 5 times faster on the largest circuit that was tested.

In the introduction, we sketched several shortcomings of the literature on knowledge compilation with regards to data structures that are used for quantum algorithm analysis. Notably, existing data structures in widespread use in this field had not been compared on their succinctness and could not be compared on their rapidity; and the tradeoffs between the tractability of key operations had not been systematically compared across existing data structures. Therefore, we asked:

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

We answered Research question 2 by giving a quantum knowledge compilation map

*Specifically, we compared LIMDDs to a specific instantiation of the extended stabilizer formalism which captures the way this formalism is often used in practice; for details see Sec. 3.3.4.

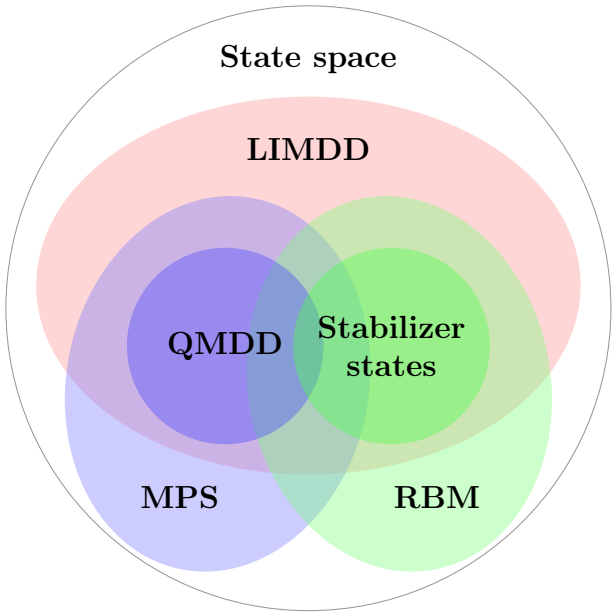

Figure 8.1: A Venn diagram showing the expressive power of almost all data structures studied in this thesis. The circle of a data structure (e.g., the pink circle labeled LIMDD) can be interpreted as either (i) the set of quantum states that can be represented in polynomial space using that data structure, thus depicting the *succinctness relations*; or (ii) the set of tasks that can be solved in polynomial time using this data structure, thus depicting the *rapidity relations*.

in the style of Darwiche and Marquis [97] and Fargier et al. [113] in Chapter 5 in which we mapped the succinctness, tractability and rapidity of many data structures used in quantum algorithm analysis: ADDs, QMDDs, LIMDDs, matrix product states, restricted Boltzmann machines and the explicit state vector representation. The Venn diagram in Figure 8.1 summarizes these results. Notably, we found that LIMDDs and matrix product states are more rapid than QMDDs.

To this end, we extended the definition of rapidity to also cover the case of non-canonical data structures. We then gave a simple sufficient condition for when one (non-canonical) data structure is more rapid than another. This sufficient condition allowed us to establish almost all rapidity relations between the data structures we studied. We argued that rapidity is an especially useful metric when choosing which data structure to use for a given task because, contrary to the tractability criterion, it directly compares the absolute amounts of time two data structures require for a

given operation, correctly taking into account the fact that they perform the operation on instances of different sizes due to their different succinctness. Lai et al. [194] formulate this same idea: "[rapidity] reflects the idea that exponential operations may be preferable if the language can be exponentially more succinct."

By contrast, the criterion of tractability was found to be less informative because it appears to "punish" data structures for adding functionality which increases their succinctness, even when canonicity is preserved and this additional functionality does not increase the absolute time required for a given operation. Let us briefly give two examples of this phenomenon to illustrate this subtle point. First, applying a Hadamard gate to a quantum state is tractable in the state vector representation but is intractable for QMDDs and LIMDDs. Second, computing the fidelity of two quantum states is tractable for QMDDs but intractable for LIMDDs. Both of these results are counterintuitive, since both the step from the state vector representation to QMDDs, and from QMDDs to LIMDDs, are steps which add functionality to a data structure, which improve their succinctness and which do not increase the runtimes of any operations (up to polynomial factors). The criterion of rapidity, on the other hand, yields the expected result that LIMDDs are more rapid than QMDDs, which in turn are more rapid than the state vector representation, for all these operations.

We further answered Research question 2 in Chapter 3 by analytically comparing LIMDDs to matrix product states and the extended stabilizer formalism (ESF). To this end, we gave an example of a quantum circuit which LIMDDs can analyze efficiently, whereas we describe a specific instantiation of the ESF which requires exponential time for this same task, conditional on the exponential time hypothesis.

In the future, we hope that researchers will investigate the rapidity of the data structures they analyze, both when introducing new data structures and when making knowledge compilation maps. More broadly, we hope that rapidity, rather than succinctness or tractability, will become the primary design objective when designing new data structures.

We remarked that very few decision diagrams have been adapted to the quantum setting and asked in which cases it might be lucrative to do so:

> **Research question 3.** *Which classical decision diagrams might be effective for the analysis of quantum algorithms, if they were suitably adapted?*

To answer this research question, in Chapter 6 and 7, we investigated two DDs which we find promising candidates for future development in quantum settings: the DSDBDD and the SDD, respectively. We now treat these two DDs in turn.

We found that the DSDBDD is exponentially more succinct than the BDD due to its superior ability to recognize structure in a Boolean function. We comment more on how to extend this data structure to the quantum setting in Sec. 8.3.2.

We are the first to apply SDDs to model checking. In Chapter 7, we show the empirical results on a large standard model checking benchmark set. Recall that an SDD extends BDD by using a variable tree, which generalizes the BDD's notion of a variable order. A vtree consists of a full binary tree in addition to a variable order. Both these components determine the size of the SDD and the speed with which manipulation operations can be carried out, so a tool which uses an SDD needs to make good choices for both components in order to obtain good performance. To choose the vtree, we proposed the first heuristics for this purpose and found that in general it helps to group related variables together. We found that SDDs using these heuristics often outperform BDDs on memory consumption, and sometimes also on time, if its vtree is chosen well. For example: noting that an integer program variable consists of multiple bits, for every such integer variable, the vtree should contain a subtree which contains only the bits of this integer variable. The resulting software package integrates SDDs into the publicly available, open source model checker LTSmin [210]. To extend the SDD to the quantum domain, a promising start may be to draw inspiration from the Probabilistic SDD [180] and Tree Tensor Networks [236, 290].

8.2 Discussion

Unless P = BQP,[†] any classical simulation of quantum systems will have to make some tradeoff between speed, memory consumption, and accuracy, or has to specialize in a restricted set of circuits. The LIMDDs presented in this work represent a novel point in this optimization space, but they necessarily make some tradeoff compared to other methods. Therefore, we briefly but critically reflect on the limitations of LIMDDs as a data structure, and on the work in this thesis more broadly.

Compared to QMDDs, LIMDDs have an $\mathcal{O}(n^3)$ computational overhead when perform-

[†]It is widely believed that P $\neq$ BQP. See [20] for definitions of these complexity classes.

ing almost any manipulation operation. The purpose of the subroutines that cause this overhead is to allow for exponentially greater compression, i.e., to allow for less memory consumption; indeed, this compression allows LIMDDs to be faster in those cases where the workload saved due to compression outweighs the overhead which facilitated this compression. However, this overhead may become a liability in cases where LIMDDs obtain no significant compaction. Fortunately, in Chapter 4 we have seen that this worst case scenario does not necessarily materialize, as LIMDDs outperform QMDDs when analyzing the quantum Fourier transform even though their size is not much smaller. This can be explained by the fact that most of the overhead is due to Gaussian elimination of the stabilizer group stored with each node, but in this case these groups are almost all empty, so there is almost no work to be done.

Numerical accuracy and numerical stability are problems for any real-valued decision diagram [156, 157] and LIMDDs are no exception. We have not investigated to what extent these problems affect LIMDDs and leave this important question to future work. An optimistic hypothesis is that one should expect to encounter these problems to a lesser extent for LIMDDs than for QMDDs because, all other things equal, LIMDDs contain fewer floating point numbers than QMDDs do, simply because they are smaller.

We concluded Chapter 4 by reflecting on the methodology of circuit equivalence testing via simulation of random states. We remarked that the theoretical guarantees given by stabilizer states are good, but not perfect (specifically, the expected fidelity is optimal in the black-box setting, but the probability that a stabilizer state is a counterexample is less than that of a Haar-random state). Therefore, if we wish to obtain better guarantees, we will need to draw the random inputs from a different set of quantum states. In choosing this set, there is a tension between a set which is *random enough* to yield good theoretical guarantees, yet *structured enough* that computational methods are effective. Concretely, using a random stabilizer state to analyze a circuit U is equivalent to effecting a change of basis and analyzing instead the matrix $C^\dagger UC$, where C is a Clifford circuit that produces the random stabilizer state. Since C is by design a random unitary, one may therefore expect that, all other things equal, the matrix $C^\dagger UC$ will possess less structure than U, and that computational methods will therefore have a harder time analyzing it.

In Chapter 5, we mapped the tractability of several elementary operations for many data structures. However, the tractability of elementary operations is not always the most informative data point about a given data structure. Instead, it is often

more useful to know whether a data structure can efficiently analyze a given family of quantum circuits – quantum circuits, of course, being successive applications of elementary operations (i.e., gates). Making a knowledge compilation map always involves making such a sacrifice: on one hand, we abstract away from concrete sets of use cases (such as specific families of quantum circuits, in our case), which diminishes the immediate applicability of the results; on the other hand, by examining which elementary operations are efficiently supported by a data structure, one ostensibly gains insights which are more likely to be general enough to carry over from one use case to another. We say more about future work which might diminish the chasm between these opposing goals in Research challenge 11.

8.3 Future work

The work in this thesis does not close the book on quantum algorithm analysis or decision diagram research – quite the contrary: we have raised more questions than we have answered. We list here several open problems which we find promising avenues for future research. We first lay out future work for specific decision diagrams and then ask several more general questions.

8.3.1 Future work on LIMDDs.

There are several avenues for further development of LIMDDs. Broadly speaking, we see three categories: (i) LIMDDs should be deployed in practical and easy-to-use software applications for quantum simulation and verification; (ii) the existing LIMDD implementations can be improved and extended; and (iii) there are several analytical questions about the expressive power of LIMDDs, which can shed light on which LIMDD extensions are most promising.

Research challenge 1. *Facilitate the deployment of LIMDDs.*

Although we have implemented LIMDDs and although the software package is versatile and easy to use, the package is currently more or less a stand-alone application. However, a software package will only start to deliver value when it is integrated into a toolchain or an IDE in a way that makes it accessible and easy to use for the end

user – in this case, programmers of quantum computers (as noted also by, e.g., Raed el Aoun et al. [269]).

Research challenge 2. *Develop LIMDDs for classical applications.*

We are enthousiastic about the potential of LIMDDs in the classical domain. To this end, we must redefine what it means for two Boolean functions to be isomorphic. For example, we can say that two Boolean functions $f, g\colon \{0,1\}^n \to \{0,1\}$ are Pauli-isomorphic if there exist Boolean values $z_1, \ldots, z_n, a_1, \ldots, a_n, b \in \{0,1\}$ such that

$$f(x_1, \ldots, x_n) = b \oplus x_1 z_1 \oplus \cdots \oplus x_n z_n \oplus g(x_1 \oplus a_1, \ldots, x_n \oplus a_n) \quad \text{for all } x \in \{0,1\}^n \tag{8.1}$$

Thus, two functions are classically Pauli-isomorphic if they are equal modulo (i) complementation of the variables combined with (ii) an affine linear function. An encouraging first result is that we can immediately obtain an exponential separation between classical Pauli-LIMDD versus OBDD because the separation between $\langle X \rangle$-LIMDD and QMDD for coset states (Corollary A.1) carries over to this context. Namely, in the classical domain, the equivalent of a "coset state" is a set of solutions to a system of linear equations over $\mathbb{F}_2$.

Let us name two appealing properties of exploring the classical domain. First, many DDs seem to falter at "two-dimensional" problems (e.g., [318]), i.e., problems in which the variables are arranged in a grid, whereas LIMDDs have been shown to solve at least one such problem, namely it can efficiently represent and manipulate graph states. Second, numerical instability (resulting from inaccurate floating-point arithmetic) will likely not be an issue. Therefore, we can efficiently and reliably apply dynamic variable reordering to these diagrams, as opposed to the quantum case [156].

Research challenge 3. *Can DDs with other architectures than a variable order benefit from LIMs?*

Although the landscape of decision diagrams is rich and diverse, the underlying architecture of every proposed decision diagram can (to the best of the author's knowledge) be classified as one of the following three (see also Table 2.2, which lists examples decision diagrams using the various architectures):

1. architectures based on a **variable order** (these constitute the vast majority of implementations, including BDD, [67] ZDD, [229] DSDBDD, [46, 264] TBDD, [328] KFBDD, [103] LIMDD, [337] QMDD, [227, 374] AADD, [281]...);
2. architectures based on a **variable tree** (these include the SDD [96] and its extensions: the Tagged SDD, [111] ZSDD, [245] VS-SDD [235] and probabilistic SDD [180]);
3. architectures based on a **variable decision diagram** (these include the graph-based BDD [129] and Partitioned ROBDD [237]).

In our work, we have explored only architectures based on a variable order. We are excited to note that, by following the simple guiding principle that isomorphic nodes should be merged, a LIMDD-like structure can be implemented on any of these architectures. More generally, we can take any one of the decision diagrams named above as a starting point and augment it with LIMs to obtain a new, canonical diagram. This way we may, in principle, obtain, e.g., (i) a combination of CCDD [195] with classical LIMDD; or (ii) a zero-suppressed LIMDD (by skipping nodes of the form $|\varphi\rangle = |0\rangle\,|\varphi_0\rangle$); or any other combination. Of course, this is easier said than done: we note that it is easy to *define* a decision diagram, and much harder to *realize* it in software, especially if its performance needs to compete with the state-of-the-art.

> **Research challenge 4.** *Analyze and implement G-LIMDDs for various choices of G.*

In our work, we have worked primarily with $G = \textsc{Pauli}$, but there are many interesting choices that lead to more succinct and more rapid G-LIMDDs . For example, the choice $G = \langle X, T\rangle$ allows us to apply T gates in constant time and controlled-T gates in polynomial time. Alternatively, by taking the cyclic group $G = \langle R(n)\rangle$ generated by the matrix $R(n) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi 2^{-n}} \end{bmatrix}$ we can represent the Quantum Fourier Transform (QFT) in only $\mathcal{O}(n^4)$ space. This is a big improvement compared to QMDD and Pauli-LIMDD, for which the QFT is a worse-case situation (namely, these diagrams require size $\Omega(2^n)$). Giving LIMDDs slightly more power still, the group $G = \langle X, R(n)\rangle$ allows us to efficiently represent the QFT, all Clifford circuits, all stabilizer states and the so-called XS-stabilizer states studied by Ni et al. [239]. More ambitiously, we can consider multi-qubit groups, e.g., the group $G = CZ$ generated by all controlled-Z gates. This group allows us to represent in polynomial space (i) hypergraph states

and (ii) all circuits composed only of Z gates with zero, one, or two control qubits. We may even combine this group with, e.g., the Pauli group. More generally, whenever we have efficient algorithms constructing G-LIMDDs and H-LIMDDs for two groups G and H, we may consider constructing the J-LIMDD with $J = \langle G, H \rangle$. The LIMDD using the group of controlled-Z gates, proposed above, also has a natural classical analogue: two Boolean functions f, g are "Controlled-Z-isomorphic" if they are equal modulo a degree-2 multilinear polynomial q over $\mathbb{F}_2$, i.e., if $f(x) = q(x) \oplus g(x)$.

In the spirit of exploring the opportunities afforded by choosing a group G, it is natural to ask the following:

> **Research challenge 5.** *Prove succinctness and rapidity separations between G, H-LIMDDs for groups G, H.*

We have seen that the capability of a G-LIMDD to represent and manipulate quantum circuits is determined by the choice of the group G. For example, PAULI-LIMDDs can represent all stabilizer states, including coset states, but $\langle Z \rangle$-LIMDDs cannot efficiently represent coset states: PAULI-LIMDDs are therefore strictly more succinct and more rapid than $\langle Z \rangle$-LIMDDs. It is always true that, if $G \subseteq H$, then H-LIMDDs are at least as succinct as G-LIMDDs . Is this separation always strict when $G \subsetneq H$? Is there always a separation when $G \setminus H$ is non-empty? These questions would be answered by a map displaying how the choice of G affects the succinctness of the resulting G-LIMDD . On one hand, such a map may in principle be convoluted, since G may be chosen to be any one of the uncountably infinitely many groups of invertible matrices, i.e., subgroups of $GL(2)$. On the other hand, settling the following conjecture in the affirmative would fully resolve this question for succinctness (although not for rapidity).

Conjecture 8.1. If G and H are single-qubit groups, and if $G \setminus H$ is non-empty, then there are states with polynomial-size G-LIMDD but which have only exponential-size H-LIMDD. In fact, there is a Tower-G-LIMDD with this property.

> **Research challenge 6.** *When is measurement NP-hard in LIMDDs?*

In this work, we have given fast algorithms for measurement in G-LIMDDs for certain choices of G, namely for $G \subseteq$ Pauli. This is important to allow fast simulation of

quantum computation. These same algorithms would work equally well for any G-LIMDD where G contains only unitary matrices. However, if G contains non-unitary matrices, the algorithm that we presented must be modified, and the result is no longer polynomial-time. To see why, recall that the squared norm of an edge $\xrightarrow{\lambda P}\boxed{v}$ is $|\lambda^\dagger\lambda| \cdot |\langle v| P^\dagger P |v\rangle|$. Therefore, if G is unitary, then $P \in G$ is unitary, so $P^\dagger P = \mathbb{I}$, so we may recurse to compute the squared norm of node v. By contrast, a non-unitary P will not effect this cancellation, and the operator $P^\dagger P$ must be propagated to the recursive call; consequently, the runtime of the algorithm may become exponential, even with dynamic programming. It is natural to wonder whether this exponential running time is necessary, and if so, for which non-unitary groups G? To this end, we formulate the following conjecture for one such group.

Conjecture 8.2. Measurement in G-LIMDD is NP-Hard for $G = \{\left[\begin{smallmatrix} 1 & 0 \\ 0 & x \end{smallmatrix}\right] \mid x \in \mathbb{R}_{\neq 0}\}$. Specifically, given such a LIMDD for a quantum state $|\varphi\rangle$, it is NP-Hard to compute the probability of the outcome of a measurement of the first qubit. This remains true even when the LIMDD is a Tower.

8.3.2 Future work on DSDBDDs

Research challenge 7. *Is DSDBDD more rapid than BDD?*

We have shown that DSDBDD is more succinct than BDD. Bertacco, Damiano and Plazo gave an efficient way to convert a BDD to a DSDBDD [46, 264]. This suggests that one may apply the sufficient condition we formulated in Chapter 5 to show that DSDBDD is more rapid than BDD.

Research challenge 8. *Can we develop a Quantum DSDBDD?*

The definition of DSDBDD (Definition 6.2) effects a diagram which encodes a Boolean function (i.e, of the form $f\colon \{0,1\}^n \to \{0,1\}$), not a pseudo-Boolean function. It is not self-evident how a composition $k \circ g_1, \ldots, g_m$ of a kernel k with factors $g_1, \ldots, g_m$ should be defined for pseudo-Boolean functions unless the range of the factors g_j is reconciled with the domain of k. We make a suggestion here and defer other necessary considerations to future research, e.g., finding reduction rules to make the diagram canonical, and finding algorithms to keep the diagram reduced, noting that we believe

that such needs can be met. For any pseudoboolean function $k\colon \{0,1\}^n \to \mathbb{C}$, let $\overline{k}\colon \mathbb{C}^n \to \mathbb{C}$ be the unique multilinear extension of k. In more detail, any pseudoboolean function k has a unique expression as a weighted sum of monomials:

$$k(x_1, \ldots, x_n) = \sum_{S \subseteq \{x_1, \ldots, x_n\}} \alpha_S \prod_{x \in S} x \qquad \text{with } \alpha_S \in \mathbb{C} \tag{8.2}$$

This expression can be interpreted as a multilinear polynomial $\overline{k}$ over $\mathbb{C}$; this polynomial $\overline{k}$ is called the (unique) multilinear extension of k. Using the concept of multilinear extensions, we can define the composition of pseudoboolean functions as follows. Let $k, g_1, \ldots, g_m$ be pseudoboolean functions. Then the composition $h = k \circ (g_1, \ldots, g_m)$ is defined as the following pseudoboolean function h,

$$h(\vec{x}_1, \ldots, \vec{x}_m) = \overline{k}(g_1(\vec{x}_1), \ldots, g_m(\vec{x}_m)) \tag{8.3}$$

Thus, the composition of pseudoboolean functions yields another pseudoboolean function. For example,[‡] using the kernel $k(x_1, x_2) = x_1 \cdot x_2$, we get that $k \circ (|\varphi_1\rangle, |\varphi_2\rangle) = |\varphi_1\rangle \otimes |\varphi_2\rangle$. That is, we can easily recover the familiar tensor product by considering compositions of pseudoboolean functions. Which other familiar constructions might this tool elucidate?

8.3.3 Future work on SDDs

> **Research challenge 9.** *Can SDDs be augmented with disjoint support decompositions?*

We have seen that extending BDDs with DSDs significantly improves their succinctness (and perhaps their rapidity). Is the same true of SDDs? A good first step is to show that DSDs make SDDs more succinct (the example we provided in Chapter 6 likely suffices for this purpose). The more difficult second step is finding an algorithm which takes the SDD node of a Boolean function f and finds a decomposition $f = k \circ (g_1, \ldots, g_m)$, if one exists.

[‡]Some abuse of language happens in this example: we write $|\varphi\rangle$ where we mean the amplitude function of that state.

Research challenge 10. *Is conjunction of SDDs tractable?*

Van den Broeck and Choi raised this question [322]. They showed that the answer is *no* if the output vtree must be the same as the input vtrees. So any polynomial-time SDD conjunction algorithm, if it exists, must sometimes output an SDD with a different vtree than the input SDDs. A weaker question is to ask whether there always *exists* a polynomial-size SDD representing the conjunction of two DDs, regardless of the question of how to find it algorithmically.

8.3.4 Future work on quantum knowledge compilation

We sketch two avenues for future work in the area of knowledge compilation.

First, in our knowledge compilation map, we map the succinctness, tractability and rapidity of many popular quantum data structures. However, we omit the popular tensor network data structure. There is a rich literature on tensor networks employing various topologies, e.g., arranging the tensors on a line, on a grid, and in other ways [110, 248]. It is difficult to place these many variations on the knowledge compilation map in a meaningful way because the manipulation operations are often tractable, and indeed trivial (e.g., one "applies" an elementary gate to a tensor network in $\mathcal{O}(1)$ time by simply appending it to the open indices of an existing tensor network), whereas the queries are often intractable (they are #P-complete in many cases). Therefore, the many variations of tensor networks each look identical from the point of view of tractability. Moreover, if there are no (efficient) transformations which turn one type of tensor network into another, then rapidity results will be difficult to obtain. We leave it to future research to devise methodologies which allow to compare these data structures.

Second, in both our and Lai et al.'s [194] definition of rapidity, we require that a data structure be more rapid *on all instances*. However, it is often interesting to look at a restricted class of use cases, for example, one might be interested only in stabilizer circuits. Therefore, it is often natural tot ask whether one data structure is more rapid than another *on a particular family of inputs*.

> **Research challenge 11.** *Formulate and investigate a notion of rapidity restricted to a given family of inputs.*

For example, we have shown that restricted Boltzmann machines (RMBs) are incomparable to QMDDs in terms of rapidity, but we may expect that restricted Boltzmann machines are more rapid than QMDDs when restricted to analyzing stabilizer circuits. This allows for a more fine-grained comparison between two data structures which ordinarily would be incomparable in terms of rapidity. This may be an avenue for approaching the question raised above, where it is difficult to compare different versions of tensor networks: each may have their own strengths when analyzing particular families of circuits and this may be illuminated by a suitable notion of rapidity.

8.3.5 Future work on verification and decision diagrams

In this thesis, we have attacked the problem of quantum circuit equivalence checking. But quantum algorithm verification is broader than merely equivalence checking; therefore, we ask:

> **Research challenge 12.** *How can we do quantum algorithm verification beyond circuit equivalence checking?*

We see three ways in which algorithm verification is broader than merely circuit equivalence checking, which present three avenues for research:

- **Use a specification which is not another quantum circuit.**
  We have seen (in Section 2.4) that there are several temporal logics that are tailor-made for quantum applications, such as QCTL, QLTL, and quantum Hoare logic [30, 79, 100, 216, 217, 355, 359, 363, 366, 367]. Several model checking tools for such quantum temporal logics already exist (e.g., [116, 128, 161, 205]). There are good reasons to prefer using a temporal logic over circuit equivalence checking in certain situations: in such a logic, a user can specify the intended behaviour of an algorithm in a more natural and more fine-grained way than they can in circuit equivalence checking; e.g., in QCTL, it is possible to assert that two given qubits eventually become entangled [30]. Given that DD-based

techniques have been fruitful in the contexts of (i) classical model checking using similar temporal logics and of (ii) quantum circuit equivalence checking, we are optimistic that these same techniques can help bring the theory of quantum model checking with temporal logics into practice. To the best of our knowledge, no DD-based approach has been used in this context.

- **Verify an *algorithm* rather than a *circuit*.**
  An *algorithm* is often understood to receive an input of unbounded length, as opposed to a *circuit*, which takes an input of fixed length. For example, looking back at Chapter 4, even if we were to verify that an implementation of QFT is correct for $n = 2 \dots 24$ qubits, this would not guarantee correctness for $n = 25$. Therefore, the more important and more daunting task is to verify the correctness of the (classical) procedure which generates the circuits. The work of Amy et al. [13, 15], Hietala et al. [155] and Peng et al. [261] is in this direction, for example.

- **Equivalence checking of circuits with different numbers of ancilla qubits.**
  Two circuits may be understood to be equivalent, even if they use different numbers of ancilla qubits (this is akin to comparing two classical algorithms which use different amounts of memory). For example, multiple authors have cut down the number of ancilla qubits required for Shor's algorithm [35,309], but two such implementations can still be considered equivalent if the measurements at the end of the circuits produce the same outcomes. Therefore, ideally we would ask about the equivalence of two superoperators, rather than two unitary matrices. For example, Ardeshir-Larijani take this approach [17]; Hong et al. use DDs to check the (approximate) equivalence of two superoperators corresponding to two implementations [163]; and similarly Grurl, Fuß and Wille [141,143] use QMDDs to represent density matrices. Therefore, although steps have been taken in this direction, more research is called for, since this is the more natural use case in many applications. Moreover, this research goal could be a stepping stone to address the two research goals above.

Lastly, we ask how to compare decision diagrams in terms of the problems they can solve, rather than the languages they represent.

Research challenge 13. *Characterize the problems that can be solved with decision diagrams.*

This question is best illustrated by two motivating examples:

1. A BDD can succinctly represent the set of satisfying assignments of any CNF whose incidence graph has bounded treewidth [273]. In fact, such a BDD can be constructed in polynomial time, and (therefore) BDDs can efficiently decide its satisfiability. However, BDDs cannot succinctly represent the set $F \subset \mathsf{CNF}$ of satisfiable CNF formulas having bounded treewidth.

2. Say that a quantum circuit *accepts* if the probability of obtaining a 1 in the final measurement is greater than 1/2; otherwise it *rejects*. Then, since Pauli-LIMDDs can efficiently simulate Clifford circuits, including computing measurement outcome probabilities, they can efficiently decide whether a given Clifford circuit accepts or rejects. However, Pauli-LIMDDs cannot succinctly represent the set of accepting Clifford circuits.

We see that there is a difference between the problems a DD can *solve*, and the languages a DD can *represent*. The set of languages represented by a DD is traditionally denoted by a complexity class; e.g., the complexity class BDD contains all decision problems $L \subseteq \{0,1\}^*$ such that for each $n \geq 0$, the set $L_n = L \cap \{0,1\}^n$ has a polynomial-size BDD. However, the two examples above illustrate that it may be at least as interesting to consider the complexity class containing all decision problems solved by a given DD. The latter example in particular is remarkable because the task of simulating a Clifford circuit is $\oplus$L-complete[§] [3] (For our purposes, it suffices to say that $\oplus$L is a complexity class containing problems of considerable computational difficulty).

Therefore the question arises: given a decision diagram, which complexity class is characterized by the problems that can be efficiently solved using that decision diagram, for example, using a BDD, or a Pauli-LIMDD? We immediately obtain the following upper bound. Namely, by definition, if some DD $\mathcal{D}$ efficiently solves a decision problem

[§]Recall that L is the complexity class Logspace, of all problems solvable by a Turing Machine using $\mathcal{O}(\log n)$ space. Here $\oplus$L is the complexity class of problems solvable by a non-deterministic TM using $\mathcal{O}(\log n)$ space, and where the TM is defined to accept an input iff there is an odd number of accepting nondeterministic paths.

$L \subseteq \{0,1\}^*$, then using this decision diagram constitutes a polynomial-time algorithm which solves L, so L is in P. Consequently, if $\mathcal{C}_\mathcal{D}$ is the complexity class of problems efficiently solved using this DD, then $\mathcal{C}_\mathcal{D} \subseteq$ P. Can we design a decision diagram which achieves $\mathcal{C}_\mathcal{D} =$ P? Can we design decision diagrams which achieve other notable classes $\mathcal{C}_\mathcal{D}$? We leave it up to future research to more precisely define what it means for a decision diagram to "solve" a certain decision problem L.