NCON: A tensor network contractor for MATLAB

Robert N. C. Pfeifer,[1,*] Glen Evenbly,[2,†] Sukhwinder Singh,[3,‡] and Guifre Vidal[1]

[1]*Perimeter Institute for Theoretical Physics, 31 Caroline St. N, Waterloo, Ontario N2L 2Y5, Canada*
[2]*Institute for Quantum Information and Matter, California Institute of Technology, Pasadena CA 91125, USA*
[3]*Center for Engineered Quantum Systems; Dept. of Physics & Astronomy, Macquarie University, NSW 2109, Australia*

(Dated: October 3, 2016)

A fundamental process in the implementation of any numerical tensor network algorithm is that of *contracting* a tensor network. In this process, a network made up of multiple tensors connected by summed indices is reduced to a single tensor or a number by evaluating the index sums. This article presents a MATLAB function `ncon()`, or "Network CONtractor", which accepts as its input a tensor network and a *contraction sequence* describing how this network may be reduced to a single tensor or number. As its output it returns that single tensor or number. The function `ncon()` may be obtained by downloading the source of this preprint.

I. INTRODUCTION

Tensor network algorithms are a set of extremely powerful tools for the numerical simulation of quantum many-body systems. Examples include White's Density Matrix Renormalization Group (DMRG) algorithm,[1] which may be understood as a variational algorithm for optimization of the Matrix Product State (MPS) Ansatz and is arguably the preferred numerical method for the study of one-dimensional quantum systems, and more recent proposals such as Projected Entangled Pair States (PEPS)[2,3] and the Multi-scale Entanglement Renormalization Ansatz (MERA)[4–7] which offer the potential for scalable simulation of two dimensional quantum systems. Similarly important are several schemes for the contraction of a 2D tensor network representing either a partition function or a scalar product of two tensor network states, such as the Corner Transfer Matrix method (CTM),[8] the Tensor Renormalization Group (TRG),[9] and generalizations thereof.[10–12]

A task common to all of these algorithms is the need to evaluate the product of multiple tensors sharing one or more summed indices, frequently referred to as *contracting a tensor network*. Where two tensors share one or more summed indices, the evaluation of the sum over these indices is termed a *pairwise contraction*, and where two tensors are combined which do not share any indices, this is termed a *pairwise outer product*. Any tensor network may be contracted by means of a sequence of pairwise contractions and pairwise outer products, collectively called a *pairwise contraction sequence*, but the computational cost of this process may vary with the sequence chosen. It is known[13] that pairwise sequences are generally to be preferred over operations involving more than two tensors, and the problem of finding an optimal pairwise contraction sequence is discussed further in Ref. 13.

Given both a tensor network and a pairwise contraction sequence, the present article introduces a MATLAB function `ncon()` which contracts the given tensor network in a manner determined by the specified contraction sequence.

II. HISTORY AND MOTIVATION

As the problem of contracting a tensor network is so fundamental to the implementation of tensor network algorithms, there have been multiple pieces of software written over the years to perform this task. The function `ncon()` provided in this preprint may be considered a conceptual descendant of G. Vidal's earlier `scon()` function, which has enjoyed wide distribution and many informal modifications over the years. One objective in releasing `ncon()` is to provide a definitive update to `scon()`, incorporating in a single centralised piece of code all the features which have been added to various branches over time.

A second reason for releasing `ncon()` is that many versions of `scon()` contain code by multiple authors, frequently unattributed and seldom documented. In contrast the file `ncon.m` which implements the `ncon()` function is of known provenance, having been written from scratch by R.N.C. Pfeifer, and is centrally maintained. While `ncon()` does not make use of any pre-existing code, it is nevertheless appropriate to acknowledge where previous work has influenced the way in which `ncon()` is implemented:

- As mentioned above, the idea for `ncon()` has its origins in the earlier function `scon()` ("Sequential CONtractor") by G. Vidal.
- The method of efficiently implementing pairwise tensor contractions was inspired by the `con2t()` function of F. Verstraete. In this function the contraction of a pair of tensors is preceded by using the `permute()` command to collect together all the contracted and non-contracted indices on each tensor, and the `reshape()` command to combine these indices. The contraction is then realised as a matrix multiplication, which MATLAB performs using the BLAS library. Further reshaping and permuting

transforms the resulting matrix back into a tensor. This approach has become a *de facto* standard for the performance of tensor contractions in MATLAB.

Although the syntax of `ncon()` has deliberately been kept the same as that of `scon()`, allowing for maximum backward compatibility, with `ncon()` we have taken the opportunity to release a more fully-featured network contractor including support for disjoint networks, trivial (dimension 1) indices, 1D (and, under limited circumstances, 0D) objects, traces, outer products, and the zeros-in-sequence notation used by `netcon()` in Ref. 13.

III. USAGE

A. Obtaining the reference implementation

1. While viewing the abstract page for the latest version of this arXiv preprint, click "Download: Other formats".

2. Click "Download source".

3. Save the resulting file with extension ".tar". This file is an archive containing both the reference implementation and the LaTeX source for the instructions you are reading. (Note: The arXiv download page states that the file will be downloaded in .tar.gz format. This is incorrect; it is in .tar format only, and will not require unzipping with gzip.)

4. Unpack the archive using your preferred unarchiver (on a UNIX system you could use `tar xvf filename.tar`).

The MATLAB function `ncon()` is provided by the file `ncon.m`.

B. Using the reference implementation

Invocation of the MATLAB function `ncon()` takes the form

```
out = ncon(tensorList,legLinks,sequence,
                                  finalOrder);
```

where the input parameters are specified as follows:

`tensorList` is a $1 \times n$ cell array containing the n tensors which make up the tensor network.

`legLinks` describes the tensor network using leg-labelling notation. In brief, the tensor network is represented using the customary diagrammatic notation (for which a summary may be found in Ref. 14) and an integer label is assigned to each index (represented in the diagram by a leg). Summed indices are associated with positive integer labels, while open indices are associated with negative integer labels. The variable `legLinks` is

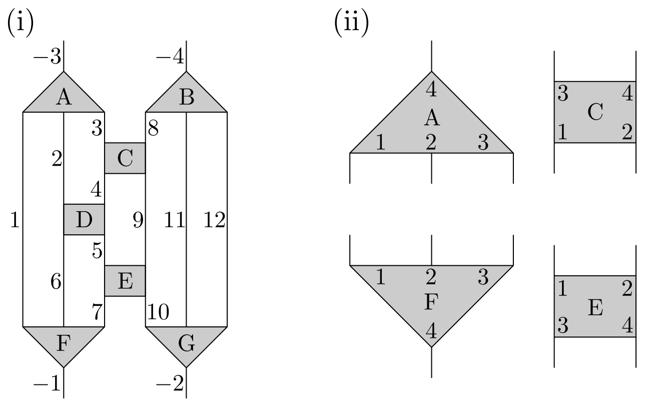

FIG. 1. (i) A tensor network diagram arising in the optimization of the 3:1 1D MERA. (ii) Ordering of indices on the tensors of diagram (i). Ordering for B is the same as for A. Ordering for D is the same as for C. Ordering for G is the same as for F. Note that ordering for tensor E differs from C and D because tensor E is customarily obtained from tensor C in the 3:1 1D MERA by complex conjugation and vertical reflection, and the process of reflection affects the leg ordering.

then a $1 \times n$ cell array with each entry being a row vector whose entries are the integer labels associated with the corresponding tensor. The ordering of these labels matches the ordering of the indices on the corresponding tensor in MATLAB. For example, Fig. 1(i) shows a diagram from the 3:1 1D MERA where all indices have been labelled with positive integers. A convention is adopted for relating the diagrammatic indices to indices in MATLAB, whereby the indices of a specific MATLAB tensor are associated with specific legs on the diagram. This is illustrated in Fig. 1(ii), where (for example) the topmost leg on tensor A is associated with the fourth index of the MATLAB tensor `A(:,:,:,:)`. If tensor A is the first object to appear in `tensorList` and tensor B is the second then, reading the labels associated with tensors A and B off the diagram of Fig. 1(i), `legLinks` takes the form `{[1 2 3 -3],[8 11 12 -4],...}`.

`sequence` is a row vector comprising positive integers and (optionally) zero, and specifies a contraction sequence for the closed tensor network. Assuming all indices in Fig. 1(i) are of identical dimension, denoted χ, an optimal index contraction sequence for this example network is

```
[5 4 9 6 7 1 2 3 11 12 8 10],
```

having a cost of $2\chi^8 + 2\chi^7 + 2\chi^6$. Interpretation of this sequence proceeds as follows: The first entry in the sequence is index 5, connecting tensors D and E. The first step of the contraction sequence is therefore to contract together these two tensors, denoted (DE). The next entry in the sequence is index 4, connecting the resulting object to tensor C, indicating that the next contraction is

$$((DE)C).$$

This contraction is performed simultaneously over all indices common to both (DE) and C, and therefore also accounts for index 9.[15] Proceeding in this fashion for the entirety of the index list, one obtains the pairwise contraction sequence

$$(((((\mathrm{DE})\mathrm{C})\mathrm{F})\mathrm{A})(\mathrm{BG})).$$

Note that if the `netcon()` reference implementation given in Ref. 13 is installed then this may be used to automatically generate an optimal index-based contraction sequence for a tensor network.

The `ncon()` function includes support for contraction sequences involving outer products, either where two tensors are contracted despite not sharing an index or where the tensors share only indices of dimension 1, and the appropriate syntaxes for `sequence` in these situations are discussed in Sec. V. Further discussion of the index-labelling notation for contraction sequences may also be found in Ref. 13, including an explicit algorithm for converting sequences of index labels into sequences of pairwise tensor contractions.

Note that the argument `sequence` is optional, and if not specified it will default to the list of all positive indices in ascending numerical order.

Finally, a fourth argument, `finalOrder`, which is also optional, may be used to modify the ordering of indices on the output tensor. By default these indices correspond to the negative entries in `legLinks` in descending numerical order, so that indices 1, 2, 3, 4, etc. of the output tensor correspond to labels `-1`, `-2`, `-3`, `-4`, etc. respectively. This behaviour may be changed by specifying a sequence for the negative labels in `finalOrder`, e.g.

```
finalOrder = [-3 -1 -2 -4 ...].
```

The first index of the output tensor would then correspond to the index in `legLinks` which bears label `-3`, and so forth. As the order of the negatively-labelled legs is explicitly specified, the requirement that these legs be numbered consecutively from -1 is also lifted when `finalOrder` is given.

The `ncon()` function incorporates substantial error checking of user-supplied input. The computational cost of this is small, but when invoking `ncon()` from stable code it may be desireable to disable the checks in exchange for a slight increase in performance. This may be achieved by declaring a global variable `ncon_skipCheckInputs` and setting its value to true:

```
global ncon_skipCheckInputs;
ncon_skipCheckInputs = true;
```

This setting will persist until the variable is cleared from the global workspace or its value is changed to anything other than `true`. For users unfamiliar with global variables, the above command should be understood as turning off error checking in `ncon()` until you exit MATLAB, issue a `clear all` command, or type

```
global ncon_skipCheckInputs;
ncon_skipCheckInputs = false;
```

to turn it back on again. To avoid confusion between global and local variables, it is recommended that the variable name `ncon_skipCheckInputs` should not be used for any other purpose.

IV. SIMPLE EXAMPLES

In this Section, the syntax of `ncon()` described in Sec. III B is illustrated with two relatively simple examples.

A. Matrix multiplication

The first example is that of matrix multiplication,

$$C = AB,$$

where A and B are matrices. Using index notation, this calculation may be written

$$\mathrm{C}_{\alpha\beta} = \sum_{\gamma} \mathrm{A}_{\alpha\gamma}\mathrm{B}_{\gamma\beta}. \tag{1}$$

Using `ncon()`, matrix C may be constructed using the commands

```
tensors = {A,B};
legLinks = {[-1 1],[1 -2]};
sequence = 1;
C = ncon(tensors,legLinks,sequence);
```

or more directly,

```
C = ncon({A,B},{[-1 1],[1 -2]},1);
```

For example, if A and B are created using the commands

```
A = rand(3,4);
B = rand(4,6);
```

then the output of the above command is seen to be exactly equal to that obtained by typing

```
C = A*B;.
```

There is seldom any reason to perform a simple matrix multiplication using `ncon()` instead of the multiplication operator `*`, but this serves as a simple illustrative example of the syntax of the `ncon()` command.

B. Matrix and two vectors

A slightly more complicated example is given by the expression

$$c = x^{\mathrm{T}} M y \tag{2}$$

where M is a matrix and x and y are column vectors. In index notation,

$$c = \sum_{\alpha,\beta} x_\alpha M_{\alpha\beta} y_\beta. \tag{3}$$

Once again, c may be calculated in MATLAB using the multiplication operator

```
c = (x.')*M*y;
```

or using `ncon()`. This time the syntax for the `ncon()` command is

```
tensors = {x,M,Y};
legLinks = {[1],[1 2],[2]};
sequence = [1 2];
c = ncon(tensors,legLinks,sequence);
```

or more briefly,

```
c = ncon({x,M,y},{[1],[1 2],[2]},[1 2]);
```

where the sequence [1 2] indicates this is to be evaluated as

$$c = (x^{\mathrm{T}} M) y \tag{4}$$

in contrast to a contraction sequence of [2 1] which would correspond to

$$c = x^{\mathrm{T}} (M y). \tag{5}$$

Which sequence is to be preferred will depend upon the relative lengths of vectors x and y, with the calculation proceeding more rapidly if the longer vector is multiplied with M first, though the value of c obtained will of course be unaffected by this choice.

V. SEQUENCES INVOLVING OUTER PRODUCTS

On occasion, the optimal contraction sequence for a tensor network may necessarily involve an outer product of two or more tensors. The `ncon()` function supports three distinct methods of specifying an outer product. Each has its own benefits and drawbacks.

Note that the discussion of outer products in this Section closely parallels (and partially reproduces) that of Appendix B in Ref. 16. Readers familiar with the function `multienv()` presented in Ref. 16 are encouraged to read Sec. V A on disjoint networks, as this feature is substantially more powerful in `ncon()` than in `multienv()`, and also to review the example invocations of `ncon()` presented in the rest of the Section and to compare these with their counterparts in Appendix B of Ref. 16. Readers not familiar with `multienv()` may be reassured that familiarity with `multienv()` is not required in order to make effective use of `ncon()`.

A. Disjoint networks

As the first method of describing a contraction sequence involving outer products, one may specify a disjoint tensor network, e.g.

```
A = rand(2,2);B = rand(2,2);C = rand(2,2);
D = ncon({A,B,C},{[-1 1],[1 -3],[-2 -4]},[1]).
```

After contracting over index 1, the resulting network is made up of two disconnected tensors. Writing

$$E_{\alpha\beta} = \sum_{\gamma} A_{\alpha\gamma} B_{\gamma\beta} \tag{6}$$

where γ corresponds to index 1, the desired output $D_{\alpha\beta\gamma\delta}$ is given by the outer product

$$D_{\alpha\beta\gamma\delta} = E_{\alpha\gamma} C_{\beta\delta}. \tag{7}$$

This situation is recognised by `ncon()`, and the outer product is automatically performed in order to explicitly return the requested object, in this instance $D_{\alpha\beta\gamma\delta}$. This method of specifying an outer product is simple and can even be used when the contraction sequence is empty, e.g.

```
F = ncon({A,B,C},{[-1 -3],[-2 -4],[-5 -6]}),
```

corresponding to $F_{\alpha\beta\gamma\delta\epsilon\zeta} = A_{\alpha\gamma} B_{\beta\delta} C_{\epsilon\zeta}$. Where such an outer product is performed across three or more tensors, as here, it is automatically evaluated in the most efficient manner possible (being a series of pairwise contractions between the tensors of smallest total dimension). The primary limitation of this method is that it is only capable of specifying outer products occuring as the final steps of the contraction sequence.

B. Trivial indices

As a second method one may introduce explicit connecting indices of dimension 1. Consider an example, adapted from Appendix B 2 of Ref. 16, where the dimensions of tensors B and C are explicitly $3 \times 1 \times 3$ and

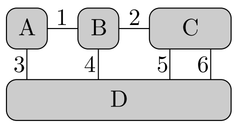

FIG. 2. Example tensor network with trivial index. All indices have dimension 3 except for the index labelled 2, which has dimension 1.

$3 \times 3 \times 1$ respectively:[17]

```
A = rand(3,3); B = rand(3,1,3);
C = rand(3,3,1); D = rand(3,3,3,3);
tensors = {A,B,C,D};
legs = {[3 1],[1 2 4],[5 6 2],[3 4 5 6]};
seq = [1 2 3 4 5 6];
E = ncon(tensors,legs,seq);
```

This tensor network is illustrated in Fig. 2, and the given index contraction sequence corresponds to a pairwise tensor contraction sequence of (((AB)C)D). The outer product is between the contraction product (AB) and the tensor C, and corresponds to contraction over the index of dimension 1 carrying the numerical label 2. This label appears in the index-based contraction sequence in the usual manner.

The inclusion of trivial indices is the most versatile means of specifying an outer product as part of the contraction sequence, with the only drawback being the need to explicitly include the indices of dimension 1 over which the contractions are to be performed.

C. Zeros-in-`sequence` notation

In Ref. 13 it was shown that for any tensor network, if an outer product of two or more tensors is *required* as part of the optimal contraction sequence and the result of this outer product is denoted Y, then an optimal contraction sequence may always be found where this outer product is always either

1. the final step in the contraction of the tensor network, or
2. followed by contracting *all* indices of object Y with another tensor, which we will denote X (as per the final Appendix of Ref. 13).

Outer products of these forms (and these forms only) may be represented by inserting zeros into the contraction sequence, with the outer product of n tensors being indicated by $n-1$ consecutive zeros. To determine the n tensors involved in the outer product when there are more than n tensors remaining, indices are read from the sequence after the zeros, and the tensors carrying these indices are noted, until $n+1$ tensors have been identified. Given the above constraints on the outer products which may be represented using this notation, it then follows that one of these $n+1$ tensors will necessarily share summed indices with all n other tensors. This tensor does not participate in the outer product, but is instead the tensor X which is subsequently contracted with object Y. For a fuller discussion of this outer product notation, and of the optimal performance of the outer products of multiple tensors, see Ref. 13. A simple example is given by

```
A = rand(2,2);B = rand(2,2);C = rand(2,2);
D = ncon({A,B,C},{[-1 1],[1 -3],[-2 -4]},[1 0]);
```

and illustrated in Fig. 3(i), and a more complicated example is given by

```
A = rand(2,1);B = rand(2,2);C = rand(2,1);
D = rand(2,2,2,2);
tensors = {A,B,C,D};
legs = {[1],[1 2],[3],[2 3 -1 -2]};
D = ncon(tensors,legs,[1 0 2 3]);
```

where the index sequence `[1 0 2 3]` corresponds to a tensor contraction sequence (((AB)C)D) and the network is illustrated in Fig. 3(ii).

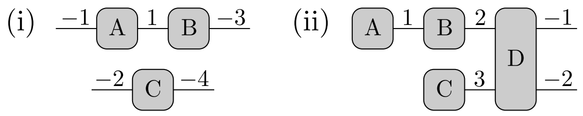

FIG. 3. Example tensor networks; all indices have dimension 2. (i) An index sequence of `[1 0]` corresponds to the pairwise tensor contraction sequence ((AB)C) where the second contraction to be performed is an outer product. (ii) An index sequence of `[1 0 2 3]` corresponds to the pairwise tensor contraction sequence ((AB)C)D) where, once again, the second contraction to be performed is an outer product.

While this approach is not as versatile as the use of trivial indices discussed in Sec. V B, it is nevertheless useful when an optimal contraction sequence is being automatically generated, for instance by using the Netcon algorithm.[13] Because there always exists an optimal contraction sequence which may be described using the zeros-in-`sequence` notation, a search algorithm for optimal sequences can return a valid response in this notation even when all optimal contraction sequences involve the performance of an outer product for which no appropriate index of dimension 1 has been provided.

As `ncon()` supports the zeros-in-`sequence` notation for outer products, it is capable of directly accepting contraction sequences generated by the reference implementation of Netcon included with Ref. 13. Note that when using the zeros-in-`sequence` notation, all outer products must be explicitly represented in the sequence and not left implicit as in Sec. V A.

VI. HOW IT WORKS

A. Contraction of tensor networks

The operation of `ncon()` is as follows:

- Basic validity checks are performed on the input data.
- The main loop of `ncon()` (which is contained within the function `performContraction()` in `ncon.m`) steps through the contraction sequence as follows:
  - If the first entry in the sequence is an index appearing on two tensors, then these tensors are to be contracted.
    - ∗ If reading further consecutive entries from the sequence yields additional indices which connect the same two tensors, these indices are also contracted over at the same time (for example indices 4 and 9 in Sec. III B).
    - ∗ If there are other indices connecting these two tensors which appear later in the sequence, then these indices are not contracted over at this time. Such a sequence is suboptimal and a warning is generated.
    - ∗ Pairwise tensor contraction is performed by the function `tcontract()` in `ncon.m`.
  - If the first entry is an index appearing twice on one tensor, then it is a trace to be performed on that tensor.
    - ∗ If reading further consecutive entries from the sequence yields additional indices corresponding to traces on the same tensor, these indices are also traced over at the same time.
    - ∗ If there are other indices corresponding to traces on the same tensor that appear later in the sequence, then these indices are not contracted over at this time. Such a sequence is suboptimal and a warning is generated.
    - ∗ Evaluation of traces is performed by the function `doTrace()` in `ncon.m`.
  - If the first entry is a zero, the corresponding outer product is determined according to the algorithm given in Appendix E of Ref. 13.
    - ∗ Parsing of zeros-in-`sequence` notation is performed by the function `zisOuterProduct()` in `ncon.m`.
  - Whatever the nature of the entry, the indices contracted over (or the zeros which have been processed) are then deleted from the sequence.
- Finally, the indices of the output object are arranged as specified by the negative indices appearing in `legLinks` and (optionally) the input parameter `finalOrder`.

B. Contraction of tensor pairs

As noted in Sec. II, the implementation of pairwise tensor contraction in `ncon()` is achieved through use of the MATLAB `permute()` and `reshape()` commands, and matrix multiplication. One consequence of this approach is that objects thus constructed are always themselves full tensors. Consequently, if given an expression such as

$$A^{\alpha\beta\gamma\delta}_{\varepsilon}B^{\zeta}_{\beta\gamma\delta\eta} \tag{8}$$

one may efficiently contract tensors A and B over indices β, γ, and δ to yield

$$T^{\alpha\zeta}_{\varepsilon\eta} = A^{\alpha\beta\gamma\delta}_{\varepsilon}B^{\zeta}_{\beta\gamma\delta\eta}, \tag{9}$$

but contraction over only a subset of these indices, say β and γ, is intrinsically inefficient, corresponding to computation of all entries in the object

$$T'^{\alpha\delta\zeta}_{\varepsilon\theta\eta} = A^{\alpha\beta\gamma\delta}_{\varepsilon}B^{\zeta}_{\beta\gamma\theta\eta} \tag{10}$$

when we will subsequently be performing the trace

$$T^{\alpha\zeta}_{\varepsilon\eta} = \sum_{\delta} T'^{\alpha\delta\zeta}_{\varepsilon\delta\eta} \tag{11}$$

and so only the entries

$$T'^{\alpha\delta\zeta}_{\varepsilon\theta\eta}\Big|_{\delta=\theta} \tag{12}$$

are actually required. Calculation of the full tensor T' is wasteful of computational resources, and consequently index sequences which do not contract over all shared indices at once are suboptimal in both time and memory usage.

It is interesting to compare this approach with the use of `for` loops to iterate over the entries in A and B as in Ref. 18. With explicit control over each individual index, rather than computing the entirety of tensor T' one may recognise that a trace will subsequently be performed over indices δ and θ, and only evaluate the elements

$$T'^{\alpha\delta\zeta}_{\varepsilon\theta\eta}\Big|_{\theta=\delta} = A^{\alpha\beta\gamma\delta}_{\varepsilon}B^{\zeta}_{\beta\gamma\theta\eta}\Big|_{\theta=\delta}. \tag{13}$$

Taking this approach, the unnecessary computational cost associated with evaluating all entries in T' is eliminated. Such an approach is still inefficient in memory, however, as prior to performing the sum it stores a separate entry for each value of δ, and consequently when contracting a single tensor network it is always preferable to contract over all shared indices simultaneously unless the calculation of intermediate objects such as T' is specifically desired. In Ref. [18] this is addressed by observing that whenever an index is repeated on a single tensor, it is always appropriate to immediately evaluate the sum over that index.

Note that while `ncon()` is able to detect inefficiencies of this sort and thus may sometimes recognise if a contraction sequence is suboptimal, the task of identifying an optimal contraction sequence for a given tensor network is non-trivial.[13] The absence of a warning does not indicate that the contraction sequence supplied to `ncon()` is optimal; merely that no obvious indicators of a suboptimal sequence were detected.

ACKNOWLEDGMENTS

R.N.C.P. thanks the Ontario Ministry of Research and Innovation Early Researcher Awards for financial support. G.E. is supported by the Sherman Fairchild foundation. S.S. acknowledges financial support from the MQNS grant by Macquarie University Grant No. 9201200274. Research at Perimeter Institute is supported by the Government of Canada through Industry Canada and by the Province of Ontario through the Ministry of Research and Innovation. This research was supported in part by the ARC Centre of Excellence in Engineered Quantum Systems (EQuS), Project No. CE110001013.

[*] rpfeifer@perimeterinstitute.ca
[†] evenbly@caltech.edu
[‡] sukhwinder.singh@mq.edu.au
[1] S. R. White, Phys. Rev. Lett. **69**, 2863 (1992).
[2] F. Verstraete and J. I. Cirac, arXiv:cond-mat/0407066v1 (2004).
[3] J. Jordan, R. Orús, G. Vidal, F. Verstraete, and J. I. Cirac, Phys. Rev. Lett. **101**, 250602 (2008).
[4] G. Vidal, Phys. Rev. Lett. **99**, 220405 (2007).
[5] L. Cincio, J. Dziarmaga, and M. M. Rams, Phys. Rev. Lett. **100**, 240603 (2008).
[6] G. Evenbly and G. Vidal, Phys. Rev. Lett. **102**, 180406 (2009).
[7] G. Vidal, in *Understanding Quantum Phase Transitions*, edited by L. D. Carr (Taylor & Francis, Boca Raton, 2010).
[8] T. Nishino and K. Okunishi, J. Phys. Soc. Jpn. **66**, 3040 (1997).
[9] M. Levin and C. P. Nave, Phys. Rev. Lett. **99**, 120601 (2007).
[10] Z.-C. Gu, M. Levin, and X.-G. Wen, Phys. Rev. B **78**, 205116 (2008).
[11] Z.-C. Gu and X.-G. Wen, Phys. Rev. B **80**, 155131 (2009).
[12] H. H. Zhao, Z. Y. Xie, Q. N. Chen, Z. C. Wei, J. W. Cai, and T. Xiang, Phys. Rev. B **81**, 174411 (2010).
[13] R. N. C. Pfeifer, arXiv:1304.6112 [cond-mat.str-el] (2013).
[14] R. N. C. Pfeifer, *Simulation of Anyons Using Symmetric Tensor Network Algorithms*, Ph.D. thesis, The University of Queensland (2011), arXiv:1202.1522v2 [cond-mat.str-el].
[15] Note that when two tensors are contracted together, all indices connecting these tensors should be listed consecutively. Sequences which do not satisfy this condition (i) are always of suboptimal efficiency (see also Sec. VI B), and (ii) will result in a warning being generated.
[16] G. Evenbly and R. N. C. Pfeifer, arXiv:1310.8023 [cond-mat.str-el] (2013).
[17] Note that MATLAB leaves trailing indices of dimension 1 implicit, and so will report the size of tensor C as 3×3 if queried, and not $3 \times 3 \times 1$. Equally, it is also valid to create tensor C using the command `C = rand(3,3);` instead of `C = rand(3,3,1);` as given above.
[18] C.-C. Lam, P. Sadayappan, and R. Wenger, Parallel Processing Letters **07**, 157 (1997).