Efficient computation of multiple environments (reference implementation)

G. Evenbly[1] and Robert N. C. Pfeifer[2]

[1] *Institute for Quantum Information and Matter, California Institute of Technology, Pasadena CA 91125, USA**

[2] *Perimeter Institute for Theoretical Physics, 31 Caroline St. N, Waterloo, Ontario N2L 2Y5, Canada*†

(Dated: February 7, 2014)

I. USING THE REFERENCE IMPLEMENTATION

The reference implementation of the approach to computing multiple tensor environments described in Sec. IV E is provided in the form of a MATLAB function `multienv()`, contained within the file `multienv.m`. Invocation takes the form

```
[env1 env2 ...] = multienv(tensorList,envList,
                           legLinks,sequence);
```

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.

`envList` is a $1 \times n$ array of integers. If an entry in `envList` is non-zero then the environment of the corresponding tensor is computed and returned in the specified output variable. For example, if the tensor list is {`A,B,C,D,E`} and `envList` is `[0 2 0 0 1]` then `multienv()` will return two tensors `env1` and `env2` corresponding to the environments of tensors `E` and `B` respectively. If an integer is repeated, the corresponding environments are added together. Thus an `envList` of `[0 1 0 0 1]` would cause `multienv()` to return a single tensor, being the sum of the environments of `B` and `E`.

`legLinks` describes the tensor network using the leg-labelling notation given in Ref. 1. In brief, the tensor network is represented using the customary diagrammatic notation (for which a summary may be found in Ref. 2) 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. In the present context it is required that the tensor network have no open indices, so only positive integer labels are required. The variable `legLinks` is 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 closed 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 13],[8 11 12 14],...`}.

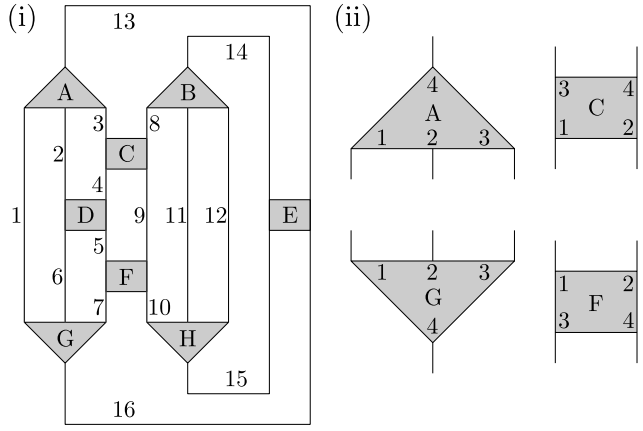

FIG. 1. (i) A closed tensor network diagram arising in the optimization of the 3:1 1D MERA. (ii) Ordering of indices on the tensors of diagram (i). Ordering for B is the same as for A. Ordering for D and E is the same as for C. Ordering for H is the same as for G. Note that ordering for tensor F differs from C, D, and E because tensor F 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.

`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

```
[11 12 14 15 7 6 5 4 9 8 10 16 1 2 3 13],
```

having a cost of $2\chi^8 + 2\chi^7 + 2\chi^6 + \chi^4$. Interpretation of this sequence proceeds as follows: The first entry in the sequence is index 11, connecting tensors B and H. The first step of the contraction sequence is therefore to contract together these two tensors, denoted (B,H). This contraction is performed simultaneously over all indices common to both B and H, and therefore also accounts for index 12.[3] The next entry in the sequence is index 14, connecting the resulting object to tensor E, indicating

that the next contraction is

$$((\mathrm{B},\mathrm{H}),\mathrm{E}).$$

Proceeding in this fashion for the entirety of the index list, one obtains the pairwise contraction sequence

$$((((\mathrm{B},\mathrm{H}),\mathrm{E}),(((\mathrm{F},\mathrm{G}),\mathrm{D}),\mathrm{C})),\mathrm{A}).$$

Note that if the `netcon()` reference implementation given in Ref. 1 is installed then this may be invoked to automatically generate an optimal index-based contraction sequence for a tensor network.

The `multienv()` 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 Appendix III. Further discussion of the index-labelling notation for contraction sequences may be found in Ref. 1, including an explicit algorithm for converting sequences of index labels into sequences of pairwise tensor contractions.

II. EXAMPLE

The tensor network given in Fig. 1 is encountered when variationally optimizing the 3:1 1D MERA in order to compute the ground state of a local Hamiltonian. Assuming translation invariance of the Hamiltonian these tensors are taken to satisfy A = B, F = $\mathrm{C}^\dagger$, and G = H = $\mathrm{A}^\dagger$, where † represents a combination of complex conjugation with vertical reflection in the diagrammatic notation. The tensor which appears in both positions A and B is known as an *isometry* tensor. The environments to be computed from this diagram are:

1. The environment of E, corresponding to a contribution to the *lifting* of the Hamiltonian (to be returned in `env1`).
2. The environment of D, corresponding to a contribution to the *lowering* of the reduced density matrix (to be returned in `env2`).
3. The environments of positions A and B, used in the variational update of the isometry tensor (to be summed and returned in `env3`).
4. The environment of C, used in the variational update of tensor C (to be returned in `env4`).

To efficiently compute these environments one may invoke `multienv()` as follows:

```
tensorList = {A,A,C,D,conj(C),conj(A),conj(A),E};
envList = [3 3 4 2 0 0 0 1];
legLinks = ...
  {[1 2 3 13],[8 11 12 14],[4 9 3 8],[6 5 2 4],[5 9 7 10],[1 6 7 16],[10 11 12 15],[15 16 14 13]};
sequence = [11 12 14 15 7 6 5 4 9 8 10 16 1 2 3 13];
[env1 env2 env3 env4] = multienv(tensorList,envList,legLinks,sequence);
```

The total cost for this invocation is $5\chi^8+4\chi^7+5\chi^6$, which may be compared with a cost of $10\chi^8 + 10\chi^7 + 10\chi^6$ if intermediate tensors are not re-used.

III. SEQUENCES INVOLVING OUTER PRODUCTS

When calculating the environments of tensors in a closed tensor network $\mathcal{N}$, the need to compute outer products may arise in two distinct situations. The first is where the user-specified contraction sequence for a closed network $\mathcal{N}$ involves the performance of outer products between two or more tensors appearing in $\mathcal{N}$. The second is where removal of a tensor T from a network $\mathcal{N}$ causes that network to separate into two or more disjoint parts (or, if $\mathcal{N}$ is already disjoint, causes a non-disjoint subnetwork of $\mathcal{N}$ to separate into two or more disjoint parts). Calculation of the environment of T then necessarily involves taking the outer product of these parts. These two situations are not mutually exclusive, and calculation of a tensor environment may include both user-specified outer products and outer products which arise when removing individual tensors to compute their environments.

The second situation—where outer products arise upon deleting a tensor T from a closed network $\mathcal{N}$—poses no challenges beyond those already discussed within this paper, and is handled automatically by `multienv()`. This Appendix, on the other hand, concerns itself with the syntaxes whereby a user may specify a contraction se-

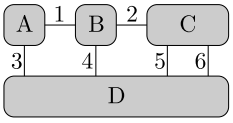

FIG. 2. Example tensor network with trivial index. All indices have dimension 3 except for the index labelled 2, which has dimension 1.

quence to `multienv()` which explicitly involves the performance of outer products.

A. Disjoint networks

When invoking `multienv()` the supplied tensor network $\mathcal{N}$ may be disjoint, being made up of n separate closed subnetworks $\mathcal{N}_1, \ldots, \mathcal{N}_n$. The problem of contracting network $\mathcal{N}$ to a number may then be subdivided into the problems of contracting each subnetwork to a number, and multiplying these numbers together. As a number may be understood as a tensor of dimension 0, formally these final multiplications correspond to outer products between tensors of dimension 0. If `multienv()` is supplied with a disjoint network $\mathcal{N}$ and a contraction sequence which reduces all subnetworks to numbers, the contraction tree for network $\mathcal{N}$ is then completed by performing the pairwise contraction (multiplication) of these numbers until only a single numerical value remains. Contraction sequences for the calculation of individual tensor environments are then determined in the normal way by manipulating the resulting fusion tree for the whole of $\mathcal{N}$ as described in Sec. IV.

B. Trivial indices

To perform outer products between tensors of dimension greater than zero during the contraction of a tensor network $\mathcal{N}$, one may introduce labelled connecting indices of dimension 1. Consider the example

```
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};
envList = [0 0 0 1];
legs = {[3 1],[1 2 4],[5 6 2],[3 4 5 6]};
seq = [1 2 3 4 5 6];
envD = multienv(tensors,envlist,legs,seq);
```

where the dimensions of tensors B and C are $3 \times 1 \times 3$ and $3 \times 3 \times 1$ respectively.[4] This tensor network is illustrated in Fig. 2, and the given index contraction sequence corresponds to a pairwise tensor contraction sequence of (((A,B),C),D). The outer product is between the contraction product (A,B) 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 indices of dimension 1 over which the contractions are to be performed.

C. Zeros-in-sequence notation

While any contraction sequence involving outer products may always be described using the trivial index approach of Appendix III B, it is sometimes useful to use a different notation involving the insertion of zeros into the index sequence passed to `multienv()`. This notation was introduced in Ref. 1 as a means of describing contraction sequences involving outer products when appropriate trivial indices are not present. Although this notation is not capable of describing every contraction sequence which involves an outer product, for any given tensor network there always exists an optimal (minimal cost) contraction sequence which can be described using this notation.

In brief, in Ref. 1 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.

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 hat 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 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 mul-

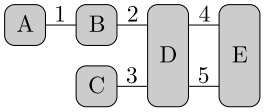

FIG. 3. Example tensor network; all indices have dimension 2. An index sequence of `[1 0 2 3 4 5]` corresponds to the pairwise tensor contraction sequence (((A,B),C),D),E) where the second contraction to be performed is an outer product.

tiple tensors, see Ref. 1. An example is given by

```
A = rand(2,1); B = rand(2,2); C = rand(2,1);
D = rand(2,2,2,2); E = rand(2,2);
tensors = {A,B,C,D,E};
envList = [0 0 0 0 1];
legs = {[1],[1 2],[3],[2 3 4 5],[4 5]};
seq = [1 0 2 3 4 5];
envE = multienv(tensors,envList,legs,seq);
```

where the index sequence `[1 0 2 3 4 5]` corresponds to a tensor contraction sequence ((((A,B),C),D),E) and the network is illustrated in Fig. 3.

While this approach is not as versatile as the use of trivial indices discussed in Appendix III B, it is nevertheless useful when an optimal contraction sequence is being automatically generated, for instance by using the Netcon algorithm.[1] 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 performing an outer product and no appropriate index of dimension 1 has been provided.

By supporting the zeros-in-`sequence` notation for outer products, `multienv()` is capable of directly accepting contraction sequences generated by the reference implementation of Netcon provided with Ref. 1.

---

* evenbly@caltech.edu
† rpfeifer@perimeterinstitute.ca

[1] R. N. C. Pfeifer, arXiv:1304.6112 [cond-mat.str-el] (2013).
[2] 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].
[3] Note that when two tensors are contracted together, all indices connecting these tensors must be listed consecutively. Sequences which do not satisfy this condition (i) are always of suboptimal efficiency, and (ii) are unsupported, and will result in a warning.
[4] 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.