PAPER • OPEN ACCESS

Algorithms for tensor network contraction ordering

MACHINE LEARNING
Science and Technology

PAPER

Algorithms for tensor network contraction ordering

Frank Schindler[1,2] and Adam S Jermyn[3,2]

[1] Department of Physics, University of Zurich, Winterthurerstrasse 190, 8057 Zurich, Switzerland
[2] Kavli Institute for Theoretical Physics, University of California, Santa Barbara, CA 93106, United States of America
[3] Center for Computational Astrophysics, Flatiron Institute, New York, NY 10010, United States of America

E-mail: adamjermyn@gmail.com and frank.schindler@uzh.ch

Abstract

Contracting tensor networks is often computationally demanding. Well-designed contraction sequences can dramatically reduce the contraction cost. We explore the performance of simulated annealing and genetic algorithms, two common discrete optimization techniques, to this ordering problem. We benchmark their performance as well as that of the commonly-used greedy search on physically relevant tensor networks. Where computationally feasible, we also compare them with the optimal contraction sequence obtained by an exhaustive search. Furthermore, we present a systematic comparison with state-of-the-art tree decomposition and graph partitioning algorithms in the context of random regular graph tensor networks. We find that the algorithms we consider consistently outperform a greedy search given equal computational resources, with an advantage that scales with tensor network size. We compare the obtained contraction sequences and identify signs of highly non-local optimization, with the more sophisticated algorithms sacrificing run-time early in the contraction for better overall performance.

1. Introduction

Tensor networks are a convenient language for studying the statistics of discrete systems with local interactions. The partition function and correlation functions of many lattice models may be written as tensor networks. Similarly, typical states of quantum systems often admit an efficient representation as a tensor network, either in the form of matrix product states (MPS) [1, 2] or more general states such as tree tensor networks [3, 4] and projected entangled pair states (PEPS) [5]. Tensor networks have also been used as machine learning classifiers [6, 7].

At the core of these applications is the problem of tensor network contraction, in which all intermediate bonds in a tensor network are summed to evaluate the network. Because these sums are performed simultaneously, a naive tensor network contraction uses computational resources which are exponential in system size, and so better approaches are needed.

Unfortunately, contracting tensor networks requires exponential resources in general [8]. Nonetheless, it is often possible to approximate tensor network contraction, resulting in efficiently-computable answers with controllable errors [9, 5, 11–14]. Moreover, many useful tensor networks, including MPS networks [1], can be contracted exactly in polynomial time by taking advantage of the property that the computational cost of contracting a tensor network depends strongly on the order of summation while the result does not. Hence while the worst cases may be intractable, there is still room to improve in typical or special cases.

To that end, we examine two algorithms which are widely used in discrete optimization. Our aim is to see if these algorithms provide any improvement over standard methods and hand-crafted contraction sequences. These are Genetic Algorithms [15–17] and Simulated Annealing [18, 19]. We begin in section 2 by reviewing the structure of tensor networks, Penrose notation, and the computational cost of contracting sequences. In section 3 we then describe the algorithms in more detail, along with the commonly-used Greedy Search algorithm [20] and the reference Exhaustive Search method [21–24]. We then perform numerical experiments in section 4, testing these methods on both two-dimensional square tensor networks and Erdös-Rényi random networks, and find that both algorithms outperform the Greedy Search in most of

our experiments, often by many orders of magnitude. We examine specific contraction sequences in section 5 to understand how these algorithms craft such efficient sequences. In section 6 we apply our methods to random regular graphs, which enables a direct comparison with recent state-of-the-art approaches to tensor network contraction. We then conclude with a discussion of our results in section 7.

2. Tensor network contraction

A tensor network is a list of tensors along with a specification of which pairs of their indices are meant to be contracted. So for instance,

$$N_{klmn} = \sum_{ij} T_{ijkl} X_i Y_{jmn} \tag{1}$$

specifies a network N formed of three tensors T, X and Y, with two contracted pairs of indices, namely i and j. The network itself is tensor-valued, with the four indices k, l, m and n, which correspond to the indices of the constituent tensors which were not contracted.

A key feature of tensor network contraction is that individual summations commute. That is, the sums in equation (1) may be done simultaneously, but we could also perform first the sum over i, producing the intermediate tensor

$$Q_{jkl} = \sum_i T_{ijkl} X_i, \tag{2}$$

and only then perform the sum over j to evaluate

$$N_{klmn} = \sum_j Q_{jkl} Y_{jmn}. \tag{3}$$

Likewise, we could first sum over j, producing

$$Q'_{iklmn} = \sum_j T_{ijkl} Y_{jmn}, \tag{4}$$

and then sum over i to obtain

$$N_{klmn} = \sum_i Q'_{iklmn} X_i. \tag{5}$$

Both pathways arrive at the same answer, but they may have very different computational costs. For instance, the intermediate Q' has a higher rank (number of indices) than the intermediate Q, and so if all bonds have the same dimension, Q' requires more memory to store and more computation time to evaluate. It is often convenient to write small tensor networks explicitly, as in equation (1), but for large ones this quickly becomes cumbersome. Instead we depict larger networks graphically using Penrose notation, with squares representing tensors and lines representing indices [25]. So, for instance, the network specified by the right-hand side of equation (1) is shown graphically in figure 1.

In this notation, performing a single sum amounts to combining two nodes in the graph into one. Hence, summing over j in equation (1) produces the network shown in figure 2. Then, summing over i finally yields the evaluated network shown in figure 3.

We call the order in which we contract pairs of indices a contraction sequence. To calculate the computational cost of a given contraction sequence, we count the number of floating-point multiplications that have to be performed [21]. This is equal to the number of floating-point additions, and so counts the number of operations required and the run-time up to a constant factor. Our cost function thus reads

$$\text{cost}(\{E\}) = \sum_{e \in \{E\}} \prod_{m \in \{v_e\}} \chi(m), \tag{6}$$

where $\{E\}$ denotes the ordered set of edges to be contracted, $\{v_e\}$ denotes the set of edges adjoining the two vertices connected by the edge e (including e itself) at a given contraction step, and $\chi(m)$ is the bond dimension of edge m, i.e. the number of different values that the index associated with m can assume. Holding coordination number fixed, the computational complexity of evaluating this cost function is $\mathcal{O}(E)$ where E is the number of edges in the network.

As an example, consider the contraction in equation (1). Contracting the index i first amounts to $\chi(i)\chi(j)\chi(k)\chi(l)$ elementary operations. Following up with the sum over the index j then adds another $\chi(j)\chi(k)\chi(l)\chi(m)\chi(n)$ operations. The total cost of this contraction ordering is therefore
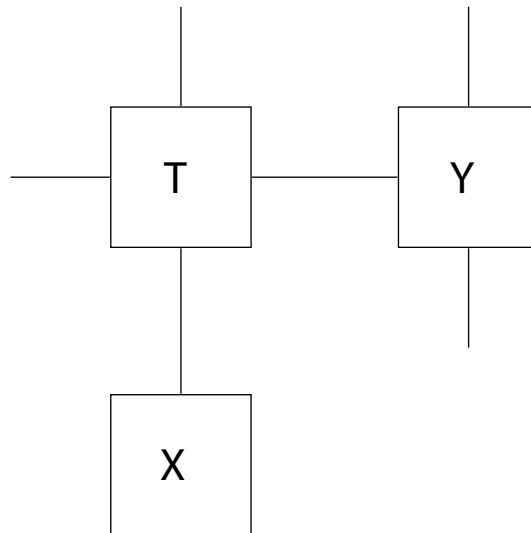
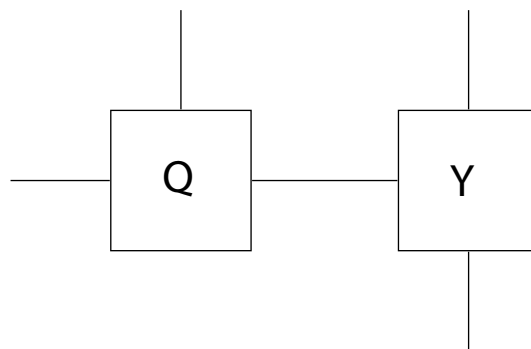**Figure 1.** The tensor network specified by equation (1) in Penrose notation.

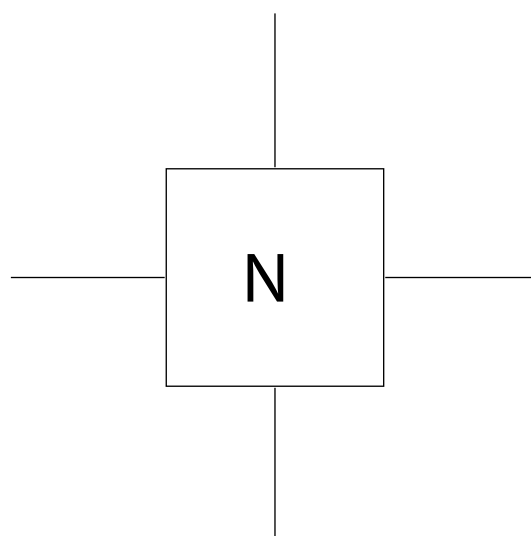**Figure 2.** The tensor network specified by equation (3) in Penrose notation.

Figure 3. The tensor N appearing in equation (1) in Penrose notation.

$$\text{cost}_{ij} = \chi(i)\chi(j)\chi(k)\chi(l) + \chi(j)\chi(k)\chi(l)\chi(m)\chi(n). \tag{7}$$

By contrast, the cost of contracting first j then i is

$$\text{cost}_{ji} = \chi(i)\chi(j)\chi(k)\chi(l)\chi(m)\chi(n) + \chi(i)\chi(k)\chi(l)\chi(m)\chi(n). \tag{8}$$

These are clearly different, and so there is a room to optimize by picking the lower-cost option.

3. Algorithms

We have tested four algorithms. Two of these, namely Exhaustive Search and Greedy Search, are in common use for obtaining tensor network contraction sequences. So far as we are aware, the other two have not previously been used to this end.

The other algorithms, namely Simulated Annealing and the Genetic Algorithm, share a common structure which will aid in their comparison. Each consists of a procedure for generating batches of contraction sequences. Only one batch is considered at a time. The cost of contracting the given tensor network with each sequence in the batch is then computed. If any sequence in the proposal has a lower cost than the previous lowest-cost sequence it is stored in place of the previous best sequence. The algorithm then proceeds to propose a new batch, possibly using the results of the previous batches. This process iterates until either all possible orderings have been considered or a time limit is reached. Where these methods differ is in the rule for producing new batches.

Because of this structure both methods may be run for as long as desired and can at any point return the best ordering found so far. We will take advantage of this to restrict each method to a fixed number of evaluations of the cost function. This limitation is a proxy for a runtime limit that is insensitive to details of the implementation of the algorithm or the underlying hardware, which makes it a useful means of comparison.

We now detail the four algorithms. Implementations for the Greedy Search, the Genetic Algorithm, and Simulated Annealing can be found at github.com/frankschindler/OptimizedTensorContraction.

3.1. Exhaustive search
Exhaustive Search comes in several varieties. In its most basic version every possible contraction ordering is considered exactly once. The cost of each is evaluated and the ordering with the lowest cost is returned.

This algorithm is deterministic and always returns the optimal contraction sequence. Because the number of contraction sequences to consider scales like $\mathcal{O}(e^E)$, where again E is the number of edges in the network, the run-time of this algorithm is exponential. More advanced variants of this algorithm incorporate tree pruning [21] to avoid considering sequences which can be proven to have higher cost than others, but in the worst case the cost is still exponential.

For the numerical results we present for the Exhaustive Search, we adapted the MATLAB version of the Netcon algorithm from Reference [21] to also output the accumulated number of cost function evaluations. We then ran it with the parameter choice costType $= 1$, muCap $= 1$, allowOPs $=$ false. We used the MATLAB version R2019a and Netcon version 2.01.

3.2. Greedy search
Greedy Search begins by considering the cost of performing just one step in the contraction. Evaluating this incremental cost takes time which is $\mathcal{O}(1)$. Each possible first step is considered, and the one with the lowest cost is taken. The method proceeds recursively, considering next all possible second steps.

Alternate variants of Greedy Search have been used which consider multiple steps simultaneously [20, 21]. For instance one could consider all possibilities for the next two steps, or more generally for the next k steps. The cost of this algorithm considering k steps simultaneously is $\mathcal{O}(E^k)$ incremental cost function evaluations, which is equivalent to $\mathcal{O}(E^{k-1})$ evaluations of the full cost function. Because the cost grows rapidly with k we only consider the commonly-used [20] case of $k = 2$ in the following.

For the numerical results we present for the Greedy Search and the remaining algorithms, we made use of Python 3.7.4, with the libraries *Numpy* 1.17.2, *Scipy* 1.3.1, *itertools*, and *copy*. We implemented the k-step Greedy Search as a standalone Python function built on these tools.

3.3. Genetic algorithm
The Genetic Algorithm begins by evaluating the fitness (the negative cost) of each contraction in a starting population of randomly generated sequences [15, 16]. It then samples a new population, drawing from the

starting population with replacement and with probabilities that are proportional to the fitness of the individual contraction sequences. This models the extinction of unfit specimen. Furthermore, the contractions in the new population are subject to mutation, in that there is a finite chance that the ordering of two randomly selected edges is exchanged in the respective sequence (the fittest individual of the population is always kept unchanged). This process is then iterated.

We implemented the Genetic Algorithm in Python. We chose a population size of 20 and a mutation rate of 60%. For the fitness function, we used

$$\text{fitness}(\{E\}) = \exp\left[\frac{\log\text{cost}(\{E_{\max}\}) - \log\text{cost}(\{E\})}{\log\text{cost}(\{E_{\max}\}) - \log\text{cost}(\{E_{\min}\})}\right] - 0.99, \tag{9}$$

where $\{E_{\min}\}$ and $\{E_{\max}\}$ are the contraction sequences with the lowest and highest cost in the population, respectively. This fitness function was chosen heuristically to return natural values in the range $(0, 10^{-1})$ while retaining the hierarchy of scales resolved by the original cost function (up to a power). Note that the subtraction of 0.99 matters because we generate probabilities from a population's fitness distribution after normalization. We checked that the performance of the Genetic Algorithm is not sensitive to the precise choice of fitness function.

3.4. Simulated annealing
Simulated Annealing works with an alternative representation of contraction sequences, where we encode permutations of edge labels by arrays of real numbers taken from the interval $[0, 1]$. A contraction can then be obtained from the permutation that orders the numbers in the respective array by magnitude. This representation has the advantage that it allows for a continuous deformation of the arrays while the constraint that each element represent a valid permutation is implicitly taken into account. This allows us to use the dual annealing [26] variant, which combines the standard classical annealing algorithm with a local optimizing search.

For the numerical results we present for Simulated Annealing, we employed the implementation of the dual_annealing algorithm that is available from the *optimize* package of the *Scipy* library. We used the default settings of the algorithm, which are local_search_options = {}, initial_temp = 5230.0, restart_temp_ratio = $2 * 10^{-5}$, visit = 2.62, accept = −5.0, seed = None, no_local_search = False, callback = None, x0 = None.

4. Numerical experiments

We perform our numerical experiments on two classes of tensor networks. The first are two-dimensional square lattices, shown in figure 4(a). We choose two-dimensional networks because in one-dimension the optimal contraction sequence is already known, so this provides one of the simplest non-trivial test cases.
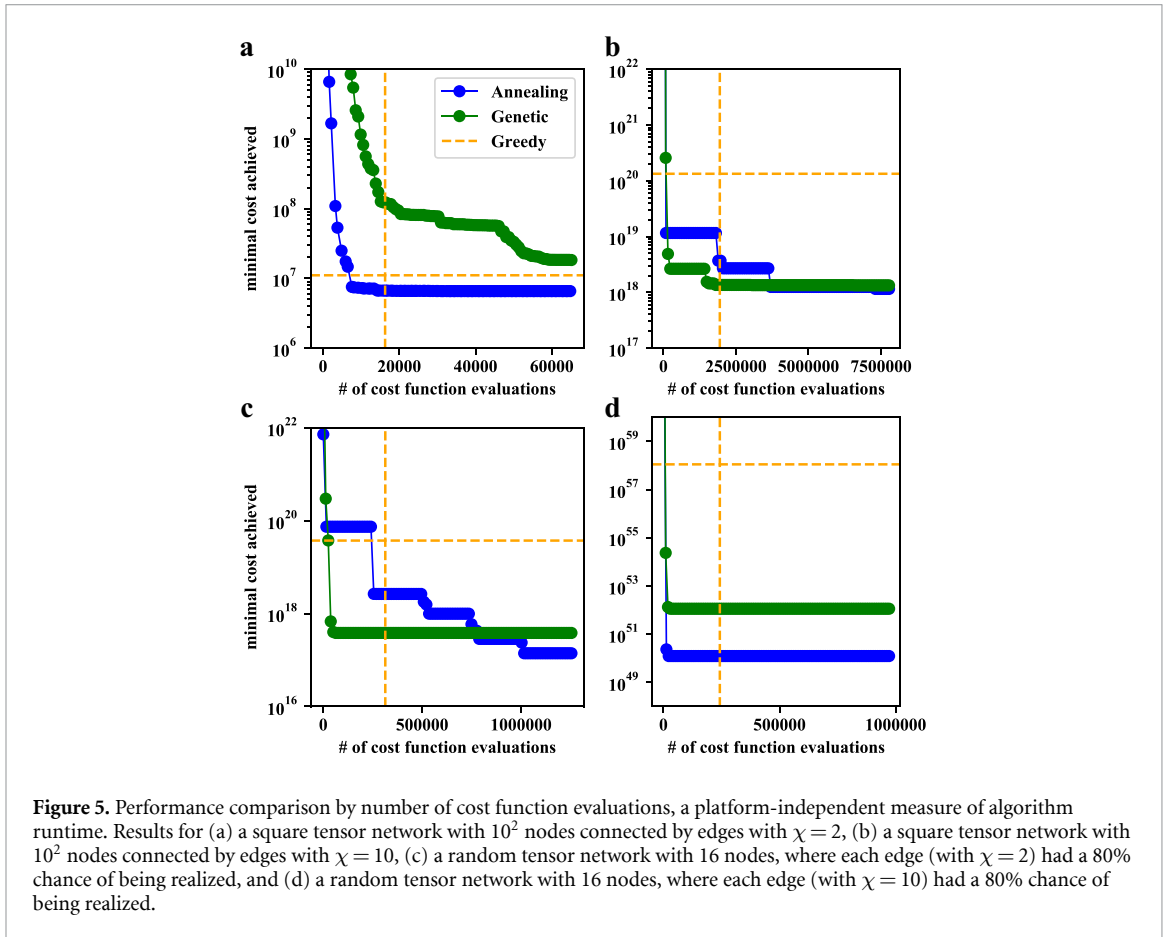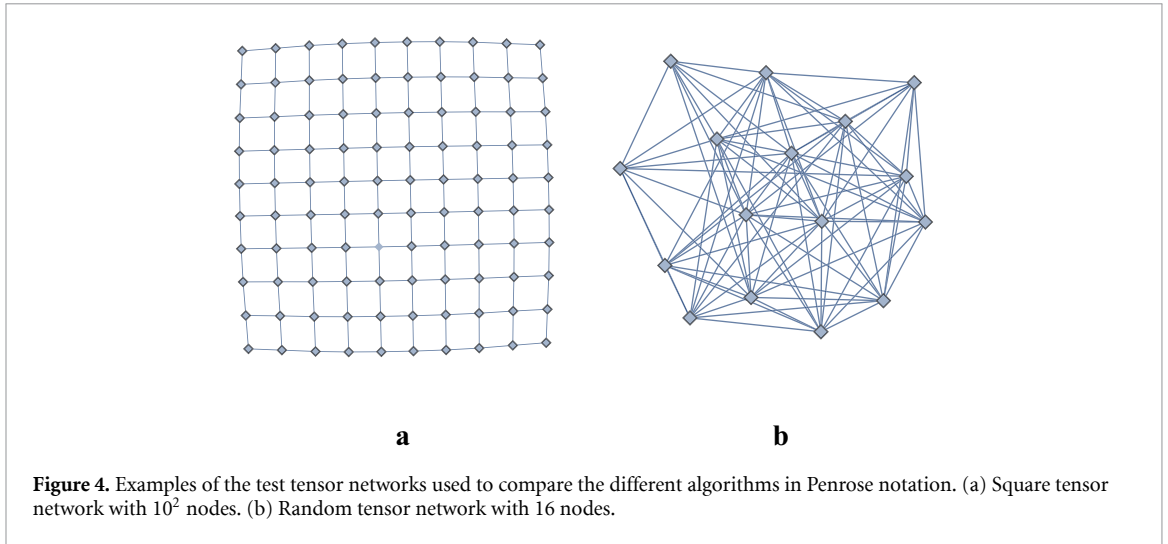
The second class of network we consider is that of Erdös-Rényi random graphs. These consist of a collection of nodes with edges distributed amongst them at random, such that all pairs of nodes have the same probability of having an edge, and such that edges are placed independently of one another. In our tests we let this probability be 80%. An example of a tensor network generated in this way is shown in figure 4(b). Such networks are analogous to spin glasses, and represent some of the most difficult tensor networks to contract due to their large connectivity and high variance in tensor rank.

4.1. Variable run-time
In our first experiment we consider the square network shown in figure 4(a), with a bond dimension of $\chi = 2$. We use each of Simulated Annealing, the Genetic Algorithm, and Greedy Search to produce contraction sequences for this network. The results are shown in figure 5(a). For Simulated Annealing and the Genetic Algorithm we show the contraction cost given by equation (6) of the best contraction sequence found as a function of the number of cost function evaluations used. The Greedy Search requires a fixed number of evaluations, and so we just show its output with that number of evaluations.

From this experiment we see that Simulated Annealing significantly outperforms the Genetic Algorithm when given the same number of function evaluations. This is not universally true, but we see the same in almost every case. We also see that the Greedy Search performs better than the Genetic Algorithm, but worse than Simulated Annealing at the same number of function evaluations.

Figure 4(b) shows the same experiment but with an increased bond dimension of $\chi = 10$. Increasing the bond dimension dramatically raises the contraction cost for each algorithm. The change in cost is of order $(\chi_{\text{new}}/\chi_{\text{old}})^{2L}$, where L is the linear size of the network. This may be seen by noting that each tensor at an intermediate stage represents a contiguous subset of the original network. Eventually those subsets come to

Figure 4. Examples of the test tensor networks used to compare the different algorithms in Penrose notation. (a) Square tensor network with 10^2 nodes. (b) Random tensor network with 16 nodes.

Figure 5. Performance comparison by number of cost function evaluations, a platform-independent measure of algorithm runtime. Results for (a) a square tensor network with 10^2 nodes connected by edges with $\chi = 2$, (b) a square tensor network with 10^2 nodes connected by edges with $\chi = 10$, (c) a random tensor network with 16 nodes, where each edge (with $\chi = 2$) had a 80% chance of being realized, and (d) a random tensor network with 16 nodes, where each edge (with $\chi = 10$) had a 80% chance of being realized.

be extensive in size and so come to have perimeter length of order L. Hence at some point each algorithm must contract two tensors with of order L bonds, with cost of order χ^{2L}.

Interestingly, with larger bond dimension Greedy Search performs worse relative to the other algorithms. We understand this as follows: for small bond dimensions it is possible for many contraction steps to matter in the total cost, because the difference between contractions of different ranks is small. As the bond dimension increases the cost of a contraction sequence comes to be dominated by the cost of the few most expensive contraction step(s). Optimizing a contraction sequence then becomes mostly a matter of avoiding the worst cases. Because the appearance of very expensive contraction steps is a function of the entire contraction sequence up to that point, this is a non-local optimization problem that Simulated Annealing and the Genetic Algorithm are better suited to.
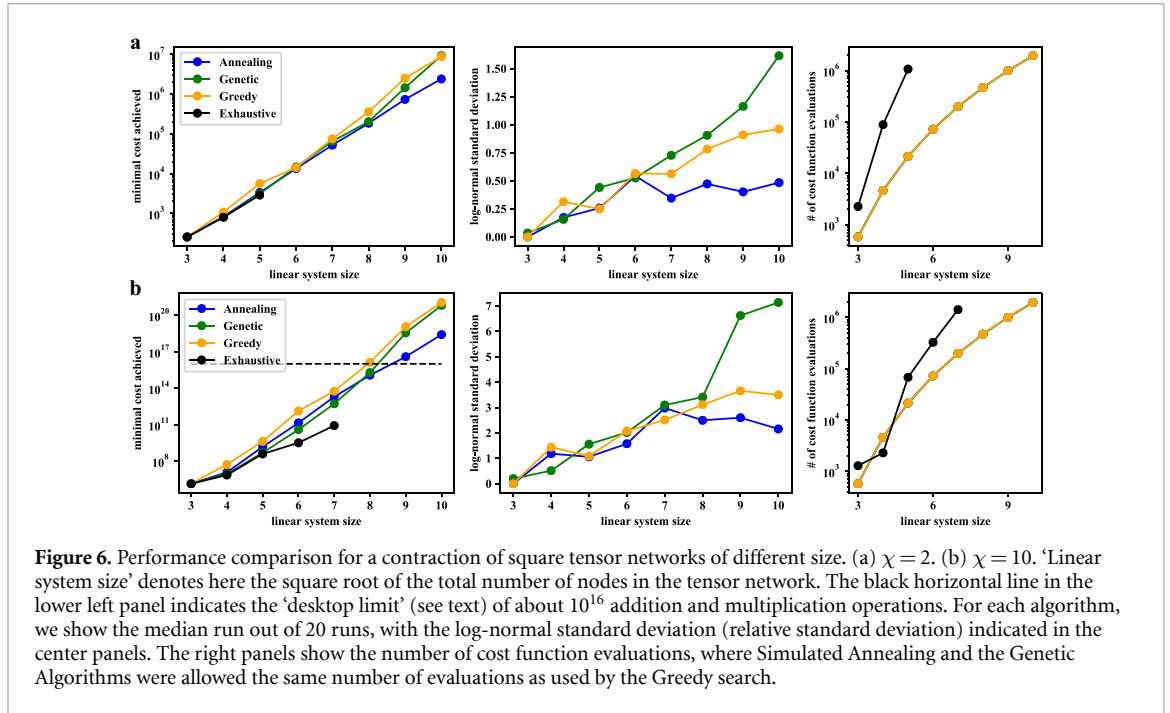
Figure 6. Performance comparison for a contraction of square tensor networks of different size. (a) $\chi = 2$. (b) $\chi = 10$. 'Linear system size' denotes here the square root of the total number of nodes in the tensor network. The black horizontal line in the lower left panel indicates the 'desktop limit' (see text) of about 10^{16} addition and multiplication operations. For each algorithm, we show the median run out of 20 runs, with the log-normal standard deviation (relative standard deviation) indicated in the center panels. The right panels show the number of cost function evaluations, where Simulated Annealing and the Genetic Algorithms were allowed the same number of evaluations as used by the Greedy search.

We next repeat these experiments for the Erdös-Rényi random graph shown in figure 4(b). The results are shown for bond dimensions $\chi = 2$ and $\chi = 10$ in figure 5(c), (d) respectively. Both Simulated Annealing and the Genetic Algorithm significantly outperform Greedy Search with a similar number of cost function evaluations. Moreover, they do so even with significantly fewer cost function evaluations. Intuitively, these non-local optimization methods are able to perform comparatively better with higher connectivity and less local structure.

Common to all of the panels of figure 5, we see that Simulated Annealing and the Genetic Algorithm improve in bursts, separated by long plateaus. This makes it difficult to arrive at strong statements about the correct number of function evaluations to use with these algorithms, as there is no clear indication of whether the search will continue improving or not.

The large, order-of-magnitude nature of the bursts, however, suggests a heuristic to use in practice, which is that the search for better contraction sequences should be conducted for a time comparable to the run-time of the current best contraction sequence. We call this the 'time-remaining' heuristic. In that way the search at most doubles the run-time if it yields no improvement, while still offering a chance of dramatic gains.
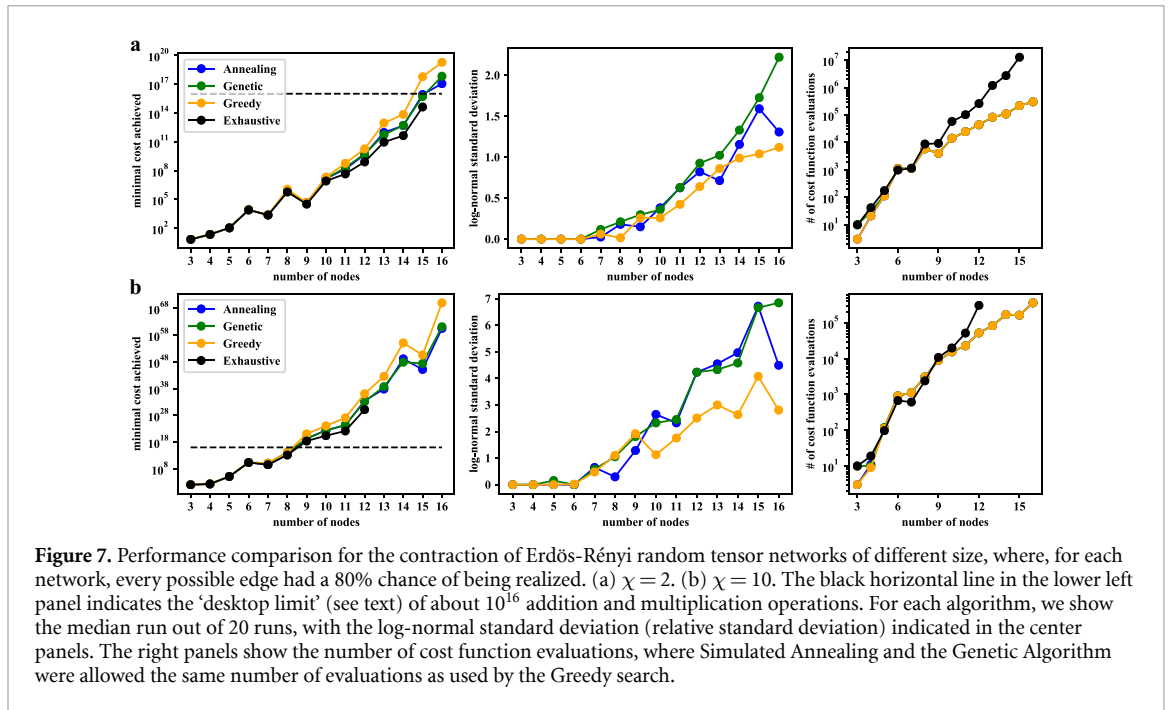
4.2. Equal run-time

For comparison purposes we do not adopt the time-remaining heuristic here. Rather we now fix the number of cost function evaluations used by each algorithm to be equal to that of the Greedy Search. This enables meaningful comparisons between algorithms with similar runtime contraints.

Figure 6(a) shows the performance of Simulated Annealing, the Genetic Algorithm and Greedy Search on two-dimensional square tensor networks of varying sizes with bond dimension $\chi = 2$. The left-most panel shows the median performance across 20 runs of each non-deterministic algorithm, along with the global optimal result provided by Exhaustive Search. The middle-panel shows the relative standard deviation in performance for the same. Finally, the right-most panel shows the number of cost function evaluations used by the Exhaustive Search and Greedy Search. Simulated Annealing and the Genetic Algorithm were both allowed the same number of evaluations as Greedy Search.

We see relatively little variation in performance across these four algorithms, and to the point where we were able to use the Exhaustive Search the algorithms perform close to the global optimum. To the extent that there is a difference, it is for larger systems, for which Simulated Annealing significantly outperforms the other algorithms.

Interestingly, the relative standard deviation in cost is much larger with both Greedy Search and the Genetic Algorithm. We are not sure why the Genetic Algorithm has a high relative standard deviation. The high relative standard deviation of the Greedy Search is understandable, however: many choices are degenerate for the Greedy Search. Often a tensor network has many different contractions with the same immediate cost, and the same is true at higher search depths. Even though the immediate cost is degenerate,

Figure 7. Performance comparison for the contraction of Erdös-Rényi random tensor networks of different size, where, for each network, every possible edge had a 80% chance of being realized. (a) $\chi = 2$. (b) $\chi = 10$. The black horizontal line in the lower left panel indicates the 'desktop limit' (see text) of about 10^{16} addition and multiplication operations. For each algorithm, we show the median run out of 20 runs, with the log-normal standard deviation (relative standard deviation) indicated in the center panels. The right panels show the number of cost function evaluations, where Simulated Annealing and the Genetic Algorithm were allowed the same number of evaluations as used by the Greedy search.

the long-term consequences of these choices on the network may be radically different, causing significant variance in total cost.

Figure 6(b) shows the same experiment but with bond dimension $\chi = 10$. We now see a larger spread between the algorithms. Here we have highlighted the so-called 'desktop limit' of cost $= 10^{16}$, which provides a rough bound on the cost of contractions that can reasonably be performed on a modern desktop computer limited to a day of runtime. (Note that if the cost is dominated by a single expensive contraction step the practical limit is somewhat lower, as tensors with 10^{16} elements are unlikely to fit into memory.) In particular, the gap between Greedy Search and Simulated Annealing is such that the former hits the desktop limit on systems roughly 10% smaller than the latter, suggesting that with the improvements offered by Simulated Annealing it should be possible to contract larger tensor networks than were previously possible.
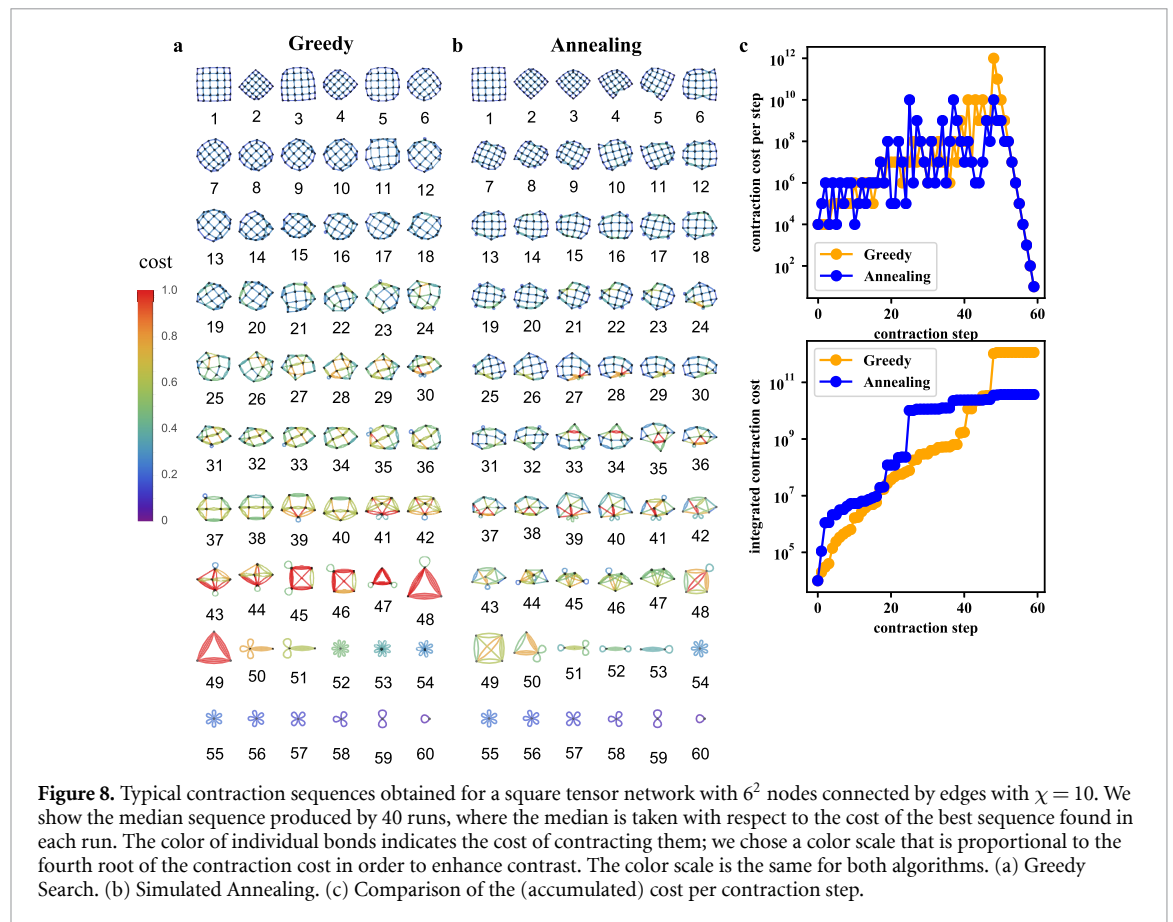
The larger bond dimension also brings about an increased relative standard deviation in their performance. The increased relative standard deviation comes about because the cost is now more sensitive to the few most expensive contraction steps, and so becomes more sensitive to the (discrete) ranks of tensors as the contraction proceeds.

We next repeated these experiments with Erdös-Rényi random graphs of varying size. The results are shown in figure 7. For small systems the algorithms all find nearly-optimal contraction sequences. As the system size increases above $10 - 11$ nodes a large difference emerges which grows until the Greedy Search performs a factor of $10 - 100$ worse than Simulated Annealing, which in turn performs a factor of 10 or so worse than optimal. The relative standard deviation in performance across runs is generally larger than in the square network cases, particularly for Simulated Annealing. The overall increase can be understood as being due to increased variance in tensor ranks making the cost more sensitive to the precise contraction sequence. We are not sure why this affects Simulated Annealing more than the other algorithms, though it may indicate that with random graphs the dual annealing implementation is less able to exploit the structure of the network in its local search steps.

5. Contraction sequences

To understand how Simulated Annealing comes to outperform the Greedy Search it is useful to examine a typical contraction sequence produced by each algorithm. In figure 8 we show the median best contraction sequence of each algorithm taken across 40 runs for a square network with linear size $L = 6$ and bond dimension $\chi = 10$. We depict the contraction sequence by showing the tensor graph before each contraction step. The colors of individual bonds are proportional to the fourth root of the contraction cost, with red indicating higher cost and blue indicating lower cost.

The Simulated Annealing sequence is roughly 100 times less expensive than that of the Greedy Search. As expected for this large bond dimension, both algorithms have costs which are dominated by three or fewer

Figure 8. Typical contraction sequences obtained for a square tensor network with 6^2 nodes connected by edges with $\chi = 10$. We show the median sequence produced by 40 runs, where the median is taken with respect to the cost of the best sequence found in each run. The color of individual bonds indicates the cost of contracting them; we chose a color scale that is proportional to the fourth root of the contraction cost in order to enhance contrast. The color scale is the same for both algorithms. (a) Greedy Search. (b) Simulated Annealing. (c) Comparison of the (accumulated) cost per contraction step.
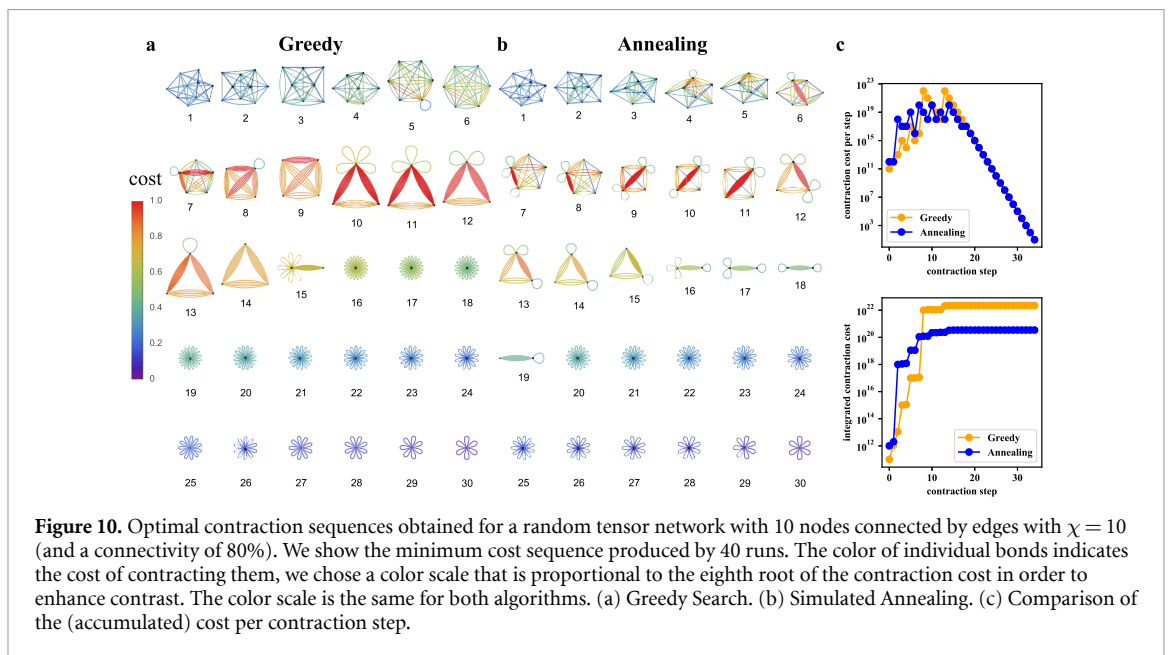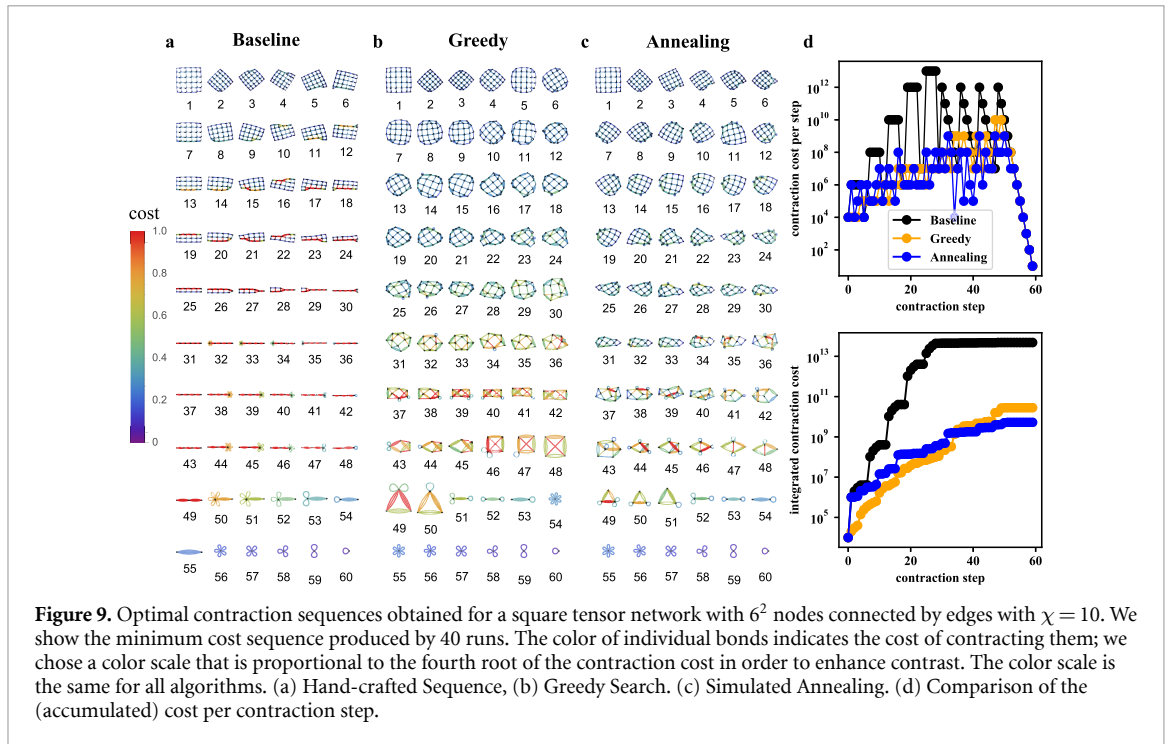
steps (figure 8(c)). From figure 8(a)-(b) we see that Simulated Annealing always leaves itself a comparatively low-cost option, while Greedy Search exhausts all such options and is forced into costly contraction steps.

Arranging to retain lower-cost options is inherently a global optimization process, because the cost of contracting an edge is strongly dependent on the stage at which it is contracted and on the contraction order leading up to that point. This explains why Simulated Annealing is able to achieve this task while Greedy Search is not: the early costlier contractions that Simulated Annealing performs act to reduce the cost of the most expensive steps towards the end.

A hint is provided by the Simulated Annealing sequence between steps 26 and 29, and again between steps 32 and 43. In both cases, one or more bonds emerge which involve expensive contractions. In the first instance Simulated Annealing performs the expensive contraction, which enables lower-cost options afterwards. In the second instance it merges nearby nodes into those adjacent to the expensive bond (for instance, 35 - 36). In doing so it produces self-loops on the adjacent nodes which, upon elimination, reduce the cost of the deferred expensive step.
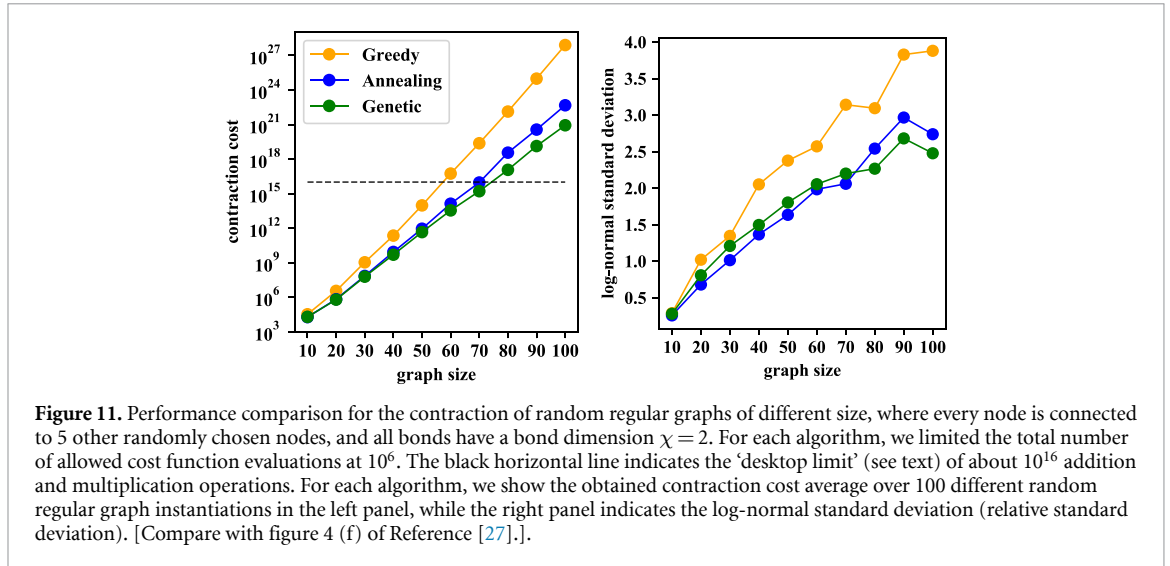
We next turn to the best contraction sequence produced by each algorithm. In figure 9(a)–(c) we show the best contraction sequence of each algorithm taken across 40 runs for the same network as in figure 8. We also show for comparison a typical hand-crafted contraction sequence similar to the corner transfer matrix method commonly used with PEPS. In this sequence rows are contracted together repeatedly until just one remains, at which point that row is contracted down to a point. We see that both Greedy Search and Simulated Annealing significantly outperform the hand-crafted sequence. Both algorithms manage this by producing fewer high-rank tensors, which is enabled by 'contracting inwards' from the perimeter rather than working with a single edge the whole time. Doing so reduces the number of extra bonds accumulated by each tensor on the edge, which holds the rank down.

In figure 9(d) we see that, like in the median case, there are just a few steps which together dominate the cost of the best contraction sequence for each algorithm. However, unlike in the median case, in the best case the costs of the different contraction sequences are of the same order of magnitude, which is a factor of 10 or so better than the median case for the Simulated Annealing algorithm. This suggests that the best case for the Greedy Search is a bigger improvement over the median case than the best case for the Simulated Annealing algorithm is over its median case.

Figure 9. Optimal contraction sequences obtained for a square tensor network with 6^2 nodes connected by edges with $\chi = 10$. We show the minimum cost sequence produced by 40 runs. The color of individual bonds indicates the cost of contracting them; we chose a color scale that is proportional to the fourth root of the contraction cost in order to enhance contrast. The color scale is the same for all algorithms. (a) Hand-crafted Sequence, (b) Greedy Search. (c) Simulated Annealing. (d) Comparison of the (accumulated) cost per contraction step.

Figure 10. Optimal contraction sequences obtained for a random tensor network with 10 nodes connected by edges with $\chi = 10$ (and a connectivity of 80%). We show the minimum cost sequence produced by 40 runs. The color of individual bonds indicates the cost of contracting them, we chose a color scale that is proportional to the eighth root of the contraction cost in order to enhance contrast. The color scale is the same for both algorithms. (a) Greedy Search. (b) Simulated Annealing. (c) Comparison of the (accumulated) cost per contraction step.

We can understand this improvement by noting that in the early stages of the Greedy contraction sequence there is significant degeneracy between the various least-expensive contractions. This results in a wide variety of different possible states following the first 10 or so contraction steps. The typical such state is evidently much harder to continue contracting than the best such state. This conclusion highlights the importance of optimizing globally over the whole contraction sequence, and not just locally as Greedy Search does.

Finally, in figure 10 we show the best contraction sequence of Greedy Search and Simulated Annealing across 40 runs for an Erdös-Rényi random graph. We see again that Simulated Annealing does a much better job of preserving comparatively good options throughout the contraction, while Greedy Search exhausts its cheap contraction options early on and is forced into a run of very expensive contraction steps. Unfortunately the random structure of this graph exacerbates our earlier challenge interpreting the Simulated Annealing contraction sequence, and it is not clear exactly what choices it is making that enable such good long-run performance. Nevertheless, the performance is remarkable: Simulated Annealing finds a

Figure 11. Performance comparison for the contraction of random regular graphs of different size, where every node is connected to 5 other randomly chosen nodes, and all bonds have a bond dimension $\chi = 2$. For each algorithm, we limited the total number of allowed cost function evaluations at 10^6. The black horizontal line indicates the 'desktop limit' (see text) of about 10^{16} addition and multiplication operations. For each algorithm, we show the obtained contraction cost average over 100 different random regular graph instantiations in the left panel, while the right panel indicates the log-normal standard deviation (relative standard deviation). [Compare with figure 4 (f) of Reference [27].].

contraction sequence that is 100 times faster than that of Greedy Search, and does so with the same number of cost function evaluations.

6. Random regular networks

Up to this point, we have focused on applying our algorithms to either sparse tensor networks that have significant spatial structure [the two-dimensional square networks, figure 4(a)], or dense tensor networks that have no spatial structure [the Erdös-Rényi random graphs, figure 4(b)]. We now consider random regular graphs as intermediary benchmark networks that allow us to compare our methods directly with the very recently introduced 'hyper-optimized' contraction algorithms of Gray and Kourtis, Reference [27]. These represent a collection of algorithms that rely on, among other techniques, tree decompositions of the line graph corresponding to a given tensor network, detection of dense node communities in the network, and top-down graph partitioning. They are thereby significantly more sophisticated and domain-specific than the combinatorial optimization algorithms we consider. In addition, the authors of Reference [27] employ a hyperparameter optimization scheme that aims to find the best numerical values of the free parameters entering the individual algorithms *for each tensor network separately*. By contrast, for simplicity and computational efficiency in our algorithms we use the same fixed parameters (listed in section 3) for all tensor networks. This comes at the expense of not being able to adapt optimization strategies, and even the choice of algorithm itself, to the specific type of tensor network under consideration, and we believe a promising path forward is to employ hyperparameter optimization with the combinatorial optimization algorithms we have discussed here.

In random k-regular graphs of size L, each node is connected to k other randomly chosen nodes. There are therefore a total of $kL/2$ bonds. This linear-in-L scaling of the number of bonds allows us to reach far larger system sizes than for Erdös-Rényi random graphs (for which the scaling of bond number is quadratic in system size). To compare with Reference [27], we sample a total of 100 5-regular graphs for each system size, ranging up to $L = 100$, and average the contraction costs of the best solutions obtained by the Greedy, Simulated Annealing, and Genetic algorithms. The results are shown in figure 11. Interestingly, we find that the Genetic Algorithm consistently outperforms Simulated Annealing in finding good contraction paths for random regular graphs, while the situation was reversed in the case of large square tensor networks (figure 6). This finding highlights that the choice of algorithm should be made on a problem dependent basis.

Comparing with the results of Reference [27], we find that the performance of Simulated Annealing and Genetic algorithms is similar to that of the QuickBB (line-graph tree decomposition) and BGreedy (Boltzmann-sampled Greedy paths) algorithms described in the reference, with the Genetic Algorithm achieving slightly better results than both. However, the GN (community detection) and KaHyPar (graph partitioning) algorithms of Reference [27] outperform our methods on average in finding efficient contraction paths for random regular graphs. This finding suggests that it might be advantageous to replace the BGreedy subroutine entering the KaHyPar algorithm by our Genetic Algorithm for even better performance. Because we have seen that algorithm performance heavily depends on the kind of tensor network considered, we also believe it will be useful to incorporate the Simulated Annealing and Genetic

algorithms themselves into future hyperparameter-optimized tensor network contraction frameworks, especially because they are conceptually simple and straightforward to implement.

The advantage our algorithms have over other state-of-the-art techniques for tensor network contraction is that: (1) They impose no assumptions on the structure of the tensor network. This suggests that they are good baseline algorithms against which other, more sophisticated methods should be compared. This role has been traditionally occupied by the Greedy algorithm which is, however, more specialized than our approaches, since it assumes that the global cost function decomposes into local factors across the contraction history. (2) They impose no assumptions on the format of the input data, depending only implicitly on the contraction cost of a given tensor ordering [equation (6)]. Hence they can run on any faithful ordering representation. In the present work, we have chosen to represent a contraction ordering by its permutation vector (Genetic Algorithm) or by a vector of real numbers (Simulated Annealing). However, there are many more choices of input data, such as inversion tables [28] or graphical representations. Input-ignorant algorithms such as these allow for a clean separation of whether performance gains are due to data pre-processing, or due to the method of optimization itself. (3) They are subject to proven convergence bounds. This provides quantitative guarantees on the convergence of both annealing [29] and genetic [30] algorithms to the desired optimal contraction path, which are at this point absent for the hyper-optimized algorithms discussed above. These advantages are of a structural nature, independent of the performance of the algorithms.

7. Conclusions

We have optimized tensor network contraction sequences using four different algorithms, namely Exhaustive Search, Greedy Search, Simulated Annealing, and a Genetic Algorithm. The first two of these are commonly used in contracting tensor networks, while to our knowledge the latter two have not been used in this domain. We find that Simulated Annealing significantly outperforms Greedy Search, both in the best case and on average. In many cases the cost of the contraction sequence found by Simulated Annealing is orders of magnitude lower than that of Greedy Search with a comparable amount of search time. This advantage grows larger with network size, and is most notable on networks with structure such as the square lattice.

With additional search time Simulated Annealing performs even better, often by a large enough margin to justify the extra time spent optimizing the contraction sequence. This suggests a potential strategy to use in practice, our 'time-remaining' heuristic, in which one optimizes the contraction sequence until the time spent optimizing is comparable to the cost of the current best known contraction sequence.

Unfortunately we have been unable to extract any further intuition from these contraction sequences. They do not appear to lead to design principles we may use to craft custom sequences for particular classes of networks. In practice, however, this may not matter: algorithmically-generated contraction sequences may well suffice, particularly if they are more efficient than hand-crafted or heuristically-guided ones.

The contraction sequence optimization methods we have discussed should prove practically useful in all fields where tensor networks provide good variational models, such as condensed matter and statistical physics (where tensor networks are realistic ground states of gapped local Hamiltonians), high energy theory (where large and unstructured tensor networks of the kind considered by us are explicit realizations of the holographic duality) [31], computational quantum chemistry [32], and even machine learning [6]. Whenever a variational model does not have some special structure (such as being one-dimensional or representable by a tree graph), the computations required for its exact contraction will scale exponentially with size, making algorithms that find efficient contraction paths highly desirable because the human time spent on optimizing contraction sequences can be substantial.

While it is true that the exact contraction of larger tensor networks than previously possible is an interesting application in itself, for instance to compare classical computing performance with that of near-term quantum computers [27, 33], the exponential scaling that applies even to optimal contraction paths fundamentally limits the scope of full exact contractions. Thus we expect our methods to be most useful as part of approximate contraction methods such as Tensor Network Renormalization, which often rely on repeatedly and exactly contracting moderate-sized networks to produce inputs into the approximation scheme [34]. These and other methods built on top of exact tensor network contraction require fast and reliable contraction machinery, and we believe that this is where the chief advantages of optimized contraction sequences lie.

Acknowledgments

Data Availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID iD

Frank Schindler ⬤ https://orcid.org/0000-0001-8113-0998

References

[1] White S R 1992 *Phys. Rev. Lett.* **69** 2863
[2] Orús R 2014 *Ann. Phys. NY* **349** 117
[3] Nakatani N and Chan G K-L 2013 *J. Chem. Phys.* **138** 134113
[4] Xu J, Liang L, Deng L, Wen C, Xie Y and Li G 2019 *Phys. Rev.* E **100** 043309
[5] Stoudenmire E and White S R 2012 *Ann. Rev. Cond. Matter Phys.* **3** 111
[6] Stoudenmire E and Schwab D J 2016 *Advances in Neural Information Processing Systems 29* Lee D D, Sugiyama M, Luxburg U V, Guyon I and Garnett R (Red Hook, NY: Curran Associates, Inc.) pp 4799–807
[7] Liu D, Ran S-J, Wittek P, Peng C, García R B, Su G and Lewenstein M 2019 *New J. Phys.* **21** 073059
[8] Chi-Chung L, Sadayappan P and Wenger R 1997 *Parallel Process. Lett.* **07** 157
[9] Evenbly G and Pfeifer R N C 2014 *Phys. Rev.* B **89** 245118
[10] Jermyn A S 2020 *SciPost Phys.* **8** 5
[11] Ran S-J, Tirrito E, Peng C, Chen X, Tagliacozzo L, Su G and Lewenstein M 2020 *Tensor Network Contractions* (Berlin: Springer) http://dx.doi.org/10.1007/978-3-030-34489-4
[12] Pan F, Zhou P, Li S, and Zhang P 2019 arXiv:1912.03014 [physics.comp-ph]
[13] Hauru M, Delcamp C and Mizera S 2018 *Phys. Rev.* B **97** 045111
[14] Morita S, Igarashi R, Zhao H-H and Kawashima N 2018 *Phys. Rev.* E **97** 033310
[15] Sadeghi J, Sadeghi S and Niaki S T A 2014 *Inf. Sci.* **272** 126
[16] Mitchell M 1996 *An Introduction to Genetic Algorithms* (Cambridge, MA: MIT Press)
[17] Jakes-Schauer J, Anekstein D, and Wocjan P 2019 arXiv:1908.11034 [cs.DM]
[18] Khachaturyan A, Semenovsovskaya S and Vainshtein B 1981 *Acta Crystallographica* A **37** 742
[19] Bollweg W, Maurer H and Kroll H 1997 Numerical prediction of crystal structures by simulated annealing *Developments in Global Optimization* eds Bomze I M, Csendes T, Horst R and Pardalos P M (Boston, MA: Springer) pp 253–88
[20] Smith D G A and Gray J 2018 *J. Open Source Software* **3** 753
[21] Pfeifer R N C, Haegeman J and Verstraete F 2014 *Phys. Rev.* E **90** 033315
[22] Pfeifer R N C, Evenbly G, Singh S, and Vidal G 2014 arXiv:1402.0939 [physics.comp-ph]
[23] Fried E S, Sawaya N P D, Cao Y, Kivlichan I D, Romero J and Aspuru-Guzik A 2018 *PLOS ONE* **13** 1
[24] Dumitrescu E F, Fisher A L, Goodrich T D, Humble T S, Sullivan B D and Wright A L 2018 *PLOS ONE* **13** 1
[25] Penrose R 1971 Combinatorial Mathematics and its Applications (California, Academic Press) 224–41
[26] Xiang Y, Sun D, Fan W and Gong X 1997 *Phys. Lett.* **233** 216
[27] Gray J and Kourtis S 2020 arXiv:2002.01935 [quant-ph]
[28] Knuth D E 1973 *The Art of Computer Programming, Volume 3: Sorting and Searching* (Reading, MA: Addison-Westley Publishing)
[29] Mitra D, Romeo F and Sangiovanni-Vincentelli A 1986 *Adv. Appl. Probab.* **18** 747–71
[30] Eiben A E, Aarts E H L and Van Hee K M 1991 *Parallel Problem Solving From Nature* ed Schwefel H-P and Männer R (Berlin, Heidelberg: Springer Berlin Heidelberg) pp 3–12
[31] Hayden P, Nezami S, Qi X-L, Thomas N, Walter M and Yang Z 2016 *J. High Energy Phys.* **2016** 9
[32] Hirata S 2003 *J. Phys. Chem.* A **107** 9887
[33] Zhou Y, Miles Stoudenmire E, and Waintal X 2020 arXiv:2002.07730 [quant-ph]
[34] Evenbly G and Vidal G 2015 *Phys. Rev. Lett.* **115** 180405