Master's Degree in Physics of Complex Systems (Fisica dei sistemi complessi) LM-44 (DM270)

Graduation Session: October 2024
Academic Year 2023/2024

New tensor network contraction techniques and algorithms for the evaluation of marginals in probabilistic graphical models

Thesis Supervisors:
Supervisor: Alfredo Braunstein
Co-supervisor: Stefano Crotti

Candidate:
Matteo Simeone

Turin, October 2024

Contents

1 Introduction

Evaluating properties of a probabilistic model, such as means, variances and correlations from probability distributions defined on factor graphs [1](FGs), is inherently challenging due to its *complexity*, here defined as the number of operations and memory allocations required, which often scale exponentially in the number of variables involved.
Tensor networks[2] (TNs) offer a novel approach to handle probability distributions defined on FGs, enabling the use of optimization techniques and heuristic strategies to significantly reduce the complexity.

Here I present

1. A deep analysis in the TN frameworks, with different maps allowing to pass from probabilistic models to specific TN's peculiar structures. It is also presented a method to transform a generic N-tensors TN in a square lattice TN which has at most $N(N/2+1)$ tensors.

2. Two TN-contraction algorithms I've written in Julia programming language[a] from scratch, namely MCE algorithm and RRC algorithm, respectively allowing the contraction of generic TNs and square-lattice TNs. In addition, simulations results are provided in dedicated chapters. These two algorithms maps the calculation of a property of interest of a probabilistic model defined on a FG into a TN contraction problem, where the balance between the precision on the result and the complexity of the contraction is tunable through 3 input parameters chosen by the user.

3. Two examples of applications: the Ising model on different graphs with different model's parameters and a Boltzmann learning based biophysics problem, namely the inference of native contacts in proteins families.

[a] https://julialang.org

[1] Factor graphs on Wikipedia
[2] https://tensornetwork.org

2 Definition of a TN

A TN consists of an undirected graph $G = (V, E)$, along with a positive integer-valued map

$$d(\cdot) : \eta \to \mathbb{N}_{>0}$$

assigning each edge $\eta \in E$ to a vector space F of dimension d_η, and a map

$$A(\cdot) : v \to \bigotimes_{\eta=(i,v)\in E | i\in V} F_{d_\eta}$$

assigning each node $v \in V$ to a tensor $A(v)$ whose shape is determined by the dimensions assigned to edges in E connecting v.

2.a A simple family of tensors: Copy Tensors

Given an orthonormal basis $B = \{e_1, \ldots, e_d\}$ for a vector space F_d, for each $n \geq 1$ we define the n-th order copy tensor associated with B to be

$$\Delta_n \doteq \sum_{x=1}^{d} (e_x)^{\otimes n}$$

The copy tensors ensures that all the variables (also called *indices*) connected to it take the same value. In figure 1 it is depicted the graphical representation of the first 4 copy tensors (purple nodes) with their indices (black links)

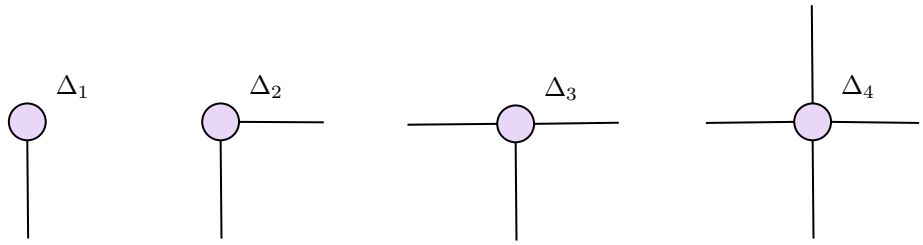

Figure 1: The first 4 copy tensors $\Delta_1, \Delta_2, \Delta_3, \Delta_4$.

Another interesting property concerning copy tensors is that connected networks of copy tensors can be arbitrarily reorganised, as depicted in Figure 2.

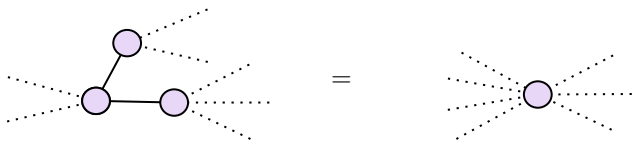

Figure 2: Connected networks of copy tensors can be arbitrarily reorganised in a new connected network, preserving the indices connecting the Δ's with the rest of the network.

2.b Construction of a TN

Starting from a FG, the following steps allows to obtain its TN representation:

1. Substitute each node representing a variable with a copy tensor of order equal to the degree of that node. The order of a tensor is the dimension of its domain.

2. Substitute each factor with a tensor encoding all the possible values that factor can assume [3].

3. Connect copy tensors of point 1 and tensors of point 2 according to the connectivity of the original FG. The edges, i.e. the indices, are the variables of the original network and the dimension of an index is the number of values that variable can assume.

In figure 3 it is depicted an example of transformation of a FG, representing a probability distribution P factorized as

$$P(X, Y, W, V) = \frac{1}{Z}\phi_X(X, W)\phi_Y(Y, W)\phi_V(V, W)$$

into a TN, where Z is the normalization term.

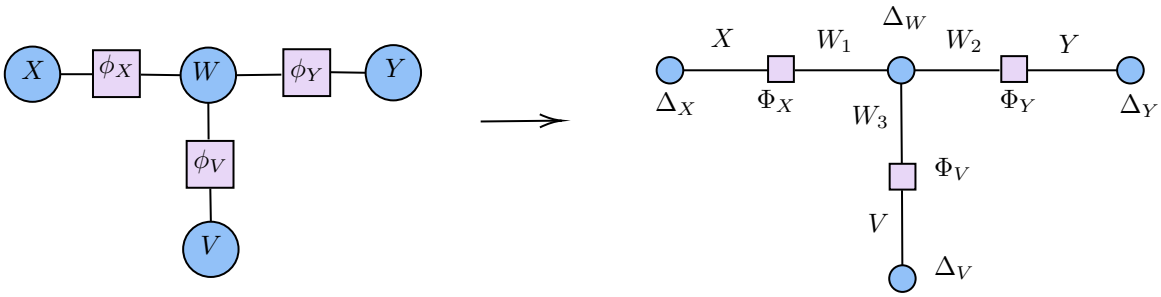

Figure 3: **Left**: the FG of interest. **Right**: The Δ's are copy tensors and purple squares are the factor's tensors. Note the role of copy tensors: Δ^W ensures that the edge-variables W_1, W_2, W_3 take the same values, i.e. they represent the same variable in the original FG.

Contracting a TN means summing over all the values of the indices.
In the example in figure 3:

$$Z = \sum_{X,W_1,W_2,W_3,Y,V} \Delta^X_X \Phi^X_{X,W_1} \Delta^W_{W_1,W_2,W_3} \Phi^V_{W_3,V} \Phi^Y_{W_2,Y} \Delta^Y_Y \Delta^V_V$$

where e.g. $\Delta^W_{W_1,W_2,W_3}$ means the tensor entry W_1, W_2, W_3 of Δ^W.

[3] E.g. if the factor connects two binary variables, then the associated tensor will be a 2 by 2 matrix, encoding the 4 values that factor can assume.

2.c Matrix Product State (MPS)

Consider an N-order tensor ϕ with indices $s_1, \ldots, s_N$. The MPS representation of ϕ is its exact reformulation in terms of a product of new smaller tensors organized on a 1D chain [6].
Let's start expressing ϕ as a product of a first tensor encoding information on the initial $N-1$ spins, and a second one containing information on s_N.

$$\phi_{s_1,\ldots,s_N} = \sum_{a_{N-1}} \phi^{[N-1]}_{s_1,\ldots,s_{N-1};a_{N-1}} A^{[N]}_{s_N;a_{N-1}}$$

Note that as a convention, we always put the physical indices in front of virtual indices (a's are called so) and use a semicolon to separate them. For the tensor $\phi^{[N-1]}$ one can do the same decomposition by grouping the first $N-2$ indices and decompose again as:

$$\phi^{[N-1]}_{s_1,\ldots,s_{N-1};a_{N-1}} = \sum_{a_{N-2}} \phi^{[N-2]}_{s_1,\ldots,s_{N-2};a_{N-2}} A^{[N-1]}_{s_{N-1};a_{N-2},a_{N-1}}$$

Repeat decomposing in the above mentioned way, until each tensor contains only one physical index. Eventually, we obtain the MPS representation of the tensor ϕ:

$$\phi_{s_1,\ldots,s_N} = \sum_{a_1,\ldots,a_{N-1}} A^{[1]}_{s_1;a_1} A^{[2]}_{s_2;a_1,a_2} \cdots A^{[N-1]}_{s_{N-1};a_{N-2},a_{N-1}} A^{[N]}_{s_N;a_{N-1}}$$

The above procedure shows that any tensor can be written exactly in its MPS form, as long as the dimensions of the virtual indices are not limited.
In doing the MPS representation of a tensor using the function ITensors.MPS() in Julia programming language, one input parameter is considered in the algorithms I've developed:

1. $\boldsymbol{\chi}$, controlling the maximum dimension for the indices $a_1, a_2 \ldots, a_{N-1}$. The greater χ the greater the precision on the MPS representation of a tensor.

In figure 4 it is depicted the detailed graphical representation of the process of obtaining the MPS representation of ϕ.

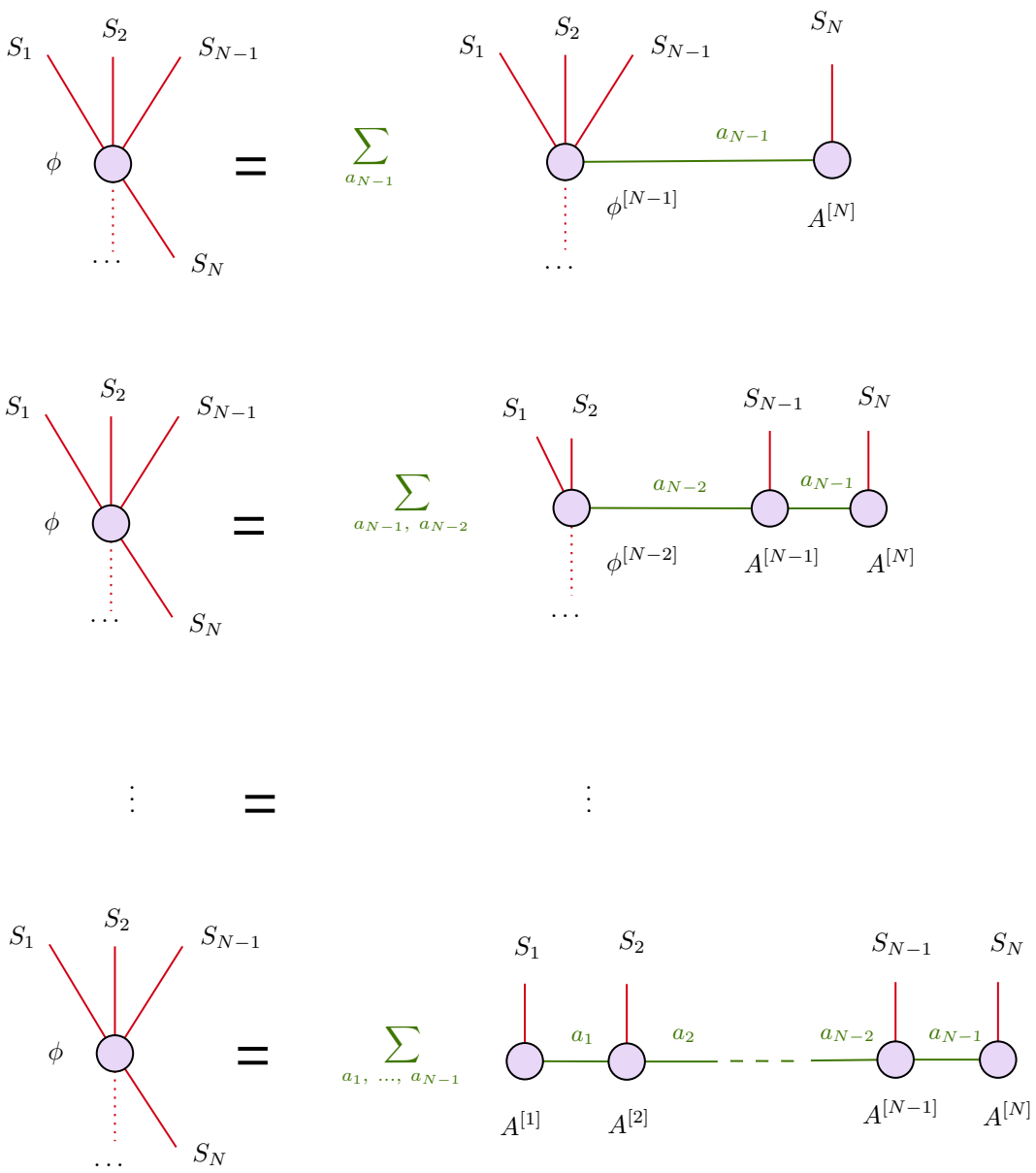

Figure 4: Graphical representation of the steps required to achieve the MPS representation of a N-th order tensor ϕ.

In Figure 5 it is depicted an example to appreciate the influence of the parameter χ considering two different TNs. Start from the two factor graphs associated to two different probabilistic models. On the left we have a FG with 4 nodes and 5 factors [4]. On the right we have a FG with 4 variables and 5 factors, but here one factor is connected with all the 4 variables [5]
On the left the factor graph is converted into its TN representation, and then the MPS representation is retrieved. Note that the MPS is already exploited because all the tensors (the green ones) have two connections, so they have order at most equal to 3. Indeed recall that a tensor with less than 4 indices is already in its MPS form.
On the right the central factor is converted into the central green tensor in the TN representation. It is a tensor of order 4, so the MPS representation will result in a train of 4 tensors.

[4]It can represent e.g. an Ising model with 4 spins and pairwise interactions.
[5]It can represent the same Ising model previously described, but with the addition of a 4 body-interaction.

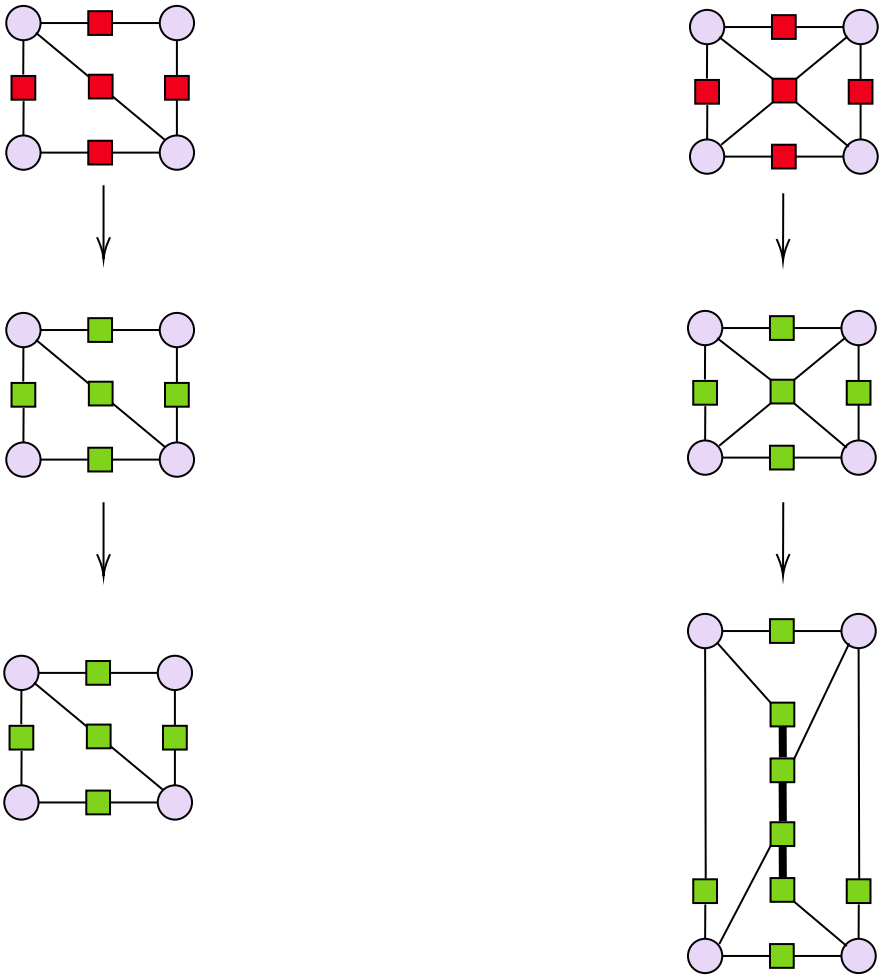

Figure 5: χ controls the precision on the MPS representation of a TN if the MPS representation is not already achieved.
Left: an initial factor graph is considered, where the pink circles represents variables and red squares represent factors. The TN representation is obtained, where the pink circles are now copy tensors and green squares are tensors encoding all the possible values the factors can assume. In the end the MPS representation of each tensor in the TN is obtained just by leaving the TN as it is, because the tensors have order at most equal to 3.
Right: Now a factor have 4 connections. The resulting TN will be graphically equal to the FG, but the MPS representation will be non trivial. Indeed the central green tensor with 4 indices will result in a train of 4 tensors connected according to the original connectivity of the TN before exploiting the MPS representation. The thick black indices have dimension $\leq \chi$. If the initial dimensions of the indices are bounded by a maximum dimension d, than χ is in general considered greater than d to avoid the occurrence of large errors. This is why these indices are drawn thicker.

In the case of a copy tensor with i indices of dimension k, due to its specific structure $\Delta_i \doteq \sum_{x=1}^{i} (e_x)^{\otimes i}$, its **exact** MPS representation can be obtained just by imposing that the first index is equal to the second, the second equal to the third, and so on. This consideration allows us to obtain the following rule for the MPS of a copy tensor:
The exact MPS of a copy tensor with i indices of dimension k require always $\chi = k$. In Figure 6 it is depicted the graphical representation of what we've just said.

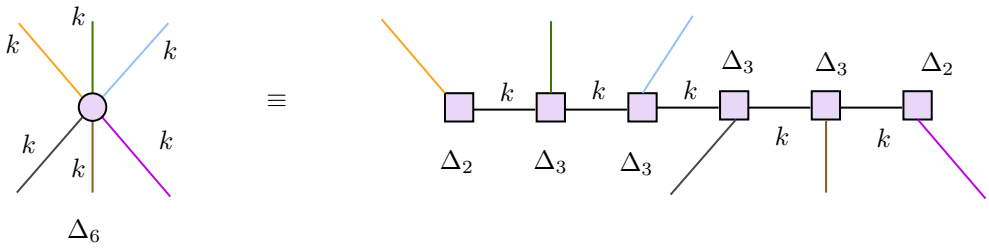

Figure 6: To achieve the MPS representation of a copy tensor it is sufficient to consider $\chi = k$. This procedure do not introduce errors.

$\boldsymbol{\chi}$ controls the precision on the MPS representation of a tensor which is non-trivial only when the tensor considered has an order greater than 3. In statistical mechanics models, as we will see in the chapters devoted to simulations, χ is useful when we are considering k-body interactions between variables, with $k > 3$.

3 SVD of a tensor

For a matrix B the SVD representation of B is a product of 3 matrices U, S, V such that S is a diagonal matrix containing the singular values of B and $B = USV^\dagger$ holds.
The same reasoning can be applied to a tensor A, e.g. with 4 indices, just by collecting two indices in a single bigger one and then depackaging it after SVD [6].

$$A_{\{i,l\},\{j,k\}} = \sum_{a_1} U_{\{i,l\},a_1} S_{a_1,a_1} V^\dagger_{a_1,\{j,k\}}$$

In figure 7 there is the graphical representation of the SVD of a tensor A with 4 indices.

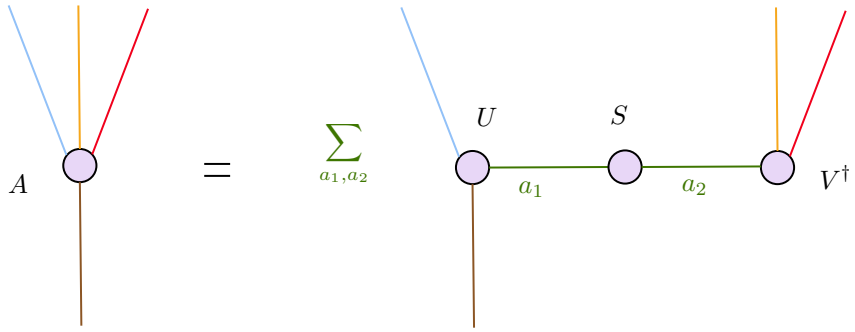

Figure 7: SVD decomposition of a tensor. Indices a_1 and a_2 have the same dimension, because S is a square matrix.

In doing the SVD using the function ITensors.svd() in Julia programming language, two input parameters are considered in the algorithms I developed:

1. $\boldsymbol{\lambda}$: the maximum common dimension allowed for the indices a_1, a_2. This dimension is the number of greatest singular values retained in the representation of S. Controlling λ means controlling the number of memory allocations used because the greater λ the higher the number of entries of U, S, V.

2. $\boldsymbol{\xi}$, the cutoff: it is defined as the desired truncation error of the eigenvalues, i.e. the sum of the squares of the smallest eigenvalues that are not retained. Controlling the cutoff ξ means imposing a specific threshold on the precision of the decomposition output.

4 Transformation of TNs

Considering a TN, our aim here is to find an efficient way to contract it. As we have seen in the first chapter, the partition function Z of a statistical mechanics model can be obtained from the total contraction of the TN obtained from the FG of the probabilist model of interest. In statistical physics, ones we have the partition function Z, all the interesting properties of the model under consideration can be obtained from Z.
This is why an efficient total contraction procedure of a TN is our goal.

4.a Exactly contractible TNs

Let's start from two classes of TNs for which the contraction procedure minimizing the number of operations and memory allocations is known and it is polynomial in the number of tensors. A graphical representation of two classes of this particular TN's is depicted in figure 8.

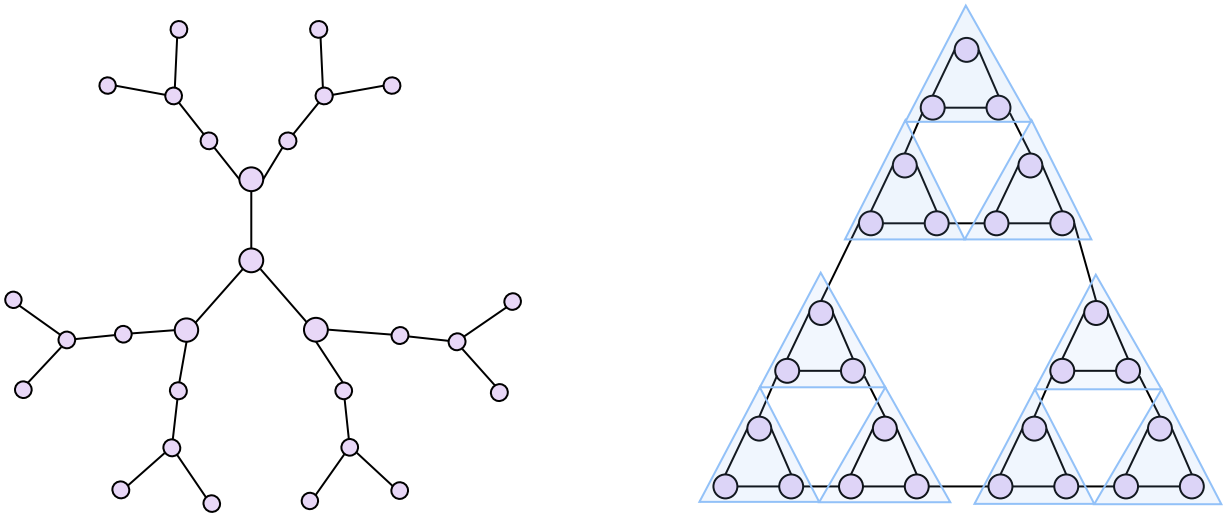

Figure 8: Two classes of TN's for which the contraction ordering is trivial: **on the left** a tree graph and **on the right** a fractal graph.

a) Tensor Networks on Tree Graphs, left panel in figure 8
We consider in the picture a tree TN with N_L layers of third-order tensors. Vectors are put on the outmost boundary. A tree TN's partition function is written as:

$$Z = \sum_{\{a\}} \prod_{n=1}^{N_L} \prod_{m=1}^{M_n} T^{[n.m]}_{a_{n,m,1},a_{n,m,2},a_{n,m,3}} \prod_k v^{[k]}_{a_k}$$

Where

1. $T^{[n,m]}$ is the m-th tensor on the n-th layer
2. M_n is the number of tensors of the n-th layer
3. $v^{[k]}$ is the k-th vectors on the boundary

Now we contract each of the tensor on the N_L-th layer with the corresponding two vectors on the boundary as:

$$v'_{a_3} = \sum_{a_1,a_2} T^{[N_L,m]}_{a_1,a_2,a_3} v^{[k_1]}_{a_1} v^{[k_2]}_{a_2}$$

After the vectors are updated by the equation above, the number of layers of the tree TN becomes $N_L - 1$. The whole tree TN can be exactly contracted by repeating this procedure. We can see from

the above contraction that if the connected graph does not contain any loops, i.e. has a tree-like structure, the dimensions of the obtained tensors during the contraction will not increase. Therefore, the TN defined on it can be exactly contracted. In conclusion, if the graph is a tree, the contraction procedure to make the TN exactly contractible is to start from the leaves and contract them recursively.

b) Tensor Networks on Fractals, right panel in figure 8
Another graph that can be exactly contracted is the TN defined on a fractal, e.g. the Sierpinski gasket. The TN can represent a statistical mechanics model defined on the Sierpinski gasket, such as Ising and Potts model. The tensor is given by the probability distribution of the three spins in a triangle.
After each round of contractions, the dimension of the tensors and the geometry of the network keep unchanged, but the number of the tensors in the TN decreases from N to $N/3$. It means we can exactly contract the whole TN by repeating the above process.

Now let's see an example clarifying why a TN cannot be contracted without a specific procedure. In figure 9 it is depicted a TN and the consequent contraction following a random contraction order. As it is clear from the graphical representation, following a random contraction order generates, step by step, bigger tensors, i.e. a random contraction procedure generates tensors with unbounded number of entries.

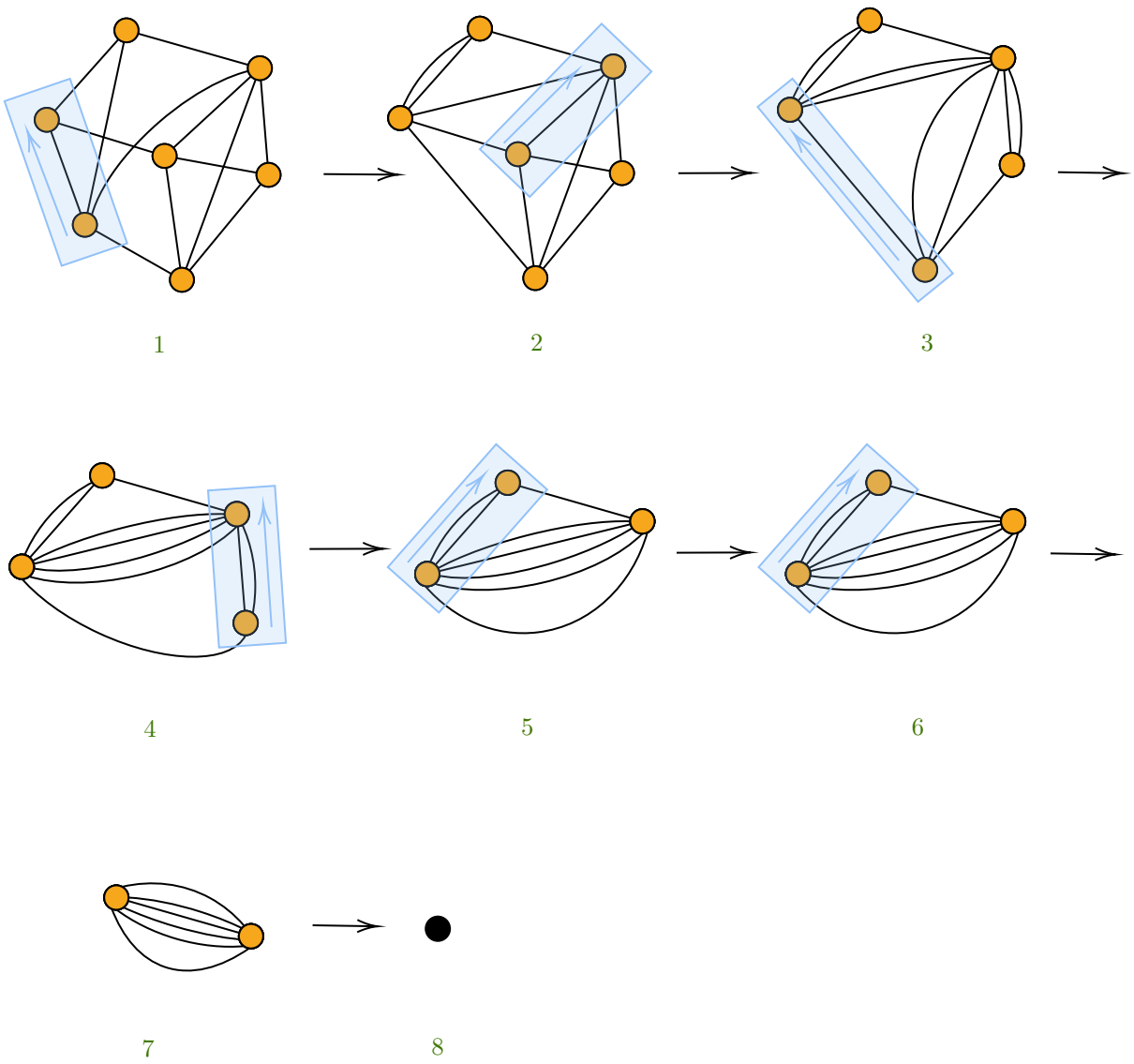

Figure 9: A random contraction procedure is inefficient for the total contraction of an arbitrary TN. In this Figure orange nodes are tensors and black indices are variables shared. Cyan highlighted elements help in understanding the contraction being performed at that moment. Consider the initial TN at step 1: the number of memory allocations is equal to the sum of the entries of the tensors, in this case, considering the variables to have the same domain's cardinality d, then we are storing $d^3 + d^3 + d^5 + d^3 + d^4 + d^4 + d^4$ which goes like d^5 for large d.
Following a random contraction ordering means that in general we are creating bigger tensors. Indeed consider the TN in step 4: the number of memory allocations here is $d^7 + d^3 + d^7 + d^2 + d^3$ which goes like $2d^7$ for large d.

5 Contraction orders for TNs

Given a generic TN, it is not easy to find the exact contraction order of the tensors, here defined as the *contraction procedure*. Indeed this is a $\mathcal{NP}$-complete problem.
Consider a TN as a graph $G = (V, E)$ where V is the set of tensors and E is the set of indices connecting them in the TN.
A contraction ordering π is an ordering of all the edges of G, $\pi(1), \pi(2), \ldots, \pi(|E(G)|)$. The complexity of π is the maximum degree of a merged vertex during the contraction process. The contraction complexity of G, denoted by $cc(G)$, is the minimum complexity of a contraction ordering.
A useful concept: treewidth of a graph
The treewidth of a graph is a useful combinatorial measure of how close the graph is to a tree. Let G be a graph. A tree decomposition of G is a tree $\mathcal{T}$, together with a function that maps each vertex $w \in V(\mathcal{T})$ to a subset $B_w \subseteq V(G)$. These subsets B_w are called bags of vertices.
In addition, the following conditions must hold:

1. $\cup_{v \in V(\mathcal{T})} B_v = V(G)$, i.e. each vertex must appear in at least one bag.
2. $\forall \{u, v\} \in E(G), \exists w \in V(\mathcal{T}), \{u, v\} \subseteq B_w$, i.e. for each edge, at least one bag must contain both of its end vertices.
3. $\forall u \in V(G)$, the set of vertices $w \in V(\mathcal{T})$ with $u \in B_w$ form a connected subtree, i.e. all bags containing a given vertex must be connected in $\mathcal{T}$.

The width of a tree decomposition is defined by $\max_{w \in V(\mathcal{T})} |B_w| - 1$. The treewidth of G is the minimum width over its tree decompositions. In figure 10 there is an example of a graph and its decomposition of width 2 with 6 bags. Computing the treewidth of an arbitrary graph is $\mathcal{NP}$-hard.

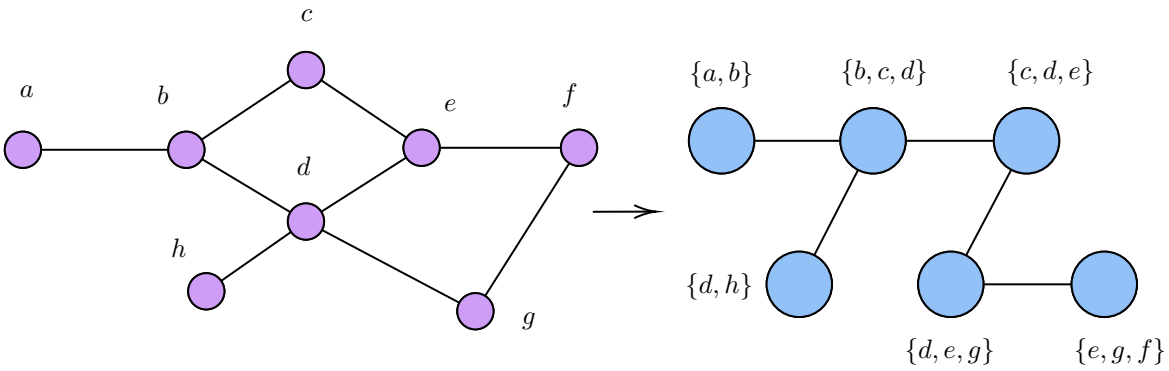

Figure 10: Example of a graph and its decomposition of width 2 with 6 bags.

Theorem by Markov and Shi, [4]: The contraction complexity of a graph equals the treewidth of its line graph: $G = (V, E), cc(G) = tw(G)$. Furthermore, given a tree decomposition $\mathcal{T}$ of width d, there is a deterministic algorithm that outputs a contraction ordering π with $cc(\pi) \leq d$ in polynomial time.

Proof: There is a one-to-one correspondence of the contraction of an edge in G and the elimination of a vertex in $\mathcal{T}$, and the degree of the merged vertex resulting from contracting an edge e in G is the same as the degree of e being eliminated in $\mathcal{T}$. Thus $cc(G) = tw(G)$.
To prove the second part of the statement, denote the tree decomposition by $\mathcal{T}$. Repeat the following until the tree decomposition becomes an empty graph. Choose a leaf l in $\mathcal{T}$. If l is the single vertex of $\mathcal{T}$, output vertices of $\mathcal{T}$ in B_l in any order. Otherwise , let l' be its parent. If $B_l \subseteq B_{l'}$, remove l and repeat the process. Otherwise, let $e \in B_l - B_{l'}$. Output e, remove it from the tree decomposition and continue the process, until all vertices of the tree decomposition are removed. The number of steps in this process is polynomial in the size of the tree decomposition. Note that each output

e appears in only one bag in the tree decomposition. Therefore, all (current) neighbors of e must appear in the same bag. Hence its induced width is at most d. By the one-to-one correspondence of the vertex elimination in $\mathcal{T}$ and the contraction process in G, $cc(G) \leq d$.

Theorem by Robertson and Seymour, [8],[7],[9] : There is a deterministic algorithm that given a graph G outputs a tree decomposition of G of width $O(tw(G))$ in time $|V(G)|^{O(1)} \exp[O(tw(G))]$.
We have proven that finding the contraction ordering minimizing the maximum degree of the intermediate tensors is an $\in \mathcal{NP}$-complete problem. But even if we had this ordering it is not guaranteed that the optimal contraction ordering can be performed.
Indeed, it depends on the computer on which we are running the contraction algorithm, which has limited memory allocations capabilities.
Here we want to build algorithms calculating approximations of TN contractions, whose complexity[6] can be tuned with input parameters, so that the user can select its personal balance between precision achieved on the results and the complexity of the total contraction.
Let's start from a general observation.
Given a generic TN T, it is possible to construct a different TN $\tilde{T}$ such that, defining $C(\cdot)$ as the contraction procedure whose output is the desired property, e.g. the partition function Z of a statistical mechanics model of interest, the following inequality holds:

$$|C(T) - C(\tilde{T})| \leq \epsilon$$

where ϵ is the error committed on the total contraction of the original TN T.
The key idea for the first algorithm I will present in the next chapter, the MCE algorithm, is to device a procedure such that starting from a generic TN T we can iteratively modify the network constructing a TN which is step by step closer to a certain class of TNs, here defined as $\mathcal{MC}$ class.
An example of a TN $\in \mathcal{MC}$ is depicted in Figure 11.
Consider the following two definitions

1. Given a TN, two or more indices are called *parallel indices* if they connect the same two tensors.
2. By saying *modulus parallel indices* I mean "considering one single index when two or more parallel indices are present".

Now we can define the $\mathcal{MC}$ class:
A TN $T \in \mathcal{MC}$ class if and only if the removal of all the chains of matrices modulus parallel indices in the TN leave a chain of matrices modulus parallel indices.
Indeed, an ending structure like the one in figure 11 is exactly contractible in polynomial time by contracting each semicircle of purple tensors accordingly to the **efficient matrix chain contraction** algorithm [7].

[6] The complexity is defined here as the memory allocations required and operations being performed for the total contraction evaluation.

[7] It is given explicitly in the final chapter.

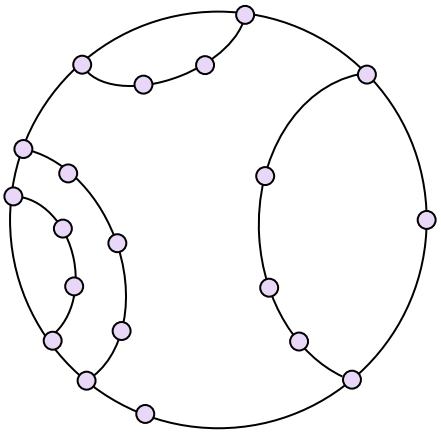

Figure 11: General structure we are interested in, where purple nodes indicate generic tensors and black links indicate variables shared by tensors. This is an example of a TN $\in \mathcal{MC}$ class.

Problem faced

Find a flexible[a] algorithm that, given a generic TN, allows to modify it such that at each step the TN obtained is closer to the $\mathcal{MC}$ class. The final contraction result has an error that does not exceed a user-tunable threshold ϵ.
For a generic TN, the MCE algorithm has this purpose. The strategy used in this algorithm is the iterative dismantling of the lighter cycles in the graphs, where the weight of a cycle is specifically designed for the TN of interest.

[a]MCE is flexible, the user can choose 3 input parameters controlling the balance between precision on the results and complexity of the contraction depending on the specific TN considered.

6 MCE algorithm

The steps of **M**inimum **C**ycle **E**limination algorithm are the following, where green elements refer to the green-numbered-steps depicted in the example in figure 12, while cyan highlighted elements helps in understanding the operations being performed at that moment:

1. Start with the FG of interest (1).
2. After the TN is created (2), find all the connected components in the TN.
   For each of them:
   - 2.1. **If** vectors or matrices are present in the TN
     - i. Contract[8] all the order 1 tensors (vectors) in the TN.
     - ii. Contract efficiently$^{\circledast}$ all the chains of matrices in the TN, (3).
     - iii. Go to point 2.1.
   - 2.2. If all the TN is contracted, go to point 3.
   - 2.3. Convert every remaining tensor of order greater than 3 in its MPS representation, using the input parameter χ as the maximum virtual index dimension.
   - 2.4. Contract efficiently all the chains of matrices in the TN.
   - 2.5. **If** the TN is not totally contracted
     - i. Find the lighter$^{\circledcirc}$ cycle in the TN, (4,5).
     - ii. Do SVD to eliminate the lighter cycle, swapping iteratively the index closing the cycle (i_{13}) with the index not composing the cycle of the subsequent tensor in the cycle (dashed index of n_2 in (5)), (6,7,8).
       In the SVD retain the greatest λ singular values and set the cutoff defined in the previous section, where both λ and the cutoff are input parameters (i.e. indices $i_{12}^{1,*}$ and $i_{12}^{2,*}$ in (8) have a common dimension $\leq \lambda$).
     - iii. Contract efficiently all the chains of matrices in the TN, (9).
     - iv. Parallel indices[9] are seen by the algorithm as a single index, (10,11). This is not a problem because the algorithm is written such that after the MPS representation step the tensor's number of indices do not grow and remain at most equal to 3.
     - v. Go to point 2.1 and then go to point 2.5, (13).
3. **Return** the connected components **and** the collection of all the contraction results on the different connected components, (13).

$^{\circledast}$: a chain of matrices can be efficiently contracted using a recursive algorithm.
$^{\circledcirc}$: the lighter cycle definition depends on the type of FG we are starting from. Possible definition I've tested for MCE algorithm that give very good results are the following:

1. The product of the dimensions of the indices of the tensors composing the cycle.
2. The sum of the degrees of the tensors composing the cycle. This is the heuristic used in the simulations I will present in the final section.

[8] Contracting A on B means that B is replaced by $B \times A$.
[9] Parallel indices are defined as 2 or more indices that connectes the same two tensors.

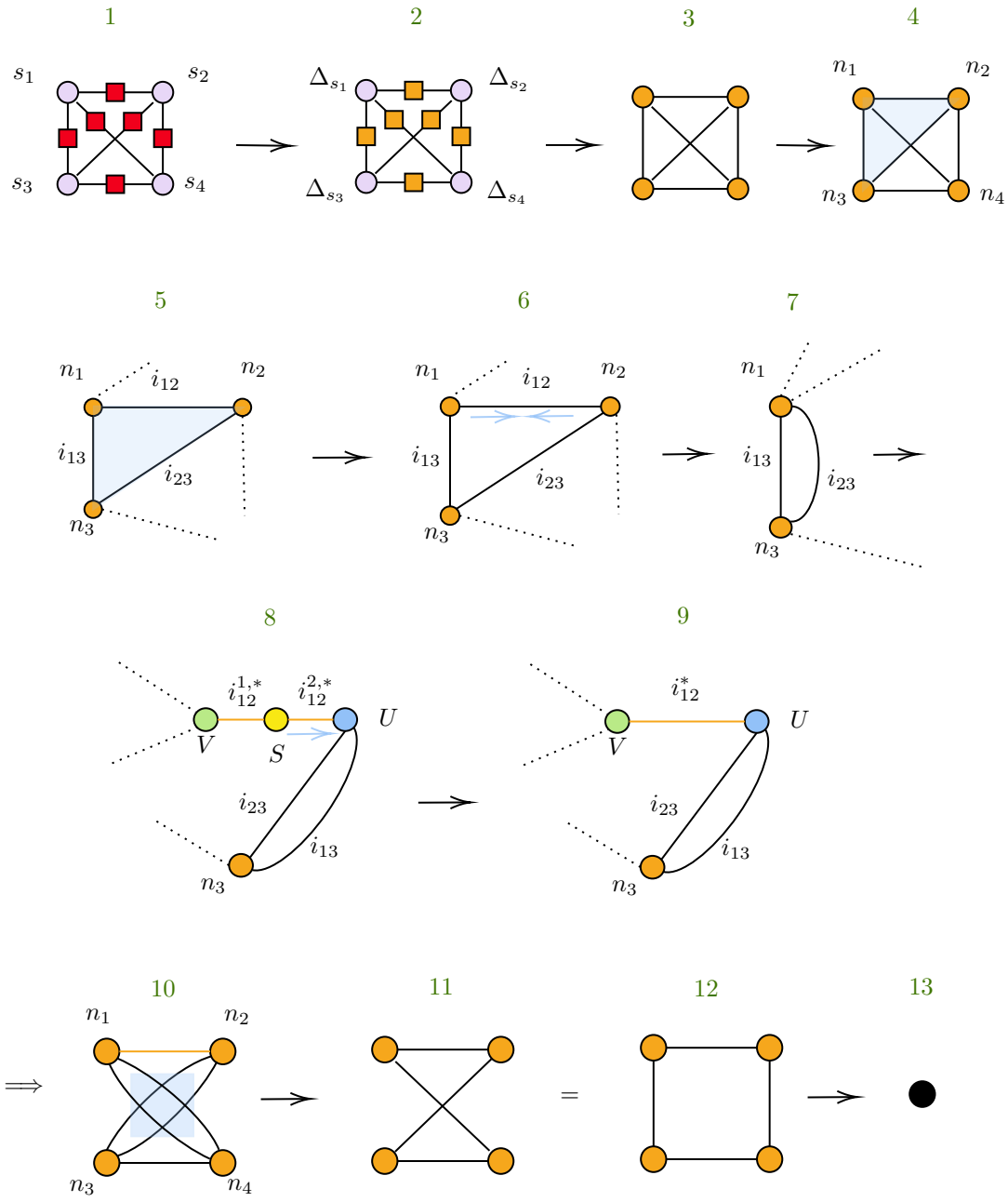

Figure 12: Steps of MCE algorithm starting from a factor graph with 4 nodes and 6 factors.

6.a Contraction heuristics

As previously mentioned, the contraction heuristics I've simulated and whose results are good are two:

1. The product of the dimensions of the indices of the tensors composing the cycle. This method is useful when the distribution of index-dimensions is expected to be uniform and constant in time [10]. This is the case of probabilistic models in which variables have heterogeneous domains' cardinality.

2. The sum of the degrees of the tensors composing the cycle. This method is faster than the previous one, because from a computationally point of view the degree is easier to extract at each step of the algorithm. This method is useful when the distribution of index-dimensions is expected to be picked around a certain dimension and to be constant in time [11]. This is the case of statistical mechanics models like the Ising model and the Potts model, in which variables have the same domains' cardinality [12]. With these assumptions this heuristic is fast and efficient.

Now let's see an example, depicted in figure 13, which will better clarify the cycle dismantling step in the MCE algorithm.

[10] I.e. all index-dimensions in a certain range are present and during the SVD steps of MCE algorithm this distribution remains approximatively the same. Rigid translations can also occur without affecting this heuristic, the only important thing is that the distribution is close to a step function $\mathbb{1}[d \in (a, b)]$ which is equal to 1 if the dimension d is in between a and b and 0 otherwise.

[11] I.e. it is closer to be proportional to a delta function $\delta(a)$ which is 1 if $d = a$ and 0 otherwise and the distribution remains the same during SVD steps in MCE algorithm, with at most a rigid shift.

[12] The indices change dimension during the MCE algorithm because the SVD create new indices with dimension $\leq \lambda$, but as we will see with simulations the distribution of indices remains picked around a certain small range of dimensions, although rigidly shifted

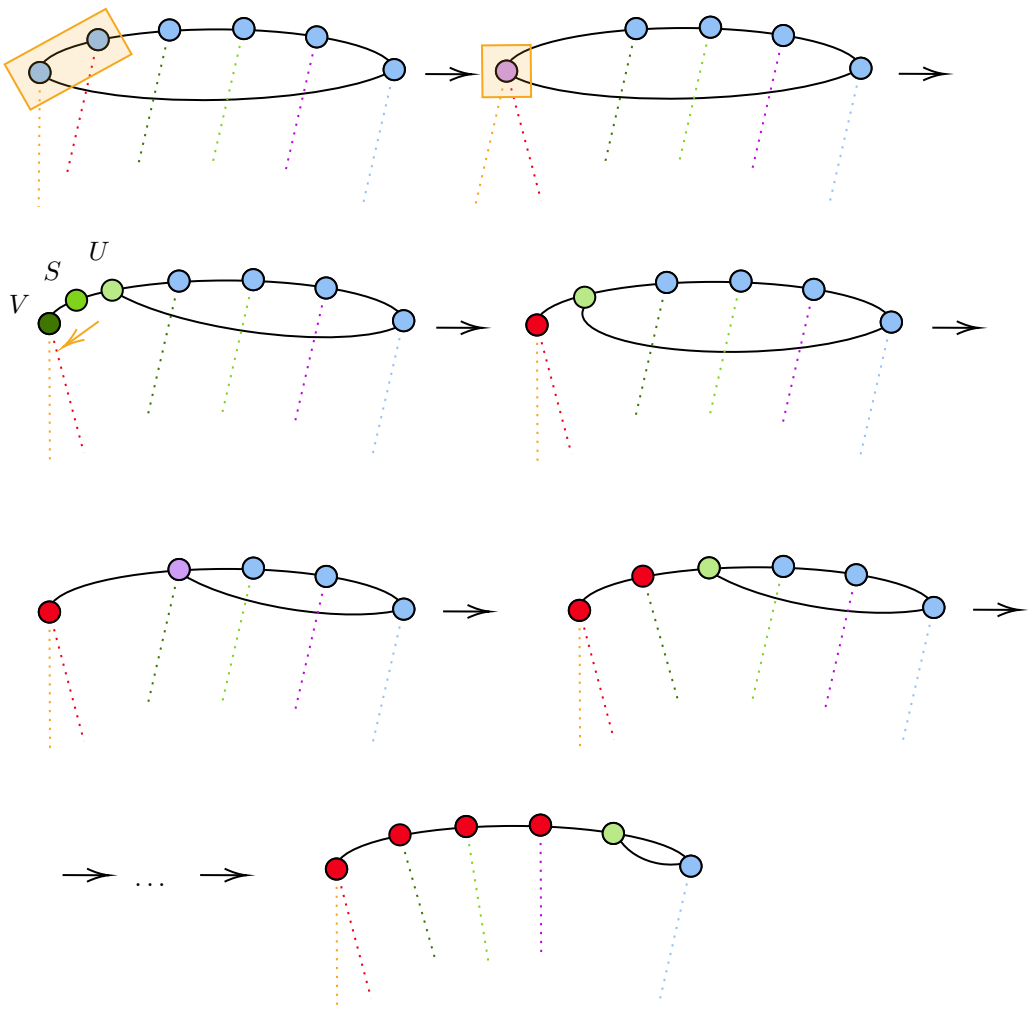

Figure 13: Cycle dismantling procedure in the MCE algorithm. Orange highlighted elements helps in understanding the procedure. Starting from a cycle of length 6 in the end the cycle is eliminated.

7 Triangle-star transformations

There is an interesting polynomial algorithm by Loh et al. [3] called "Bond propagation algorithm", which allows to calculate exactly the partition function Z of an Ising model on an square lattice **if there are no fields** in the network [3].
The procedure involves the triangle-star transformation and its inverse, which can be done *without* increasing the maximum index dimension in the network, which remains equal to 2. This is the reason behind the polynomial time [13] needed for the algorithm to contract all the network, indeed the indices' dimensions do not grow during the contraction.
It is useful to revise this algorithm such that we can answer to two important questions which are not proved in the paper by Loh et al. [3]:

1. Is the zero fields set condition a necessary condition, i.e. do exist classes of Ising model parameters[14] $\{\vec{J}, \vec{h} \neq \vec{0}\}$ such that the algorithm can be safely used ?
2. In the work of Loh et al. it is said that the only requirement is that the graph is planar, then it can be transformed in a square lattice by inserting virtual Ising spins.
   2.1. Is the planarity a necessary condition ?
   2.2. Starting from a planar graph, how can we obtain a square lattice ?

Let's review in figure 14 how the key idea of the bond propagation algorithm is devised.
A complete example of the application of the algorithm is presented in figure 15.

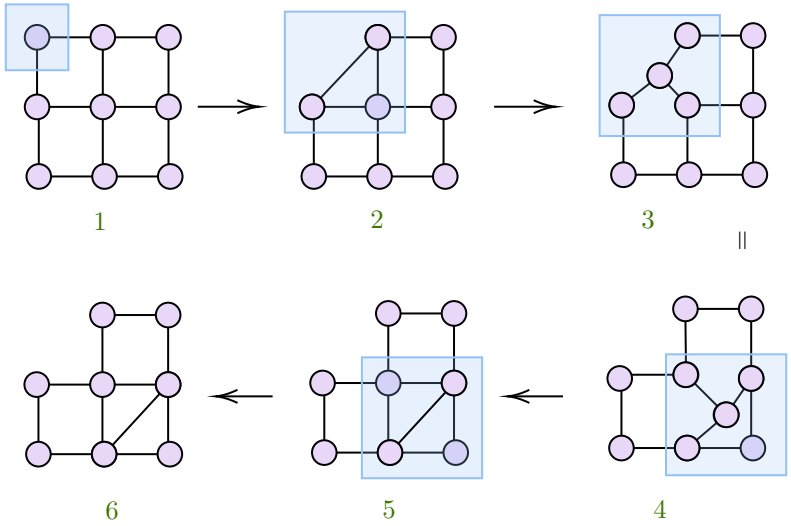

Figure 14: Key element of the bond propagation algorithm [3].
1: Starting from a square lattice, we identify the upper left 3 by 3 cell.
2: The upper left spin is traced off (the one highlighted in cyan).
3: A triangle⇒star transformation is performed.
4: Shift a bit the triangle such that the central node of a triangle is now located on the grid.
5: Perform a star⇒triangle transformation.
6: The bond on the upper corner of the 3 by 3 grid is now located one position down on the diagonal.

[13] Polynomial in the number of Ising spins in the network.
[14] $\vec{J}$ is the set of couplings and $\vec{h}$ is the set of node dependent fields.

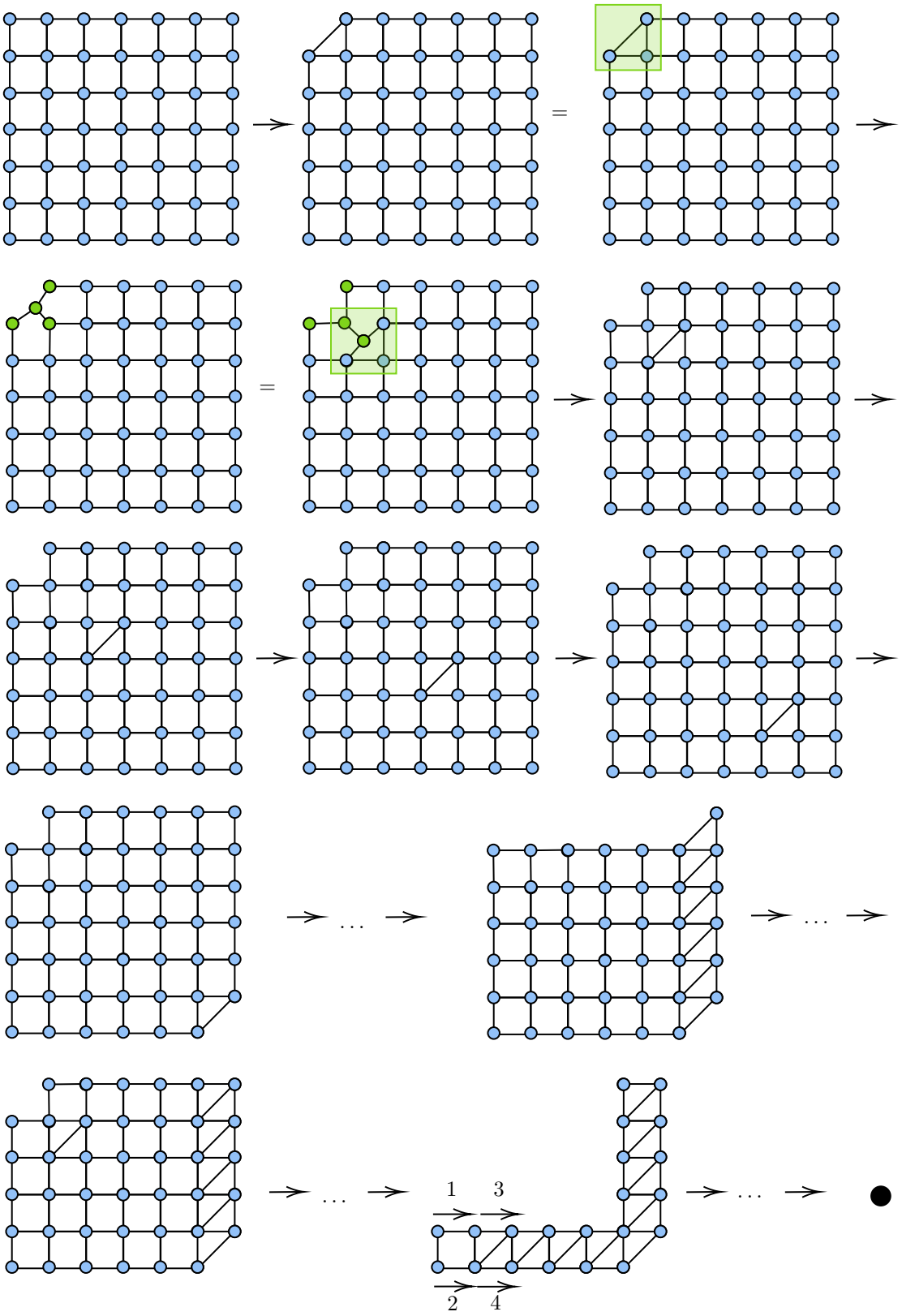

Figure 15: Example of application of the bond propagation algorithm on a square lattice with grid size = 7. The final step is a possible way of concluding the procedure. The numbers in the last row identifies the order of the spins-tracing off. That order is repeated iteratively until the final result, i.e. the partition function Z, is retrieved, here represented as a black dot.

Let's analyse in depth the triangle-star transformation so that we can answer to the questions considered at the beginning of this chapter.
Consider an n-way tensor, call it A. A generic entry if A is labeled by its indices and indicated as $A_{i_1,i_2,\dots,i_n}$. You can group m indices, with $m < n$, in a single index, call it $i_{1\to m}$ and the remaining $n-m$ indices into a second index, call it $i_{m+1\to n}$. Clearly the dimension of $i_{1\to m}$, $\dim(i_{1\to m}) = \prod_{j=1}^{m} \dim(i_j)$ and $\dim(i_{m+1\to n}) = \prod_{j=m+1}^{n} \dim(i_j)$. Then you can perform the SVD of A viewed as a matrix with entries $A_{i_{1\to m},i_{m+1\to n}}$:

$$A_{i_1,i_2,\dots,i_n} = A_{i_{1\to m},i_{m+1\to n}} = U_{i_1\to m;a_1} S_{a_1,a_2} V_{m+1\to n;a_2}$$

Using these procedure, it is possible to achieve a triangle-star transformation, as depicted in figure 16.

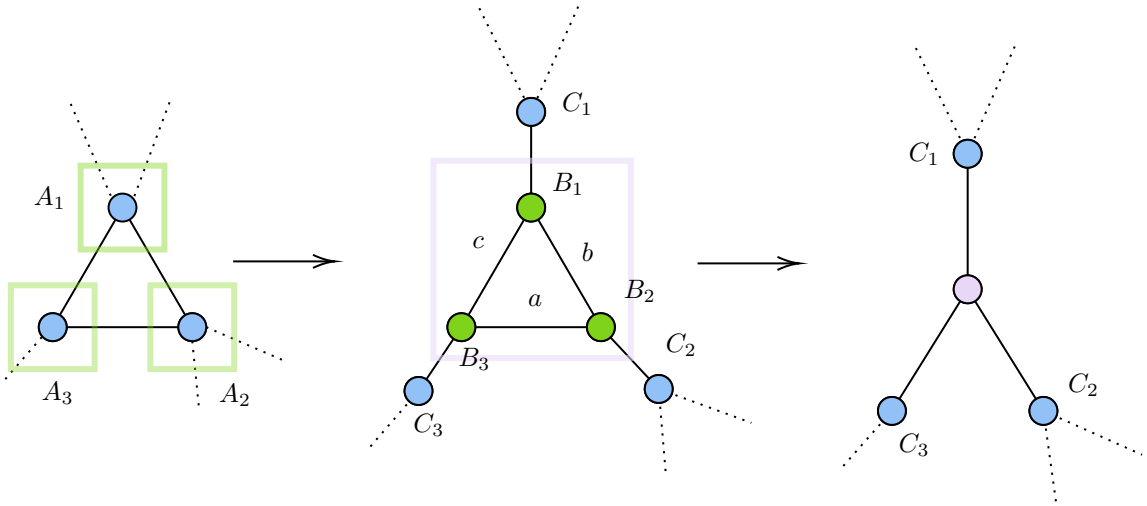

Figure 16: Star transformation of a triangle.
Left: Start with a triangle of tensors $T = A_1, A_2, A_3$, which is located inside a TN. Then apply the decomposition to the three tensors composing the triangle applying the SVD.
Center: $A_i = C_i B_i, \quad \forall i \in \{1,2,3\}$. Now a triangle composed by the B's is obtained.
Right: Contract the triangle composed by the B's obtaining a new central tensor. The triangle star transformation is achieved.

The same procedure can be done to transform a square into a star as depicted in sigure 17.

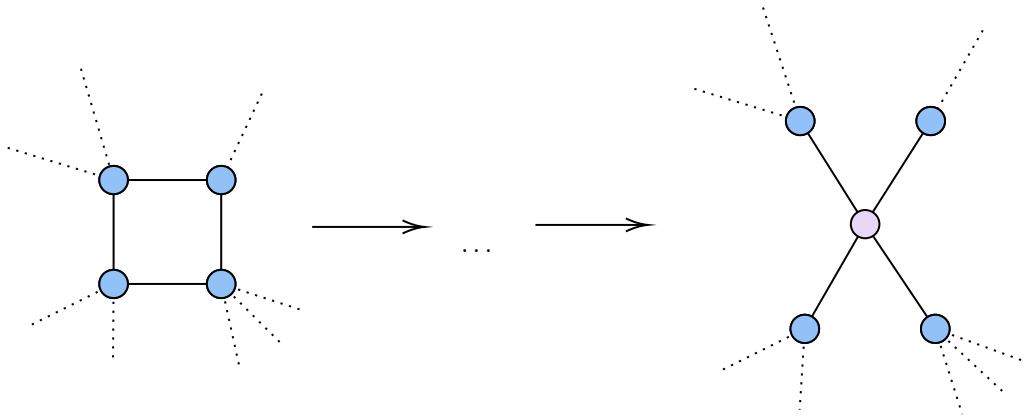

Figure 17: Star transformation of a square of tensors.

It is possible to do this transformation for a general n-side polygon.

Cost of a triangle star transformation

Now we focus on the important quantities characterizing the cost of a triangle star transformation. We can predict that the indices' dimensions will play a key role in this cost. Let's analyse only the triangle→star step[15]. Consider a general triangle in the TN, composed by three tensors $A^{(1)}, A^{(2)}, A^{(3)}$

$$A^{(j)} \in \bigotimes_{i=1}^{n^{(j)}} \mathbb{C}^{d_i^{(j)}} \qquad \forall j \in \{1,2,3\}$$

where $n^{(j)}$ is the number of indices the j-th tensor has and $d_i^{(j)}$ is the dimension of the i-th index of the j-th tensor, with $j \in \{1,2,3\}$.

Each $A^{(j)}$ has to be written as the product of a $n^{(j)}-2$ way tensor and a 3-way tensor:

$$A^{(1)}_{i_1,\dots,i_{n^{(1)}-2},i_{n^{(1)}-1},i_{n^{(1)}}} = \sum_x C^{(1)}_{i_1,\dots,i_{n^{(1)}-2},x} B^{(1)}_{x,i_{n^{(1)}-1},i_{n^{(1)}}} = \sum_x C^{(1)}_{i_1,\dots,i_{n^{(1)}-2},x} B^{(1)}_{x,b,c}$$

$$A^{(2)}_{j_1,\dots,j_{n^{(2)}-2},j_{n^{(2)}-1},j_{n^{(2)}}} = \sum_y C^{(2)}_{j_1,\dots,j_{n^{(2)}-2},y} B^{(2)}_{y,j_{n^{(2)}-1},j_{n^{(2)}}} = \sum_y C^{(2)}_{j_1,\dots,j_{n^{(2)}-2},y} B^{(2)}_{y,a,b}$$

$$A^{(3)}_{l_1,\dots l_{n^{(3)}-2},l_{n^{(3)}-1},l_{n^{(3)}}} = \sum_z C^{(3)}_{l_1,\dots,l_{n^{(3)}-2},z} B^{(3)}_{z,l_{n^{(3)}-1},l_{n^{(3)}}} = \sum_z C^{(3)}_{l_1,\dots,l_{n^{(3)}-2},z} B^{(3)}_{z,a,c}$$

where $i_{n^{(1)}} = j_{n^{(2)}-1} \doteq b, j_{n^{(2)}} = l_{n^{(3)}-1} \doteq a, l_{n^{(3)}} = i_{n^{(1)}-1} \doteq c$ as depicted in Figure 16.

The cost to obtain the exact representation is related to the cost of a single SVD, which for a matrix with indices i_1, i_2 with dimensions respectively d_1, d_2, is known to be

$$O(\min(d_1^2 d_2, d_1 d_2^2)) \doteq O(\min(d_1^2 d_2, \text{rev.}))$$

where $O(\dots)$ mean "of the order of" and "rev." means the reverse powers[16].

For a reduced SVD, i.e. a SVD truncated at the k-th singular value, the cost is known to be

$$O(k d_1 d_2)$$

Consider a single entry of the tensor D, i.e. $(D)_{i,j,k}$,resulting from the contraction of the three B's, which will be the central tensor of the final star configuration

$$(D)_{x,y,z} = (B^{(1)}B^{(2)}B^{(3)})_{x,y,z} = \sum_{a,b,c} B^{(1)}_{x,b,c} B^{(2)}_{y,a,b} B^{(3)}_{z,a,c}$$

The cost of contracting the three B tensors in the new star-central tensor D is equal to $d_x d_y d_z d_a d_b d_c$.

The total cost for a triangle star transformation $C_{\Delta,Y}$ will be the sum of the cost necessary to obtain the three SVD of the three tensors $A^{(1)}, A^{(2)}, A^{(3)}$ and the cost of the contraction of the three tensors $B^{(1)}, B^{(2)}, B^{(3)}$:

$$C_{\Delta\to Y} = O\Big\{ \min\Big(\sum_{j=1}^{3} \prod_{i=1}^{n^{(j)}-2} d_i^{(j)} d_{n^{(j)}-1}^{(j)} d_{n^{(j)}}^{(j)} \,,\ \text{rev.} \Big) \Big\} + \prod_{j=1}^{3} \Big(\prod_{m=1}^{n^{(j)}} rk(A^{(j)}_{[m]}) d_a d_b d_c \Big)$$

where $\prod_{m=1}^{n^{(j)}} rk(A^{(j)}_{[m]})$ is the product of the elements of the multilinear rank of $A^{(j)}$, $A^{(j)}_{[m]}$ is the m-th matricisation of the tensor $A^{(j)}$ and $rk(\cdot)$ is the rank.

We have obtained that the cost of a triangle-star transformation is an increasing function of the indices' dimensions, i.e. it heavily depends on the dimensions of the tensors involved, which was predictable. The cost also depends of the data. Indeed the term referring to the multilinear ranks

[15]The opposite star→triangle transformation will be a different function of the same variables. Now we are looking for other important variables different from the indices' dimensions, indeed the latters will certainly be present.

[16]It's just useful to express it in this way, the reason will be clear in a moment.

add a contribution due to the complexity of the tensors.
In the work of Loh et al.[3], the $\Delta \to Y$ transformation is applied to a square lattice TN considering an Ising model **without fields**. The only requirement is that the underlying graph is planar; indeed we can transform the network of interest in a square lattice. Such generic lattices can be reduced to or embedded in a square lattice by propagating out "effectively diagonal bonds" by inserting zero bonds or infinite bonds[3].
In the work of Loh et al. this procedure is not given, but I will provide a possible way to do that after the chapter devoted to RRC algorithm.
Now let us focus on the first question we asked ourselves at the beginning of this chapter:
Is the zero fields set condition a necessary condition, i.e. do exist classes of Ising model parameters $\{\vec{J}, \vec{h} \neq \vec{0}\}$ such that the bond propagation algorithm can be safely used ?
Clearly we cannot apply the same idea on a general Ising model because this would imply that a generic Ising model is solvable in polynomial time, but maybe there exist some classes of parameters $\vec{J}^* = \{J_{ij}\}_{i,j}, \vec{h}^* = \{h_i\}_i$ such that the same bond-propagation idea can be applied to an Ising model on a generic planar graph with random couplings and random fields belonging to that specific class. Call C the class of parameters $\{J^*, h^*\}$ with $h^* \neq \vec{0}$ of an Ising model on a random planar graph such that the the problem of calculating the partition function Z is $\in \mathcal{P}$.
Here we prove the following statement:

Start from the graph of interest on which the Ising model is considered and insert virtual spins to transform it into a planar graph. Insert new virtual spins with $J = +\infty$ or $J = 0$ couplings such that the resulting graph is a square lattice of size n. We will see in a moment how to do that.
C contains

1. All the possible sets of couplings
2. The sets of fields such that the non-boundaries-fields are 0, as depicted in figure 18.

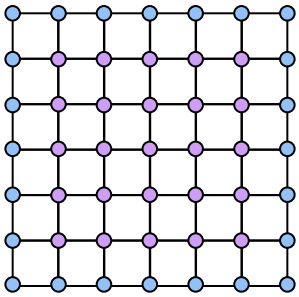

Figure 18: The purple nodes are the ones with field equal to 0. This condition is necessary and sufficient to use the bond-propagation algorithm in [3].

Now we want to prove that the C class is the one defined before.
To retrieve the classes of parameters in C consider the triangle star transformation: we have to prove the equivalence, i.e. both the implications, as shows in figure 19. Here we call the fields with capital H.

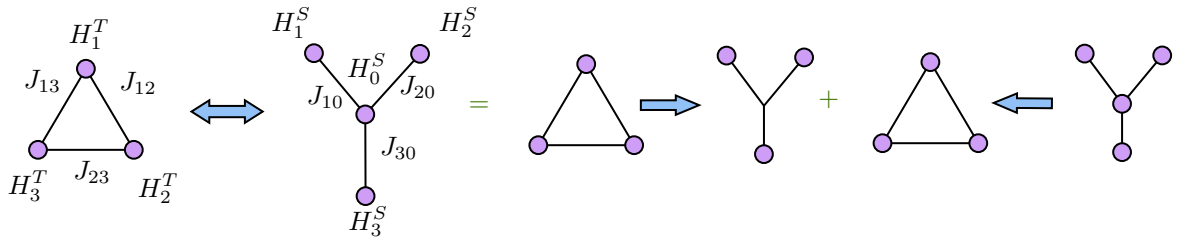

Figure 19: To prove the equivalence both the implications ($\Rightarrow$) and ($\Leftarrow$) are needed: first from a generic triangle we have to obtain a unique star, and then from a general star we have to obtain a unique triangle via an equivalence relation.

Proof
($\Rightarrow$)
Given $s_1, s_2, s_3 \in \{+1, -1\}, J_{km}, H_l \in \mathbb{R}, \forall k, m, l \in \{1, 2, 3\}$, define the Boltzmann weight associated with the configuration of the three spins as $BW(s_1, s_2, s_3)$. We impose that the BW associated with the triangle configuration, namely BW_Δ is equal to the BW of the star configuration, i.e. BW_Y

$$BW_\Delta(s_1, s_2, s_3) = \exp\{J_{12}s_1s_2 + H_1^T s_1 + H_2^T s_2 + H_3^T s_3 + J_{13}s_1s_3 + J_{23}s_2s_3\}$$

$$BW_Y(s_1, s_2, s_3) = \sum_{s_0} \exp\{J_{10}s_1s_0 + J_{20}s_2s_0 + J_{30}s_3s_0 + H_1^S s_1 + H_2^S s_2 + H_3^S s_3 + H_0 s_0 + \delta F\}$$

where δF is a constant energy shift.
We impose $BW_\Delta(s_1, s_2, s_3) = BW_Y(s_1, s_2, s_3), \quad \forall (s_1, s_2, s_3) \in \{+1, -1\}^3$.
Define

1. $\exp\{J_{km}\} \doteq j_{km} \quad \forall k, m \in \{1, 2, 3\}$
2. $\exp\{H_k\} \doteq h_k \quad \forall k \in \{1, 2, 3\}$
3. $\exp\{\delta F\} \doteq \delta f$

We obtain

$$j_{12}^{s_1s_2} h_{T,1}^{s_1} h_{2,T}^{s_2} h_{3,T}^{s_3} j_{13}^{s_1s_3} j_{23}^{s_2s_3} = \sum_{s_0} \left(j_{10}^{s_1s_0} j_{20}^{s_2s_0} j_{30}^{s_3s_0} h_{1,S}^{s_1} h_{2,S}^{s_2} h_{3,S}^{s_3} h_0^{s_0} \right) \delta f =$$
$$= h_{1,S}^{s_1} h_{2,S}^{s_2} h_{3,S}^{s_3} \delta f \sum_{s_0} j_{10}^{s_1s_0} j_{20}^{s_2s_0} j_{30}^{s_3s_0} h_0^{s_0}$$

The last equation must hold for each triple s_1, s_2, s_3 so the fields terms h_1, h_2, h_3 can be simplified, indeed it is just necessary to keep one single parameter, for example δf.

$$j_{12}^{s_1s_2} j_{13}^{s_1s_3} j_{23}^{s_2s_3} = \delta f \sum_{s_0} j_{10}^{s_1s_0} j_{20}^{s_2s_0} j_{30}^{s_3s_0} h_0^{s_0}$$

There are $2^3 = 8$ equations:

$$\begin{cases} (+++): & j_{12}j_{13}j_{23} = \left(j_{10}j_{20}j_{30}h_0 + \frac{1}{j_{10}j_{20}j_{30}h_0}\right)\delta f \\ (++-): & j_{12}\frac{1}{j_{13}}\frac{1}{j_{23}} = \left(j_{10}j_{20}\frac{1}{j_{30}}h_0 + \frac{j_{30}}{j_{10}j_{20}h_0}\right)\delta f \\ (+-+): & j_{13}\frac{1}{j_{12}}\frac{1}{j_{23}} = \left(j_{10}j_{30}\frac{1}{j_{20}}h_0 + \frac{j_{20}}{j_{10}j_{30}h_0}\right)\delta f \\ (-++): & j_{23}\frac{1}{j_{12}}\frac{1}{j_{13}} = \left(j_{20}j_{30}\frac{1}{j_{10}}h_0 + \frac{j_{10}}{j_{20}j_{30}h_0}\right)\delta f \\ (--+): & j_{12}\frac{1}{j_{13}}\frac{1}{j_{23}} = \left(\frac{j_{30}}{j_{20}j_{10}}h_0 + \frac{j_{20}j_{10}}{j_{30}h_0}\right)\delta f \\ (+--): & j_{23}\frac{1}{j_{13}}\frac{1}{j_{12}} = \left(\frac{j_{10}}{j_{20}j_{30}}h_0 + \frac{j_{20}j_{30}}{j_{10}h_0}\right)\delta f \\ (-+-): & j_{13}\frac{1}{j_{23}}\frac{1}{j_{12}} = \left(\frac{j_{20}}{j_{10}j_{30}}h_0 + \frac{j_{10}j_{30}}{j_{20}h_0}\right)\delta f \\ (---): & j_{13}j_{23}j_{12} = \left(\frac{j_{10}j_{20}j_{30}}{h_0} + \frac{h_0}{j_{10}j_{20}j_{30}}\right)\delta f \end{cases}$$

Considering $(++-)$ and $(--+)$ the left hand side is the same, so we obtain

$$j_{10}j_{20}\frac{1}{j_{30}}h_0 + \frac{j_{30}}{j_{10}j_{20}h_0} = \frac{j_{30}}{j_{20}j_{10}}h_0 + \frac{j_{20}j_{10}}{j_{30}h_0}$$

$$\Leftrightarrow \quad j_{10}j_{20}\frac{1}{j_{30}}h_0^2 + \frac{j_{30}}{j_{10}j_{20}} = \frac{j_{30}}{j_{20}j_{10}}h_0^2 + \frac{j_{20}j_{10}}{j_{30}}$$

$$2h_0^2\sinh\left(\frac{J_{10}J_{20}}{J_{30}}\right) = 2\sinh\left(\frac{J_{10}J_{20}}{J_{30}}\right)$$

$$\Leftrightarrow \quad h_0 = \pm 1 \Rightarrow H_0 \in \{0, \pm i\pi\}$$

Considering other couples of equations from the set of 8 equations written before, such that the left-hand side is equal, produces the same result.
Eventually, the 8 equations can be rewritten as

$$j_{12}^{s_1s_2}j_{13}^{s_1s_3}j_{23}^{s_2s_3} = \delta f\sum_{s_0} j_{10}^{s_1s_0}j_{20}^{s_2s_0}j_{30}^{s_3s_0}h_0^{s_0} = \pm\delta f\left(j_{10}^{s_1}j_{20}^{s_2}j_{30}^{s_3} + \frac{1}{j_{10}^{s_1}j_{20}^{s_2}j_{30}^{s_3}}\right)$$

The final equation is

$$e^{J_{12}s_1s_2 + J_{13}s_1s_3 + J_{23}s_2s_3} = \pm 2\delta f\cosh(J_{20}s_1 + J_{20}s_1 + J_{30}s_3)$$

We have obtained that Triangle $\Rightarrow$ Star if and only if $H_0 = 0$ [17], starting from a generic set of fields h_1^T, h_2^T, h_3^T.
$(\Leftarrow)$
It is clear that we cannot start from a generic star configuration, indeed the central field H_0 must be 0, because the set of 8 equations to prove this implication is the same used in $(\Rightarrow)$.

While doing the bond-propagation in the bond-propagation algorithm by Loh et al., the star-triangle transformation will not be possible because the central field is non zero in general, i.e. we have started our analysis with an Ising model with random fields. So it is not possible to apply the bond propagation algorithm to an Ising model on a a planar graph with generic fields without increasing the dimensions of the domains of the new spins s_0. In the case the square lattice Ising model contains fields equal to 0 on the non-boundary nodes, then the bond propagation algorithm can be used.

[17] The case of imaginary fields is not considered, indeed we are working with the Ising model, which has real fields.

8 RRC algorithm

The **R**otated **R**ow **C**limbing algorithm is useful when we have to deal with probability distributions factorized over the edges of a square lattice, were the nodes of the latter are variables.
The steps are described below where green elements refer to the steps depicted in the example in figure 20, while yellow elements help in understanding the procedure:

1. Start with the FG associated with the square lattice, (1).
2. Create the TN with copy tensors (cyan nodes) and tensors encoding the probability distribution's factors (purple squares), (2).
3. Contract the purple squares tensors on one of the neighbor copy tensors along the index whose dimension is the large, (3).
4. **If** the size of the square lattice is 3 go to point 10.
   Otherwise contract the matrix at the bottom-right corner on the tensor above, (4).
5. Contract the tensor on the left of the bottom-right corner of the lattice on the tensor above, call it T, (5,6).
6. Perform SVD of T, (7), creating the tensors U, S, and V, using λ and cutoff as input parameters, (8).
7. Now contract S on V and then V on the purple tensor it is connected to, (9).
8. Repeat the same procedure (i.e. from point 4 to point 7) until the last row is totally contracted, (10, 11).
9. Consider then a rotation of the remaining TN and perform the contraction of rows following the same prescription (steps from 4 to 8), as depicted in figure 21, until a 3 by 3 grid is obtained.
10. Contract the remaining TN as depicted in the last row of figure 21.

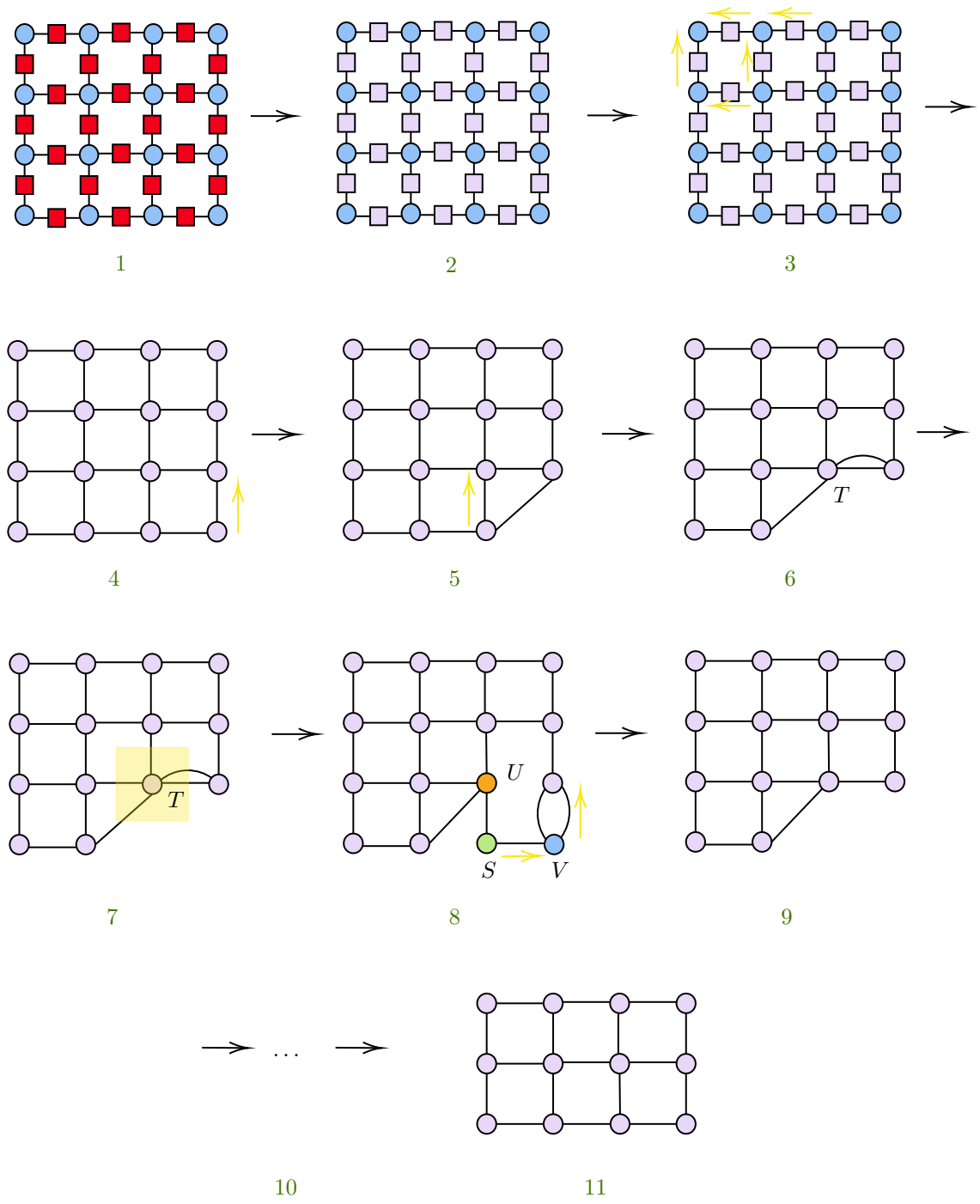

Figure 20: Single step of RRC algorithm.

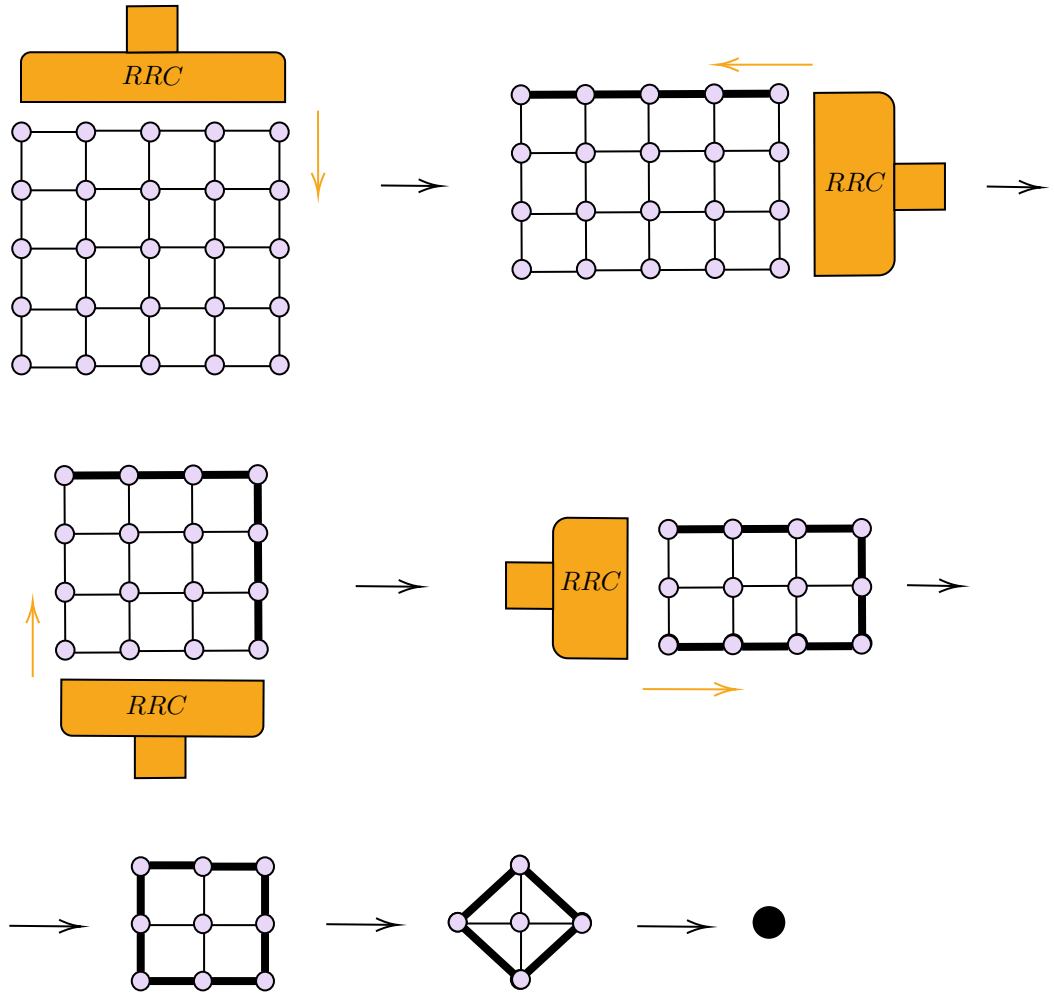

Figure 21: "Machine" representation of RRC algorithm. The thick indices have in general a higher dimension which is still $\leq \lambda$. In the last step contract all the matrices in the TN and then contract the entire TN.

Now we see why and in what situations the rotation part is important in the RRC algorithm.
Consider the figure 22. A non-rotated version of RRC algorithm (called for this reason RC algorithm in the machine representation in Figure 22) is considered for a square lattice with grid size equal to 5.
The first step is equal to RRC procedure initial step, the last row is contracted on the one above it. In a general case the indices created via the SVD procedure are bigger then the ones with which we started, so we draw them with thicker black lines. Then we proceed along the same direction, without rotating the lattice. The SVD will work on bigger tensors, due to the thicker indices we have just analysed, so the SVD will create in general bigger indices; so they grow a second time. They are drawn in red meaning that we have reached the threshold λ. This moment can be present in a sequent step, but without losing in generality we consider the saturation event of the indices to be present at the second row-contraction step. Now we proceed: the SVD can not increase the indices dimensions and errors are introduced; this event is represented by the violet indices, which afre introducing errors. The final result will be a total contraction result which is affected by errors, i.e. a purple dot. On the contrary, by looking at figure 21, we see that no red indices nor violet indices are present.

The rotation component of the RRC algorithm is very useful when we are dealing with small square lattices, but also in the case of bigger ones. Indeed the rotation component is delaying the moment when the problem of saturation of indices will arise.

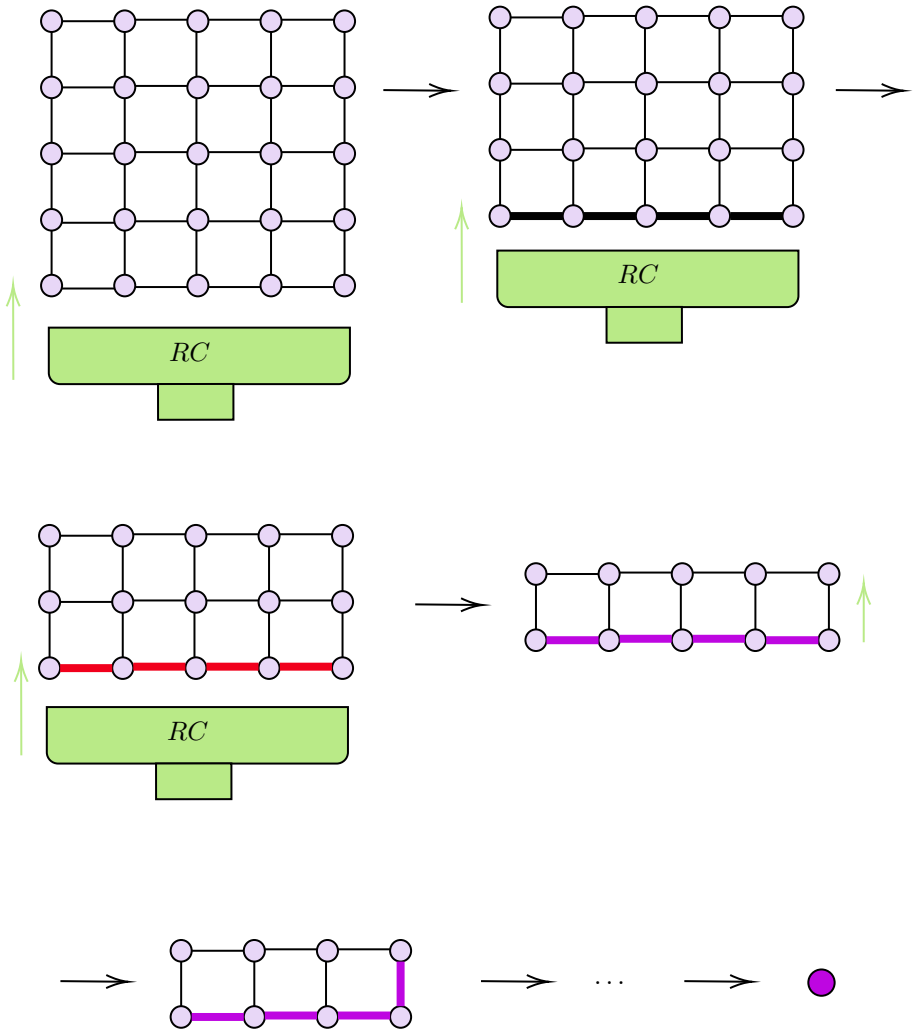

Figure 22: RC algorithm procedure, red indices indicate saturated indices, violet indices indicate the introduction of errors. The contraction result is subjected to errors.

9 Pipeline transformation: all the TNs can be mapped in the class of square lattice TNs

We have seen that RRC algorithm have very good performances, indeed we can safely bound the λ value to small sizes of the order of several tens, due to the geometric symmetries of square lattices. In this chapter we answer the prove the following general statement:

> Given an Ising model with N spins and generic sets of coupling and fields, the pipeline transformation maps the calculation of Ising properties, e.g. Z, $\langle m \rangle, \dots$, in a contraction problem of a connected subgraph of a square lattice TN, where the number of tensors is bound by $N(\frac{N}{2}+1)$ [a].
>
> [a] The equality is reached for example in the case of K-graphs, i.e. the complete graphs.

If we are able to transform a generic TN representing an Ising model into a square lattice TN then the RRC algorithm could be safely applied.
Here I propose a possible way of doing this transformation.

Pipeline transformation
Let's see the rules while looking an example. Green elements refers to green numbered steps in figure 23.

1. Start from the Ising model's graph of interest, (1).
2. Trace off the spins with degree $\in \{1, 2\}$, (1,2).
3. Repeat point 1 until all spins have degree > 2.
4. Select a random spin and mark it as visited, (cyan highlighted tensor in (3)). Call it S.
5. Select a non-visited random spin connected to S and mark it as visited.
6. Repeat point 5 until all spins are visited. This is a random ordering to construct an hamiltonian path. In general it is not possible to construct it, but it's not a problem: if you're stacked, i.e. if you cannot proceed and there is at list one non-visited spin, just select one random non-visited spin, mark it as visited and proceed as in point 4. You end with an ordered list of visited spins, (cyan highlighted arrows in (3)).
7. Put the original ordered list as a train of spins (i.e. 1D chain) connected according to the original connectivity, (Low row in (5)).
8. Select the last spin in the visited list, (Sea-green highlighted tensor in 4) and remove it from the visited list.
9. Use tensor-gadget 1 and tensor-gadget 2, both in their Ising form, to correctly reproduce the component of the Ising network associated to this selected spin, (Upper row in (5) + tensor-gadget 1,2 in Figure 25,26).
10. Iterate points 8,9 until the list of visited spins is empty, (6,7,8,9,10,11,12,13).
11. Transform the network in its associated TN and express each tensor-gadget in its tensor form (Tensor-gadget 1,2 in figures 25, 26). Contract in a single tensor the tensor-gadgets, (14): the square lattice TN is obtained.
12. If you want to complete the square lattice add zero couplings and virtual dummy spins. From the RRC algorithm point of view, this is totally useless.

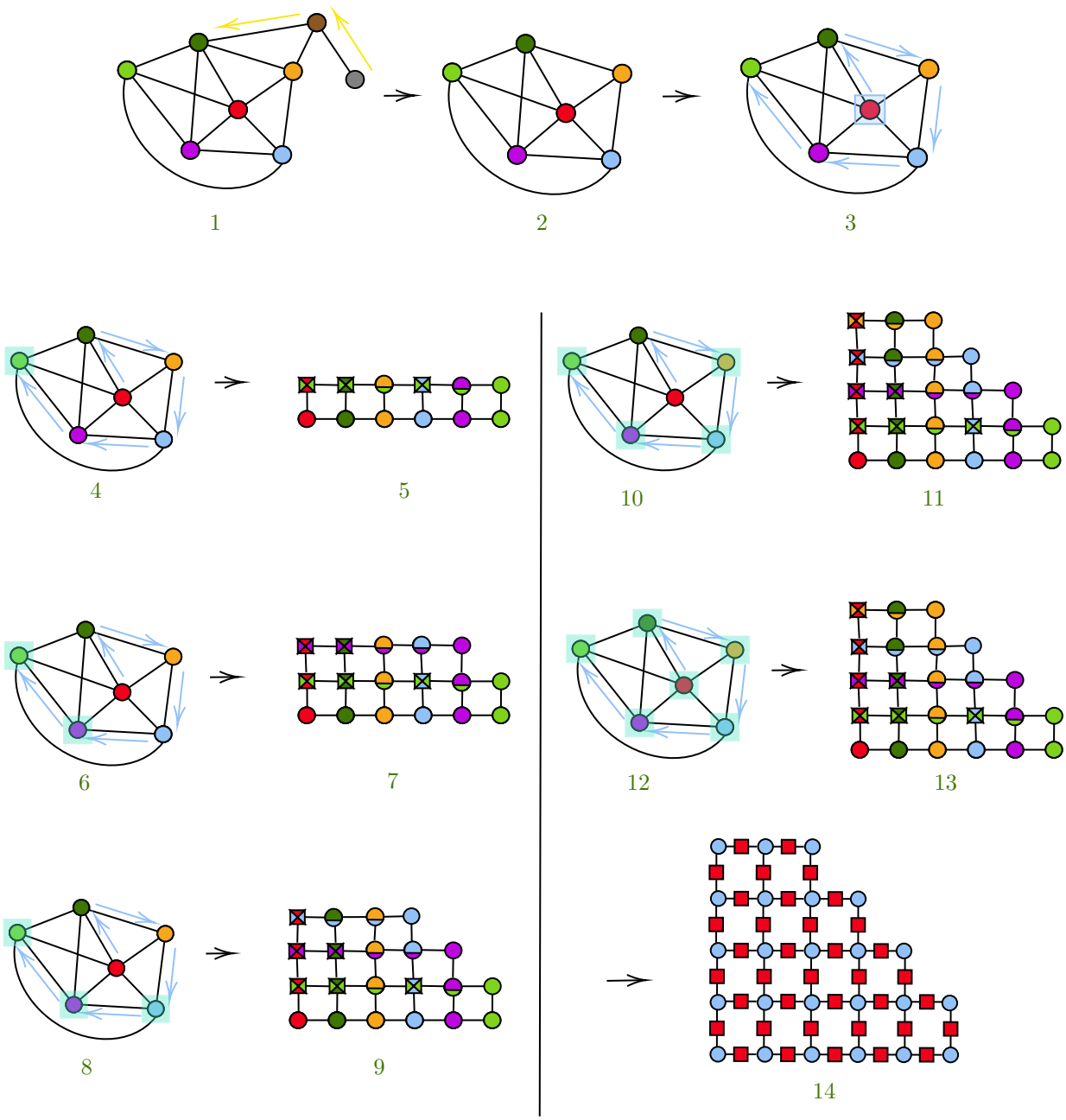

Figure 23: Pipeline transformation of a generic Ising TN in a (portion of a) square lattice TN. The final not-finished square lattice can be concluded inserting zero couplings and new virtual spins. From the RRC algorithm point of view it is clearly not necessary. In the last figure the cyan circles represent copy tensors of order 2 (Ising-like copy tensors) and the red squares are order-2 interaction tensors.

In figure 24 a different visited ordering list is considered, resulting in a different final square lattice. Despite the different structure obtainable **the bound on the number of nodes in the grid is respected**.

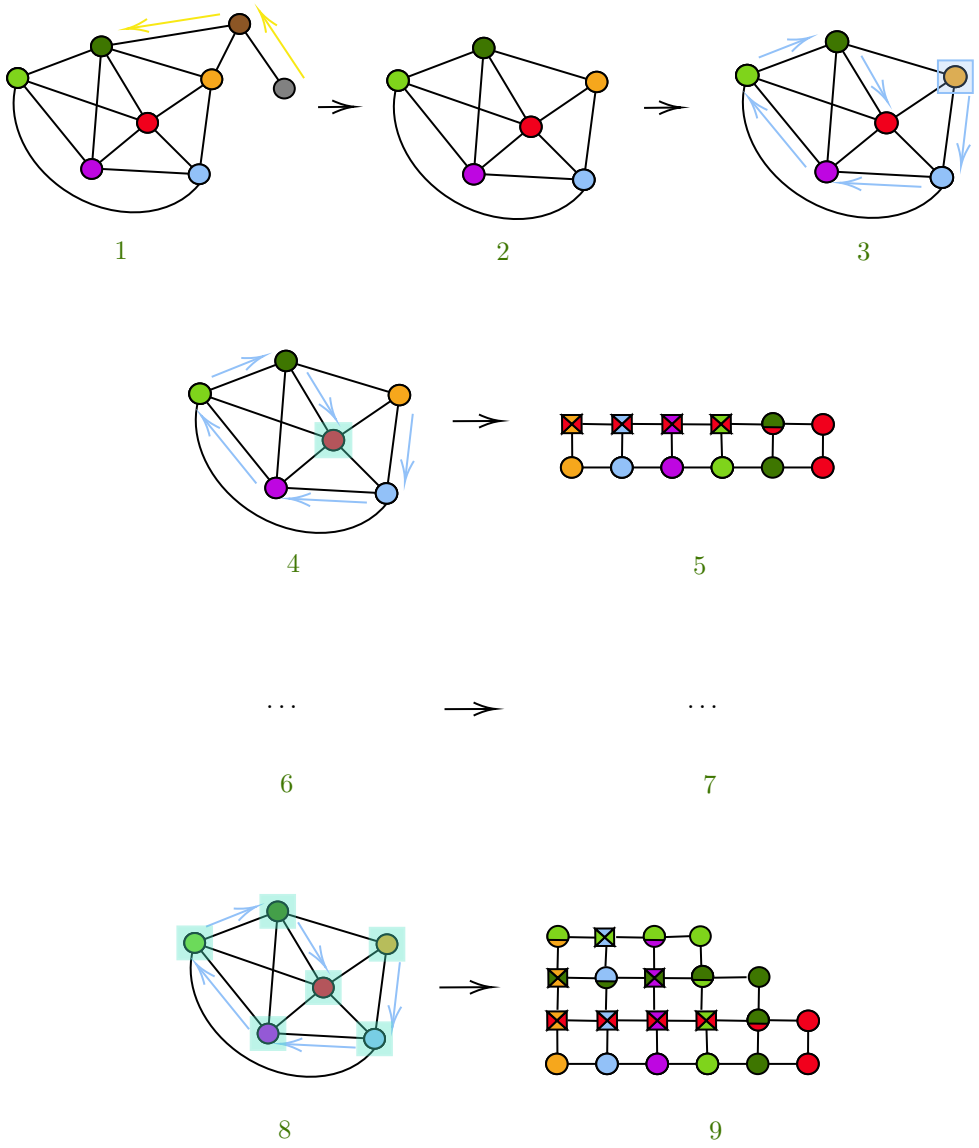

Figure 24: A different choice of the visited list in the Pipeline transformation generates a new square lattice, but the bound on the size $N(\frac{N}{2}+1)$ remains the same.

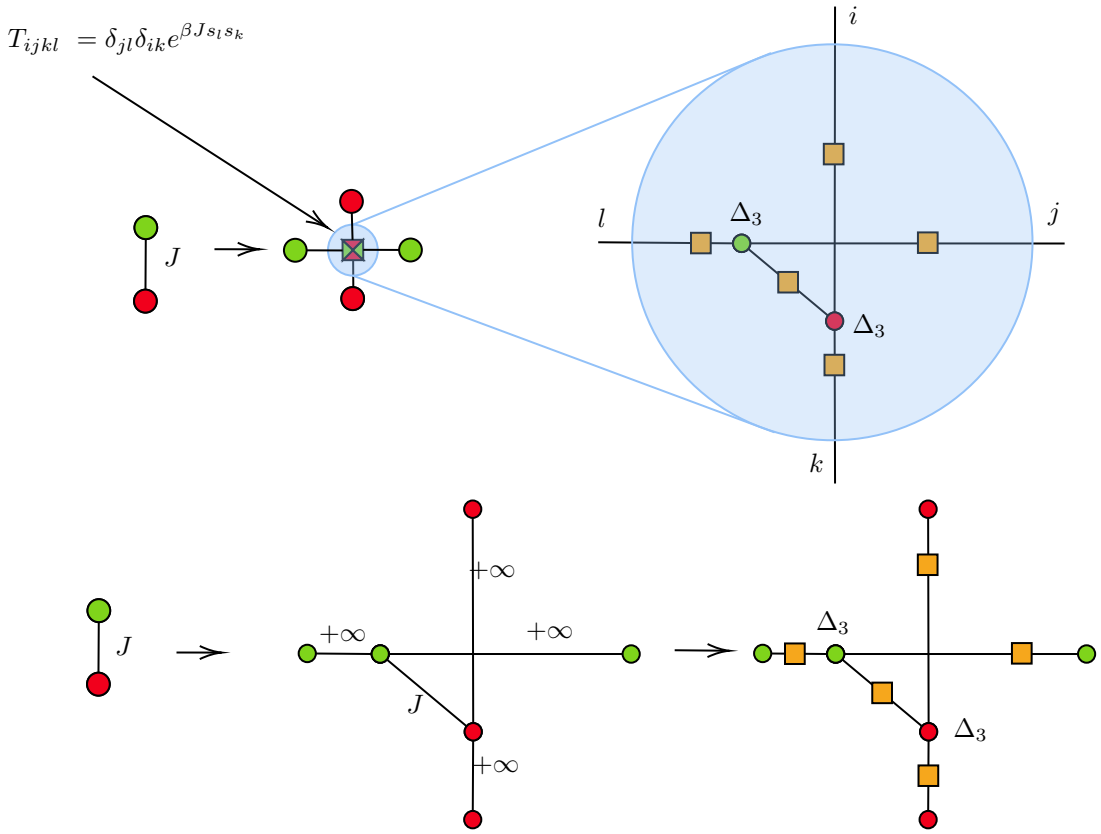

Figure 25: **Up**: Tensor-gadget 1 in the Pipeline transformation. Together with tensor gadget 2, it allows to perform the transformation starting from a generic Ising model and going to a square lattice TN Ising model. Tensor-gadget 1 allows to propagate two spins along a square lattice making them interact with a J coupling.
Down, left: Star from a coupling we want to propagate along a row in the pipeline transformation.
Down, center: Ising form of the tensor-gadget 1. Four new Ising spins are inserted and connected as depicted in the second panel. The original two spins are the green one on the right and the red one on the bottom.
Down, right: Tensor form of the tensor-gadget 1. It is obtained by generating the TN associated with the Ising form.

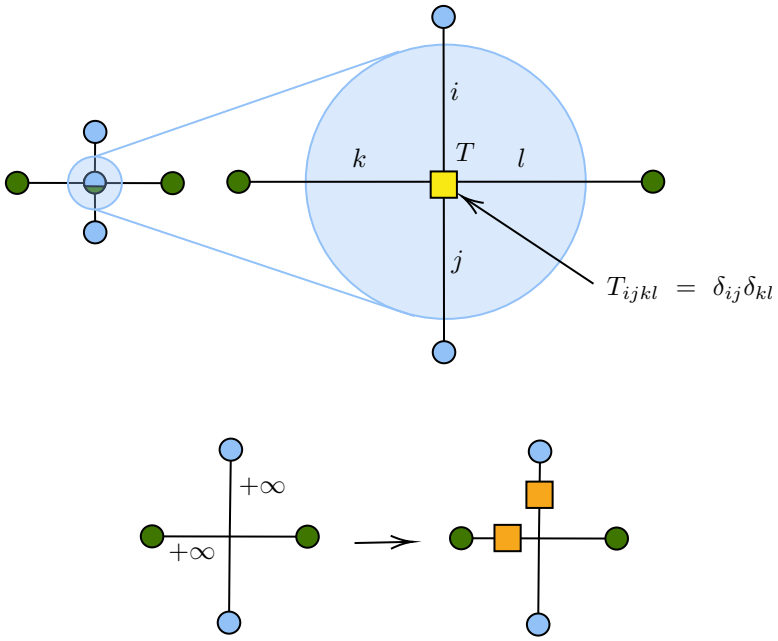

Figure 26: **Up**: Tensor-gadget 2 is used to propagate two spins along the square lattice without making them interact. . T is the gadget tensor, it is a tensor of order 4 expressed as a product of deltas.
Down, Left: Ising form of the tensor-gadget 2. Start with the two original spins (one cyan and one green) and connect each of them with a new spin (cyan with cyan, green with green) , through an infinite coupling $J = +\infty$.
Down, right: Tensor form of the tensor-gadget 2. It is obtained starting form the Ising form and transforming the connected network of 4 spins in a TN.

10 Statistical physics applications: Ising model

Consider an Ising model with N spins, set of couplings $J = \{J_{ij}\}_{i,j\in\{1,\dots,N\},i\neq j}$ and set of node dependent fields $h = \{h_i\}_{i\in\{1,\dots,N\}}$ defined on a factor graph where the nodes are the spins and the factors represents both the pair interaction between spins and the fields.
The goal considered here is to calculate efficiently the average magnetization of the Ising model on arbitrary graphs, for different values of temperatures β, and varying the input parameters λ and ξ [18].
The total contraction of the TN associated with the factor graph of the Ising model gives us the partition function Z whose value depends on the chosen sets of parameters $\{J, h\}$, so I will write $Z(J, h)$ to indicate this dependence even though we know that Z is a number

$$Z(J,h) = \sum_{s_1,\dots,s_N} \prod_{t\in T} A^{[t]}_{\{\vec{s_t}\}} = \sum_{s_1,\dots,s_N} \exp\left\{\beta\Big(\sum_{i=1}^{N}\sum_{j=i+1}^{N} J_{ij}s_is_j + \sum_{i=1}^{N} h_is_i\Big)\right\}$$

where

1. T is the set of tensors in the TN.
2. $\vec{s_t}$ are the indices (spins) of the t-th tensor $A^{[t]}$.
3. $A^{[t]}_{\vec{s_t}}$ is the entry of the t-th tensor $A^{[t]}$ labeled by the set of indices $\{\vec{s_t}\}$

Once we have Z, the free energy F can be computed

$$F(J,h) = -\frac{\log Z(J,h)}{\beta}$$

Consider the problem of determining the **magnetization of a single spin** of interest s_i.
In figure 27 a different visited ordering list is considered, resulting in a different final square lattice. Despite the different structure obtainable **the bound on the number of nodes in the grid is respected**. it is depicted this transformation in a TN from the graphical point of view, starting from the factor graph of an Ising model defined with 4 spins all connected with the others, each with its node-dependent field. The purple circles represents the copy tensors associated with the spins and the red squares represents both pairwise interactions between each couple of spins for which $J_{ij} \neq 0$ and fields terms. The fields h_i can be included in the factors encoding the pairwise interactions, once the fields are divided by the degree of the node i, call it ∂_i.
Indeed, consider

1. t label the tensors in the TN, so t can be a single spin for the field-dependent terms or a couple of spins for the interaction terms.
2. $A^{[t]}_{\{\vec{s_t}\}}(J) \doteq \exp\left\{\beta\sum_{i=1}^{N}\sum_{j=i+1}^{N} J_{ij}s_is_j\right\}$ = the factors (matrices) encoding the interaction terms in the Boltzmann weight.
3. $A^{[t]}_{\{\vec{s_t}\}}(h) \doteq \exp\left\{\beta\sum_{i=1}^{N} h_is_i\right\}$ = the factors (vectors) encoding the field-dependent terms in the Boltzmann weight.
4. ∂_t is the degree of the tensor corresponding to node i, while $\partial_t = \sum_{i\in t}\partial_i$

[18] As we have said, for statistical physics systems in which there are pairwise or three-body interactions the χ input parameter do not introduce errors if it is set equal to the maximum dimension over the variables of the model. In this case $\chi = 2$.

Then

$$Z(J,h) = \sum_{s_1,\dots,s_N} \prod_{t\in T} A^{[t]}_{\{\vec{s}_t\}} = \sum_{s_1,\dots,s_N} \prod_{t\in T} A^{[t]}_{\{\vec{s}_t\}}(J,h) = \sum_{s_1,\dots,s_N} \prod_{t\in T} A^{[t]}_{\{\vec{s}_t\}}(J) \cdot A^{[t]}_{\{\vec{s}_t\}}(h/\partial_t) =$$

$$= \sum_{s_1,\dots,s_N} \exp\left\{\beta \sum_{i=1}^{N}\sum_{j=i+1}^{N} J_{ij}s_i s_j\right\} \prod_{j=1}^{N}\prod_{k=1}^{\partial_j} \exp\left\{\beta \sum_{i=1}^{N} \frac{h_i}{\partial_k} s_i\right\} =$$

$$= \sum_{s_1,\dots,s_N} \exp\left\{\beta\left(\sum_{i=1}^{N}\sum_{j=i+1}^{N} J_{ij}s_i s_j + \sum_{i=1}^{N} \frac{\partial_i}{\partial_i} h_i s_i\right)\right\} =$$

$$= \sum_{s_1,\dots,s_N} \exp\left\{\beta\left(\sum_{i=1}^{N}\sum_{j=i+1}^{N} J_{ij}s_i s_j + \sum_{i=1}^{N} h_i s_i\right)\right\}$$

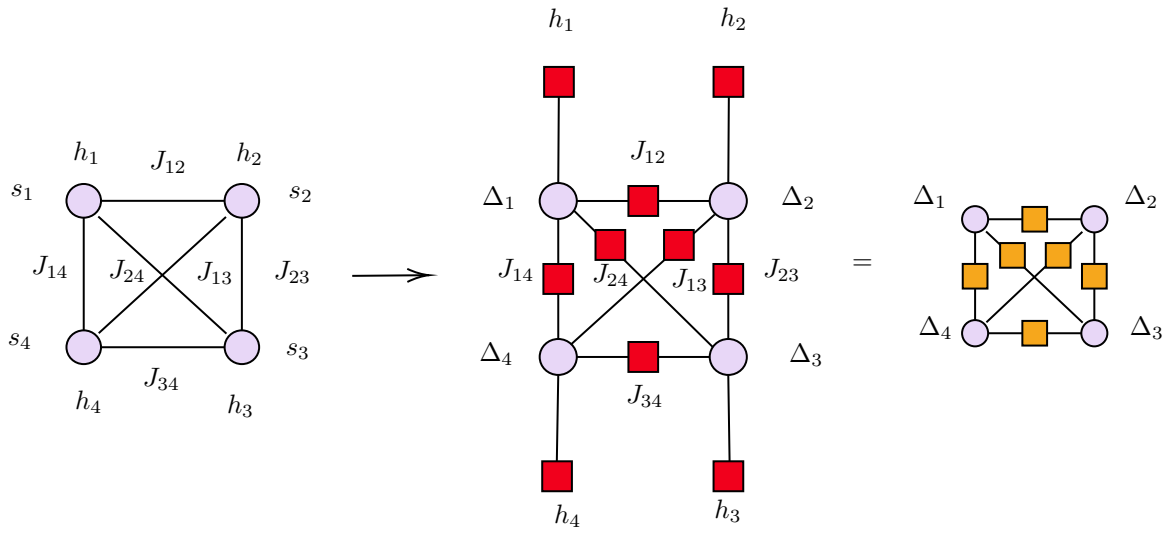

Figure 27: **Left:** start from a factor graph representing an Ising model e.g. on the complete K_4 graph, where node-dependent fields h_i are present.
Center: Construct the associated TN. Purple circles are copy tensors and red circles are tensors.
Right: include the field tensors into the coupling tensor, obtaining tensors encoding the entire couple-dependent Boltzmann weight terms. Recall that the field is rescaled by its degree.

The magnetization of the i-th spin can be computed as

$$m_i = \frac{m_i \times Z}{Z} = \frac{1}{Z}\langle s_i\rangle \times Z = \frac{1}{Z} \sum_{\{s_i\}_{i\in\{1,\dots,N\}}} s_i e^{-\beta H}$$

Consider now the left panel of figure 28 a different visited ordering list is considered, resulting in a different final square lattice. Despite the different structure obtainable **the bound on the number of nodes in the grid is respected**.. This is the representation of a portion of a TN built to represent an Ising model as described above. By looking at the expression we have just obtained for the magnetization of a single spin, m_i can be obtained by first contracting the entire network obtaining Z, and then contracting again the original network adding a leaf (a vector) to the node i with 2 entries, namely $(+1,-1)$.

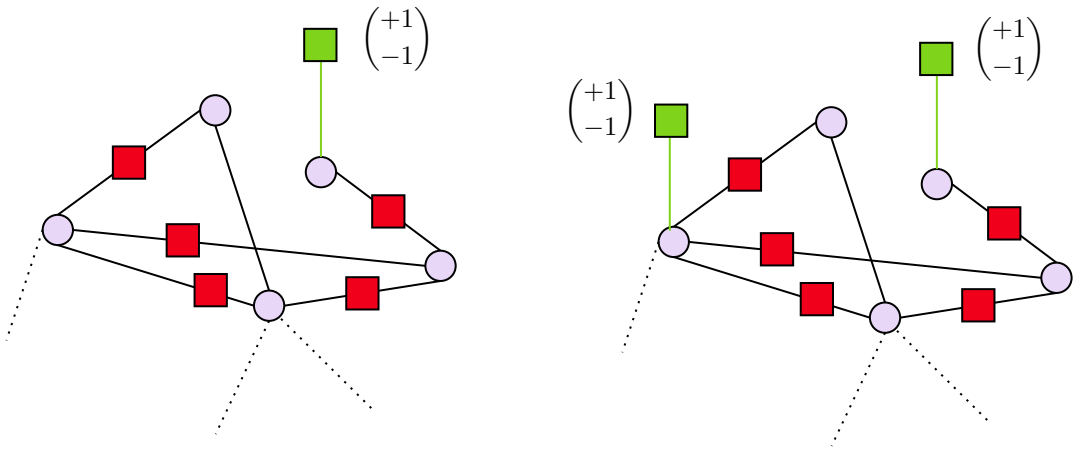

Figure 28: **Left**: Calculation of $\langle s_i \rangle \times Z$. **Right**: Calculation of $\langle s_i s_j \rangle \times Z$

Following the same reasoning we can evaluate the **average magnetization**:

$$m = \frac{1}{N}\left\langle \sum_{i=1}^{N} s_i \right\rangle = \frac{1}{N}\frac{1}{Z} \sum_{\{s_i\}_{i\in\{1,\dots,N\}}} \left(\sum_{i=1}^{N} s_i \right) e^{-\beta H} = \frac{1}{Z} \sum_{\{s_i\}_{i\in\{1,\dots,N\}}} T_{s_1,\dots,s_N} e^{-\beta H}$$

where $T_{s_1,\dots,s_N} = \frac{1}{N}\sum_{i=1}^{N} s_i$ is the entry of a N-order tensor we have to add to the network with the same prescription defined in the case of the single node magnetization.
Clearly *this method is not efficient to compute the average magnetization* because we have an extra N factor due to the computation of the magnetization of each spin.

The average magnetization m can be computed via a direct derivative of F but computationally it's not the best way, because it involves differences between very close quantities.
m can be obtained by expanding $\log Z$ around a small imaginary field vector $\tilde{h}\mathbb{1} = ix\mathbb{1}$, where x is a small real number and $\mathbb{1}$ is a vector with N entries, all equal to 1.
Let's call $\vec{h}_0$ the vector of original fields $\{h_i\}_{i\in\{1,\dots,N\}}$.
If x is chosen sufficiently small, the expansion can be safely stopped to the first order:

$$\log Z(J,\vec{h}') = \log Z(J,\vec{h}_0 + \tilde{h}\mathbb{1}) = \log Z(J,\vec{h}_0 + ix\mathbb{1}) \simeq \log Z(J,\vec{h}_0) + \left\{ \nabla_{\vec{h}} \log Z(J,\vec{h}) \right\}\Big|_{\vec{h}=\vec{h}_0} \cdot \mathbb{1}ix =$$

$$= \log Z(J,\vec{h}_0) + ix\beta \frac{1}{Z(J,\vec{h}_0)} \sum_{\{s_1,\dots,s_N\}} \exp\left\{ \beta \sum_{i=1}^{N} \left(\sum_{j=i+1}^{N} J_{ij} s_i s_j + h_i s_i \right) \right\} \sum_{i=1}^{N} s_i =$$

$$= \log Z(J,\vec{h}_0) + ix\beta \left\langle \sum_{i=1}^{N} s_i \right\rangle$$

$$\Rightarrow m \simeq \frac{1}{x\beta N} \mathfrak{Im}\left\{ \log Z(J,\vec{h}_0 + ix\mathbb{1}) \right\}$$

where $\mathfrak{Im}\{\cdot\}$ is the imaginary part.
In this way **a single contraction procedure produces** a quantity $\log Z(\vec{J},\vec{h}')$ that allows to obtain **both the average magnetization** as the imaginary part of it **and the partition function** as its real part. In a single run of the selected contraction algorithm both the average magnetization and the free energy are obtained.

11 MCE algorithm for the Ising model on random regular graphs

Consider an Ising model on a random regular graph with N nodes, where $N \in \{20, 30 \dots, 100\}$. We analyze here the worst case for the MCE algorithm, i.e., where there are no specific symmetries to exploit, both in the structure of the TN and in the entries of the tensors: a random regular graph with the Ising model sets of parameters $\{J, h\}$ extracted from a uniform distribution.
Using the formulas of the previous section, an approximation of the average magnetization can be computed and compared with Monte Carlo simulation's estimates. For the present case, the Wolff algorithm is very useful for this purpose.
The Wolff algorithm is run with 1.000.000 samples, so the absolute errors with respect to it have a resolution of $\frac{1}{\sqrt{1.000.000}} = 10^{-3}$, i.e. the absolute errors between MCE and Wolff results are meaningful only if they are greater than 10^{-3}. In the following figures, the cyan highlighted regions identify the regions below this resolution, i.e. all absolute errors in the cyan regions are considered as equal to 0 with respect to the Wolff algorithm estimates.
All simulations I will present are given with error bars associated with means and standard deviations of the evaluated quantities, together with scatter plots for the two magnetization estimates and computational times for the MCE algorithm.
Simulations outputs:

1. *Random*[19] Ising model on a random regular graph with degree 3, 20 samples for statistics, β, N varying where N is the number of Ising spins and β is the inverse temperature.
   - 1.1. Figure 29 : $\boldsymbol{\xi = 10^{-3}, \lambda = +\infty, \chi = 2, \beta \in [0.1, 0.3, 0.5, 0.7, 0.9]}$ and computational times needed.
   - 1.2. Figure 30: $\boldsymbol{\xi = 10^{-4}, \lambda + \infty, \chi = 2, \beta \in [0.1, 0.3, 0.5, 0.7, 0.9]}$ and computational times needed.
   - 1.3. Figure 34, 35, 36, 37, 38: $\boldsymbol{\xi = 10^{-3}, \lambda \in [10, 20, 30, 40, 50], \chi = 2, \beta \in [0.1, 0.3, 0.5, 0.7, 0.9]}$ and computational times needed.
   - 1.4. Figure 39: distribution of dimension of indices in performing simulations in figures 34, 35, 36, 37, 38.
2. Random Ising model on a random regular graph with degree 4, 20 samples for statistics, β, N varying.
   - 2.1. Figure 31: $\boldsymbol{\xi = 10^{-3}, \lambda = +\infty, \chi = 2, \beta \in [0.1, 0.5, 0.9]}$ and computational times needed.
   - 2.2. Figure 32: $\boldsymbol{\xi = 10^{-4}, \lambda = +\infty, \chi = 2, \beta \in [0.1, 0.5, 0.9]}$ and computational times needed.
3. Ferromagnetic, $\vec{J} = \vec{1}$ with random fields, Ising model on a random regular graph with degree 4, 20 samples for statistics, β, N varying.
   - 3.1. Figure 33: $\boldsymbol{\xi = 10^{-3}, \lambda = +\infty, \chi = 2, \beta \in [0.1 : 0.0.1 : 1.0]}$ and computational times needed.
   - 3.2. Figure 40: the computational times are analysed for different inverse temperatures $\in \{0.1 : 0.01 : 1.0\}$ in the particular case of zero fields.

[19] Random Ising model means that the coupling and field parameters are extracted from a uniform distribution for each $N, \beta, \text{sample}, \xi, \lambda, \chi$.

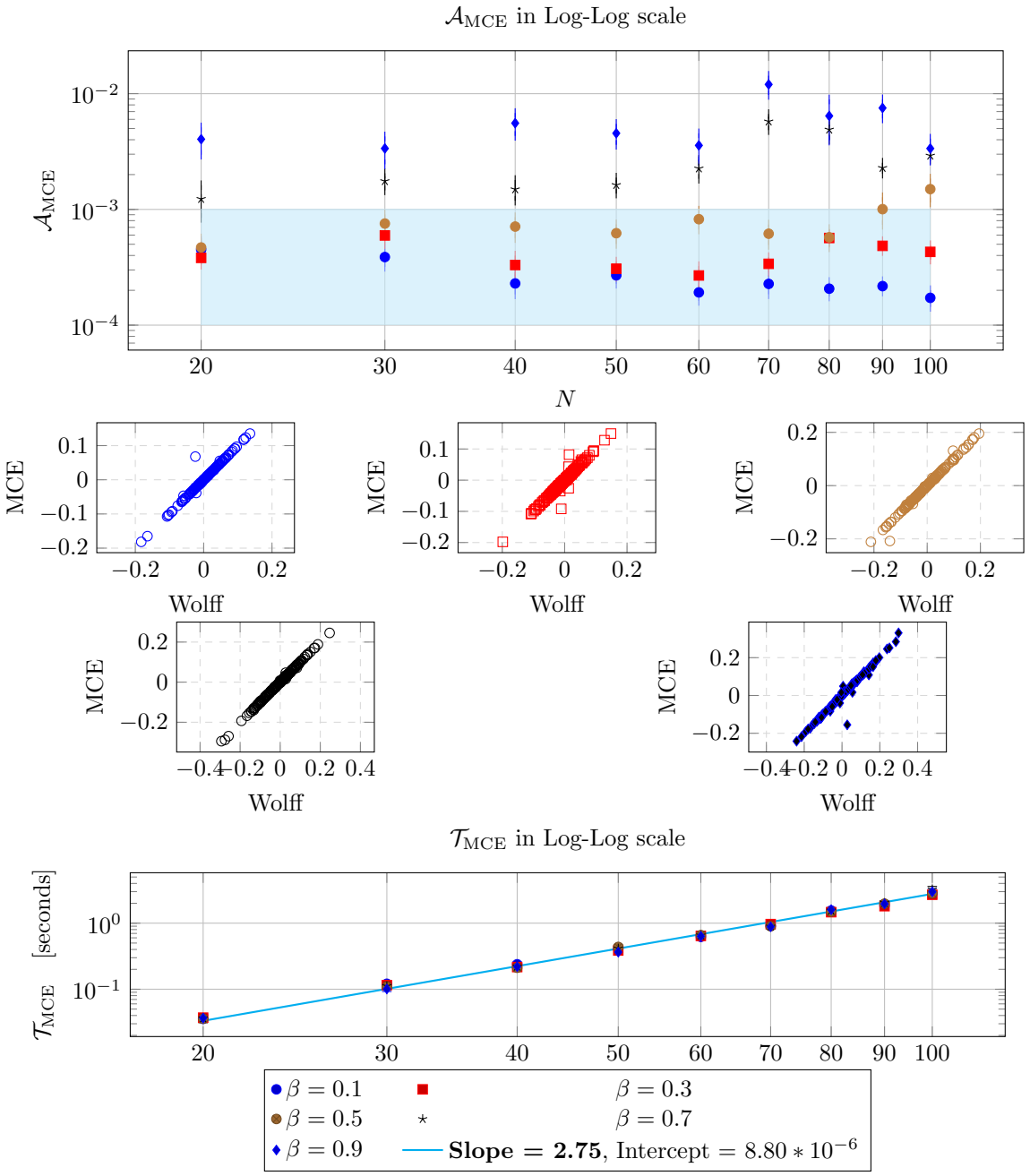

Figure 29: Input parameters: $\boldsymbol{\xi = 10^{-3}, \lambda = +\infty, \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 3, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 5 panels of MCE vs Wolff data, the entire dataset for all values of N and for all the 20 samples are presented for each value of β, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = N^{\textbf{Slope}} +$ intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

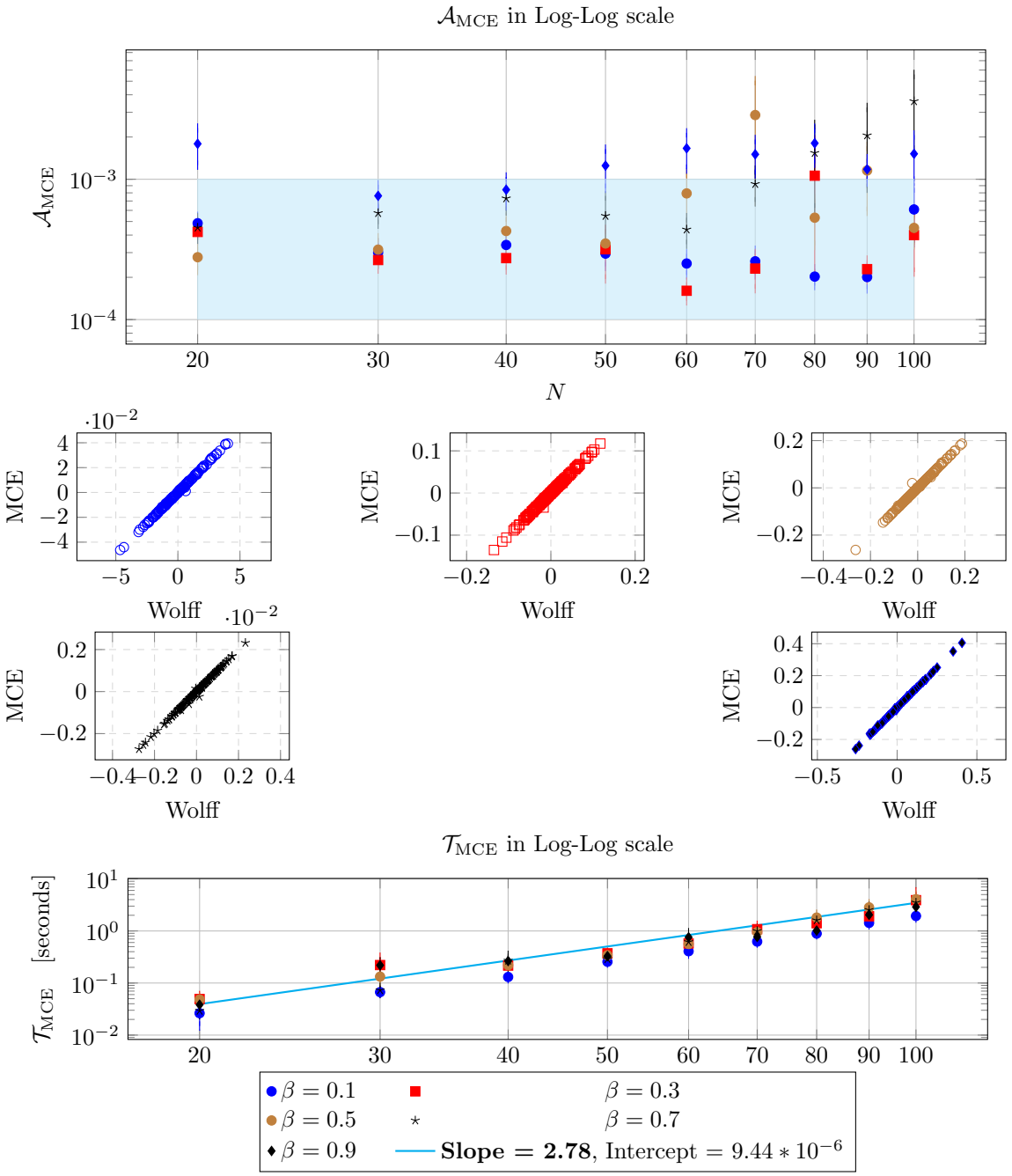

Figure 30: Input parameters: $\boldsymbol{\xi = 10^{-4}, \lambda = +\infty, \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 3, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 5 panels of MCE vs Wolff data, the entire dataset for all values of N and for all the 20 samples are presented for each value of β, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = N^{\textbf{Slope}} + \text{intercept}$ is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

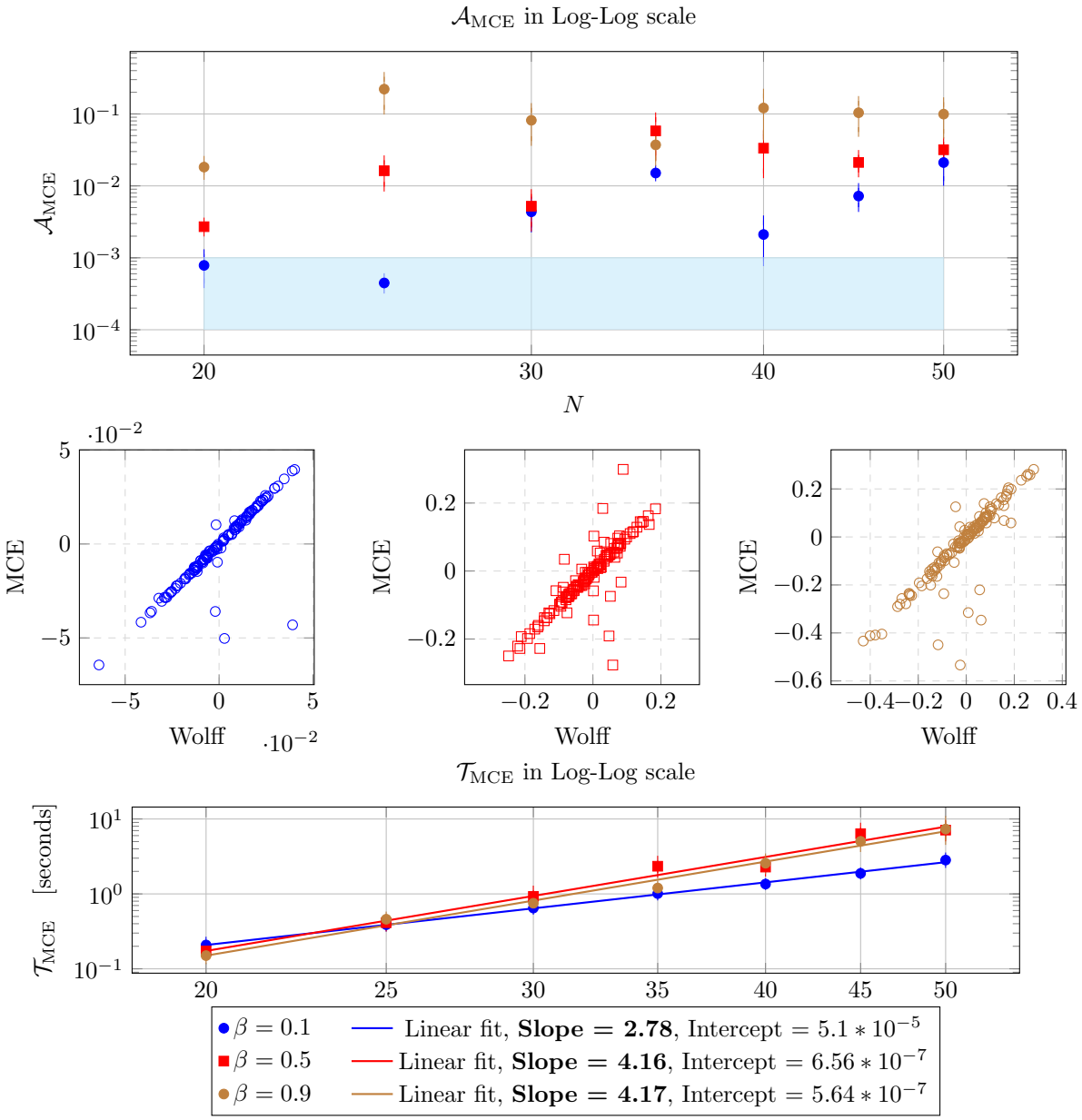

Figure 31: Input parameters: $\boldsymbol{\xi = 10^{-3}, \lambda = +\infty, \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 4, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 3 panels of MCE vs Wolff data, the entire dataset for all values of N and for all the 20 samples are presented for each value of β, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = N^{\textbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

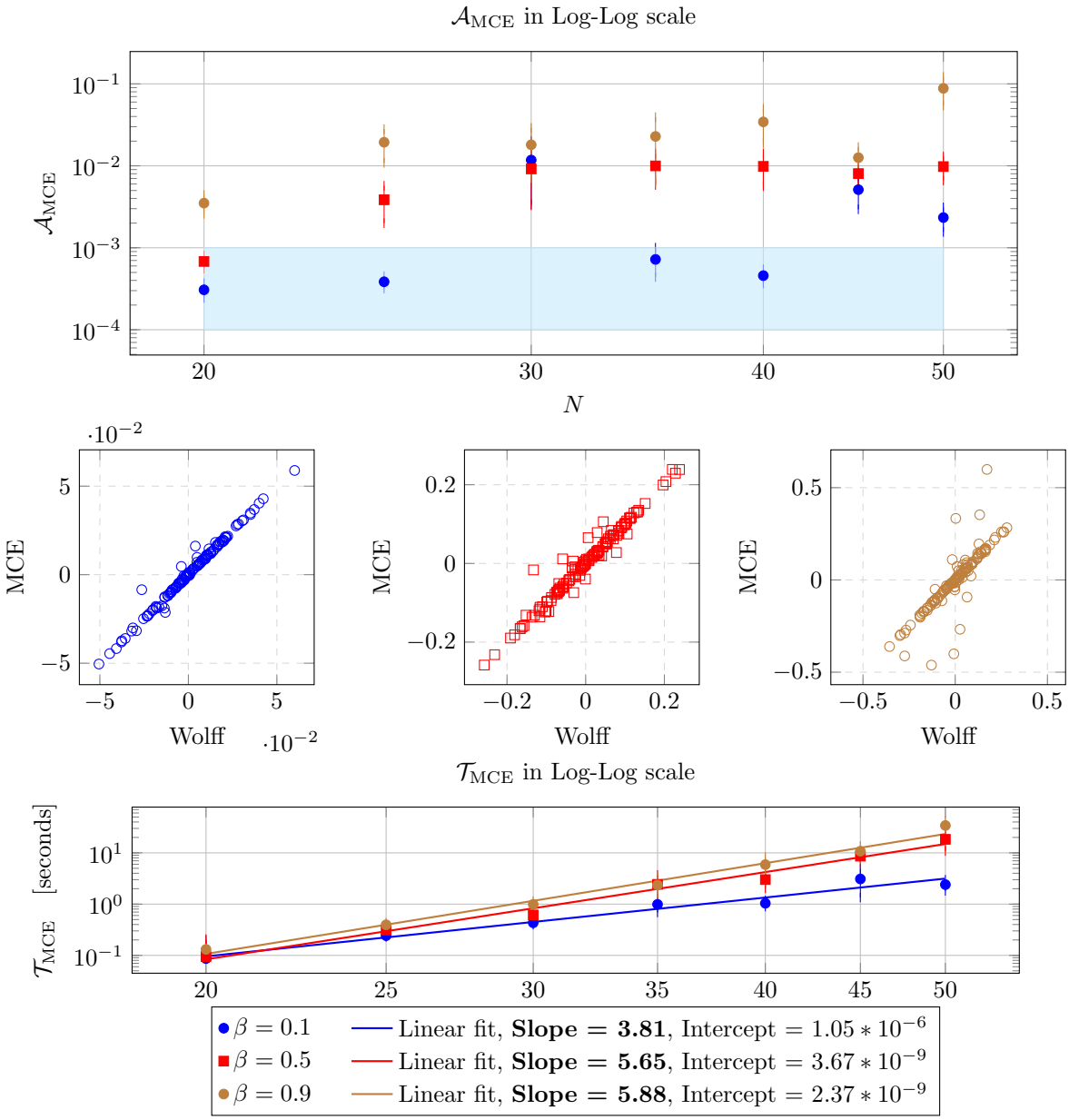

Figure 32: Input parameters: $\boldsymbol{\xi = 10^{-4}, \lambda = +\infty, \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 4, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 3 panels of MCE vs Wolff data, the entire dataset for all values of N and for all the 20 samples are presented for each value of β, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = N^{\textbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

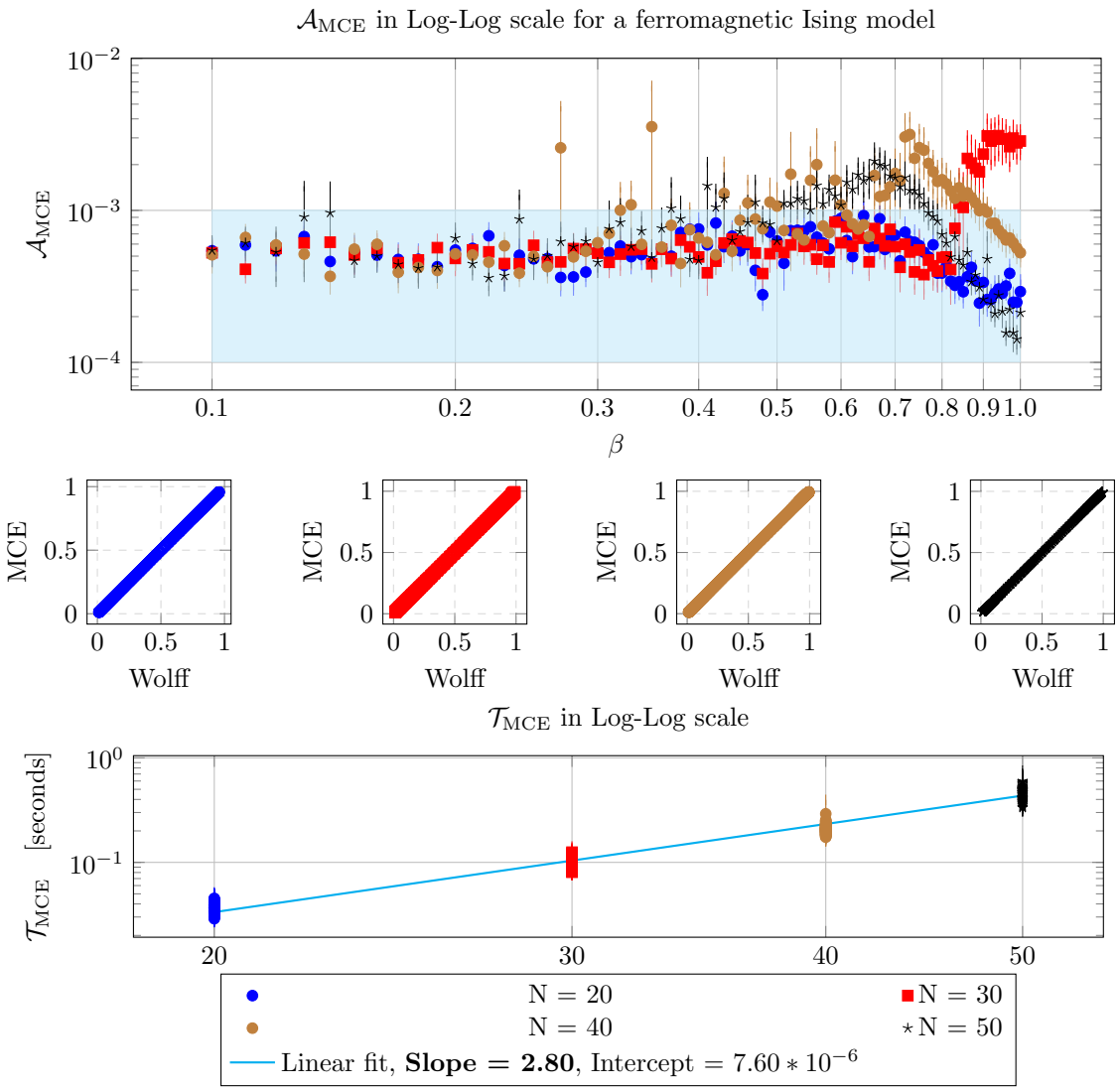

Figure 33: Input parameters: $\boldsymbol{\xi = 10^{-3}, \lambda = +\infty, \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of a ferromagnetic Ising model with N spins and *random* fields $\{h\}$ with set of couplings $\vec{J} = \vec{1}$ on a random regular graph with degree 4, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 4 panels of MCE vs Wolff data, the entire dataset for all values of N and for all the 20 samples are presented for each value of β, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = N^{\textbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

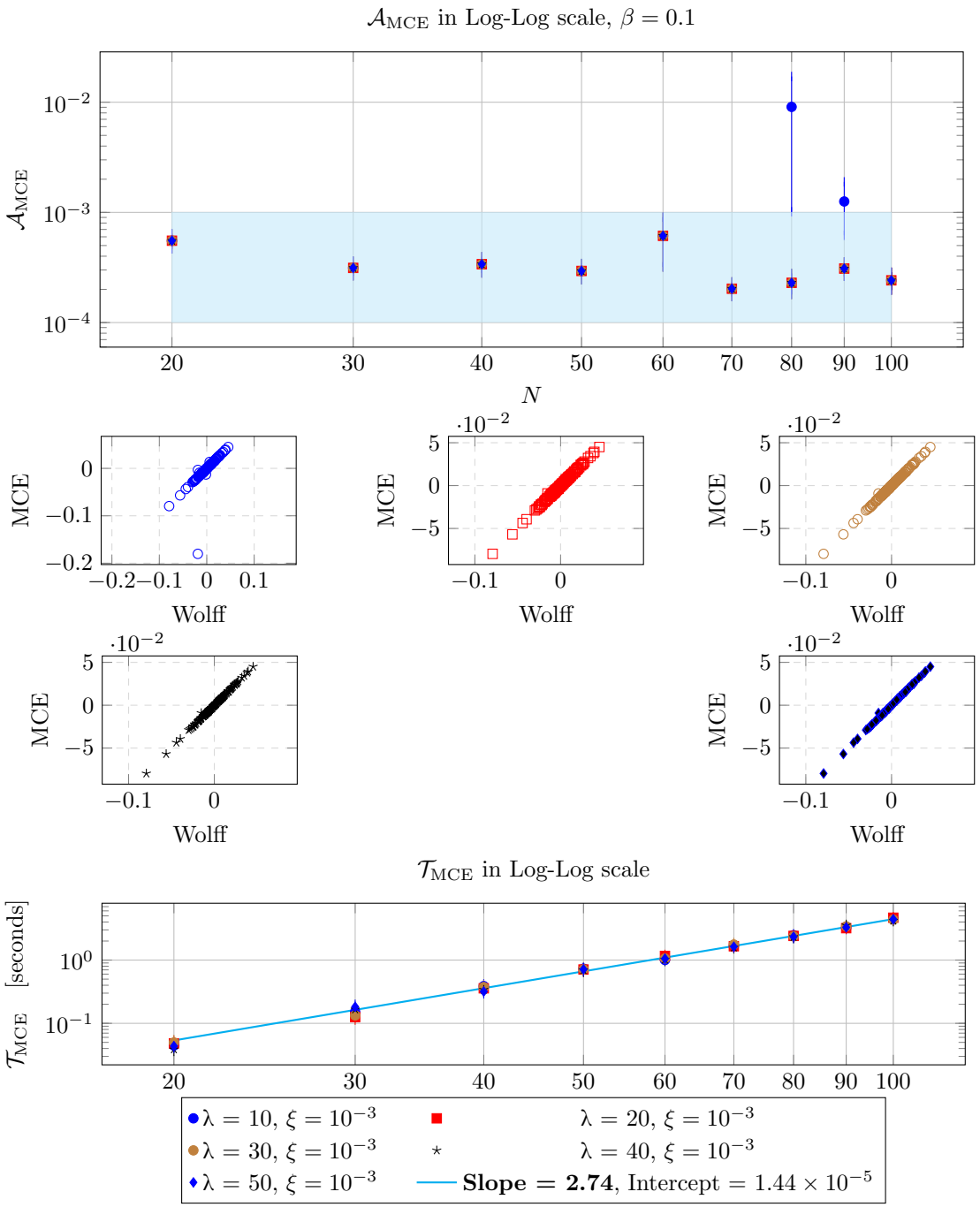

Figure 34: Input parameters: $\boldsymbol{\beta = 0.1, \xi = 10^{-3}, \lambda = [10, 20, 30, 40, 50], \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 3, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 4 panels of MCE vs Wolff data, the entire dataset for all the 20 samples and N values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = 100^{\textbf{Slope}} + \text{intercept}$ is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

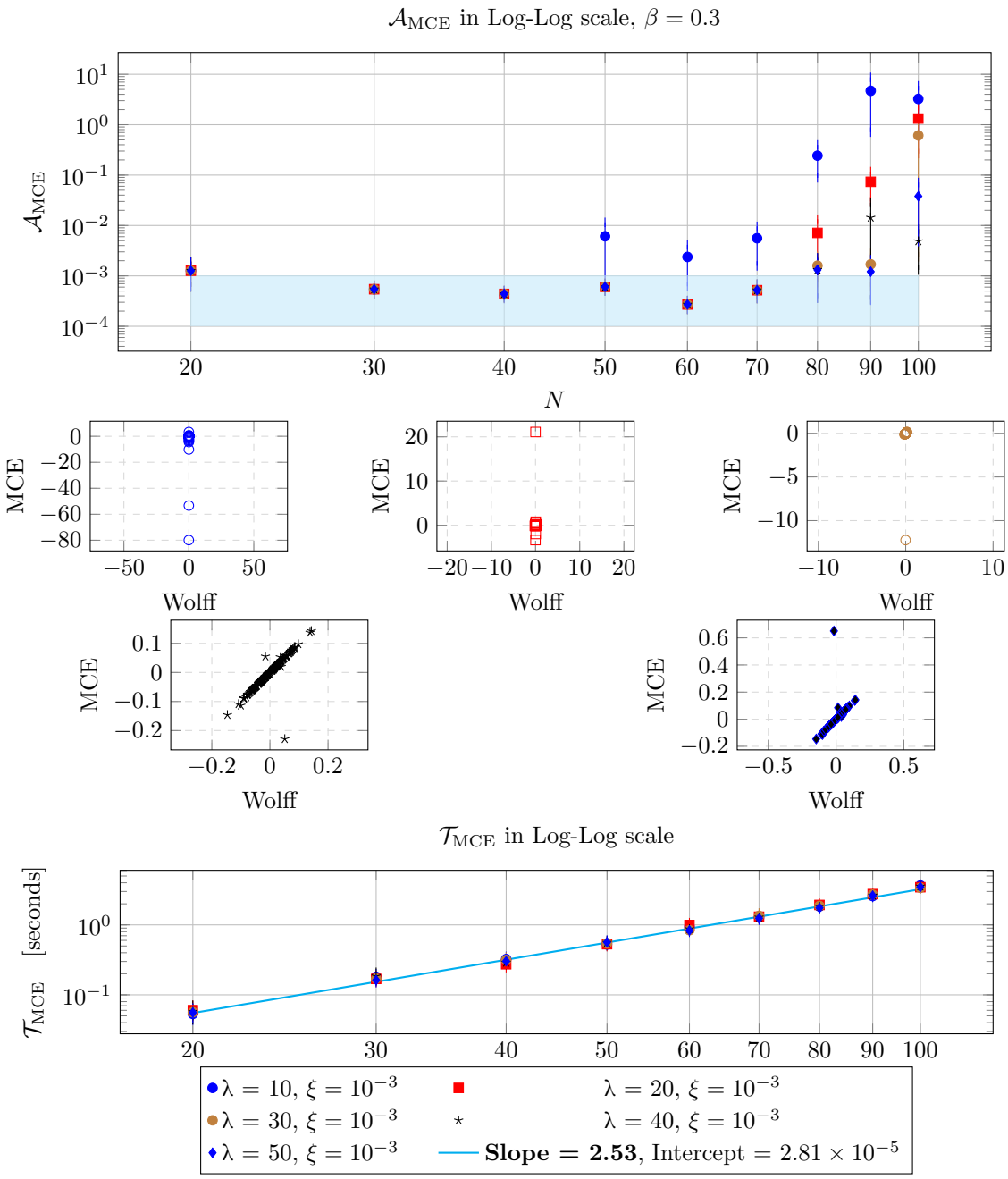

Figure 35: Input parameters: $\boldsymbol{\beta = 0.3, \xi = 10^{-3}, \lambda \in [10, 20, 30, 40, 50], \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 3, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 4 panels of MCE vs Wolff data, the entire dataset for all the 20 samples and N values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = 100^{\textbf{Slope}} + \text{intercept}$ is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

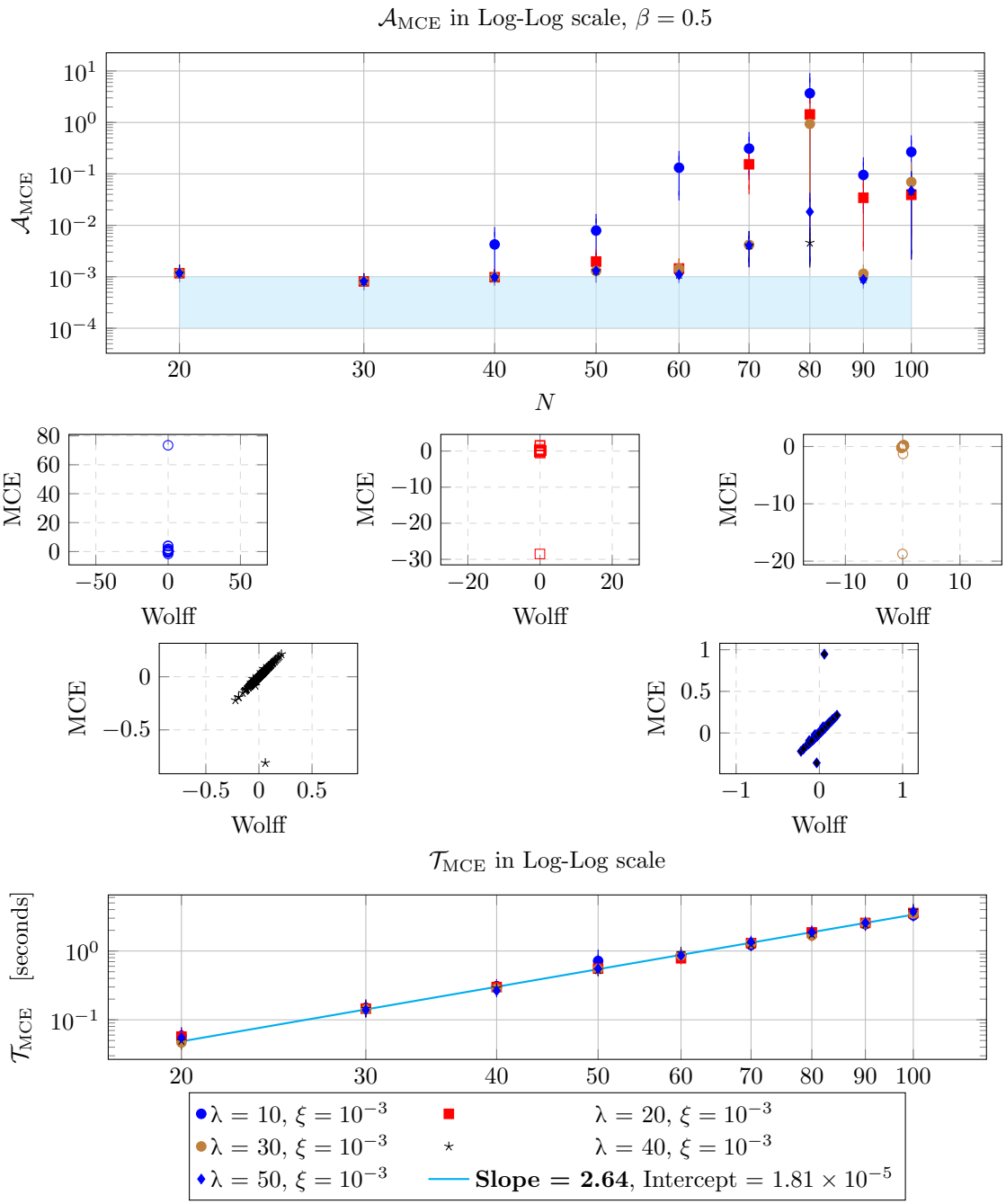

Figure 36: Input parameters: $\boldsymbol{\beta = 0.5, \xi = 10^{-3}, \lambda \in [10, 20, 30, 40, 50], \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 3, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 4 panels of MCE vs Wolff data, the entire dataset for all the 20 samples and N values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = 100^{\textbf{Slope}} + \text{intercept}$ is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

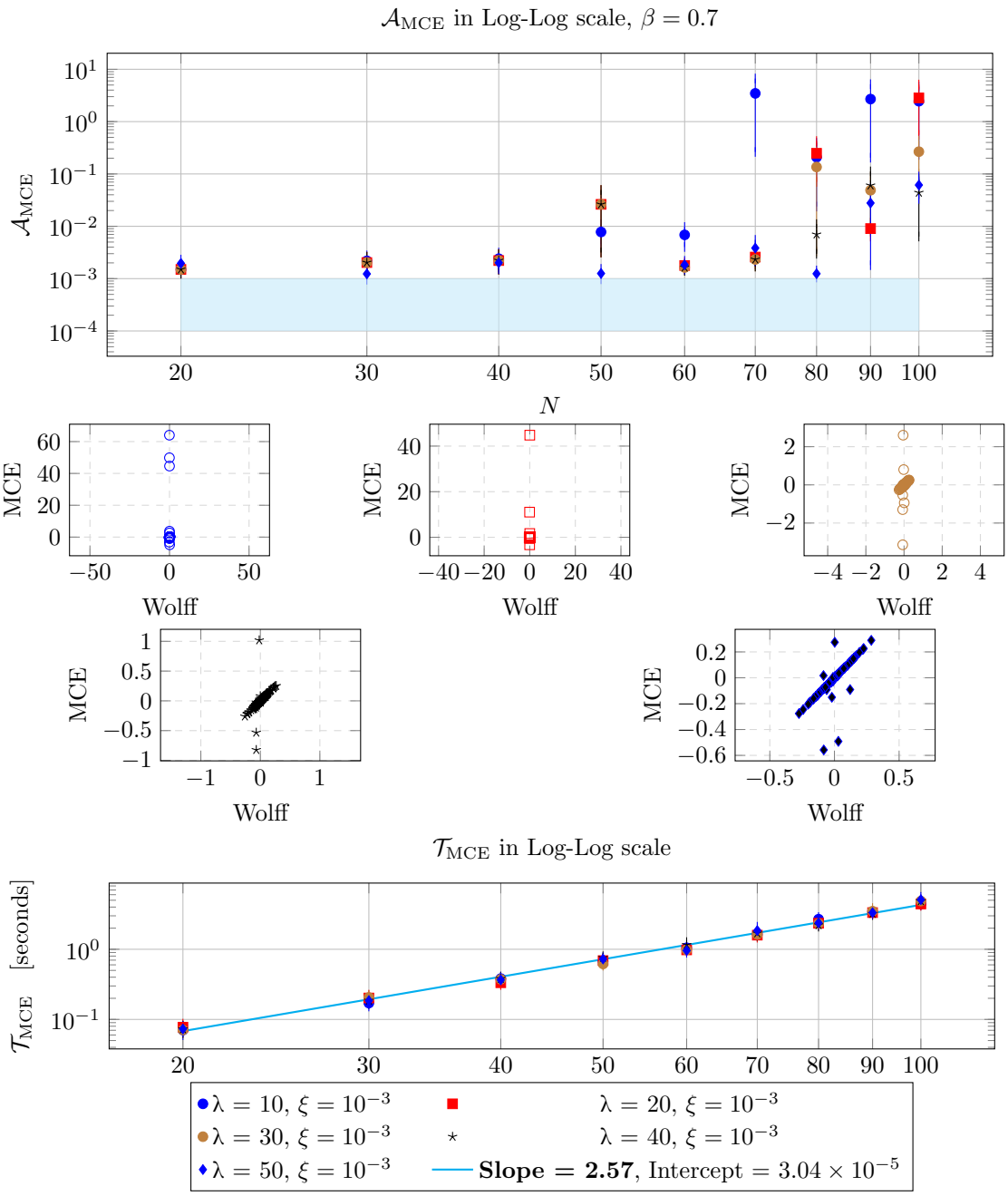

Figure 37: Input parameters: $\boldsymbol{\beta = 0.7, \xi = 10^{-3}, \lambda \in [10, 20, 30, 40, 50], \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 3, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 4 panels of MCE vs Wolff data, the entire dataset for all the 20 samples and N values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = 100^{\textbf{Slope}} + \text{intercept}$ is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

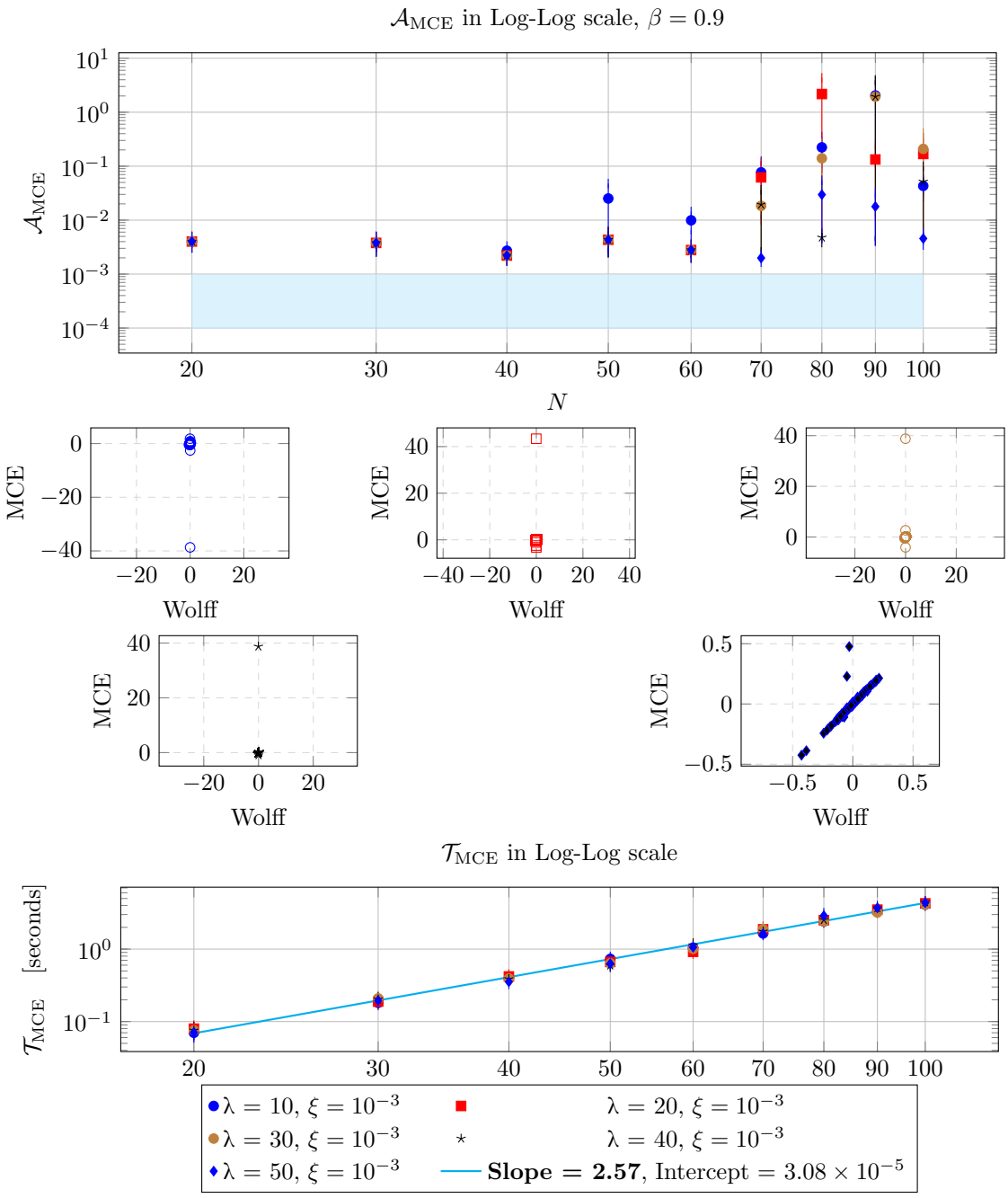

Figure 38: Input parameters: $\boldsymbol{\beta = 0.9, \xi = 10^{-3}, \lambda \in [10, 20, 30, 40, 50], \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{MCE}}$ on the calculation of the average magnetization of an Ising model with N spins and *random* parameters $\{J, h\}$ on a random regular graph with degree 3, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various β values.
Central figures: in the 4 panels of MCE vs Wolff data, the entire dataset for all the 20 samples and N values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{MCE} = 100^{\textbf{Slope}} + \text{intercept}$ is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

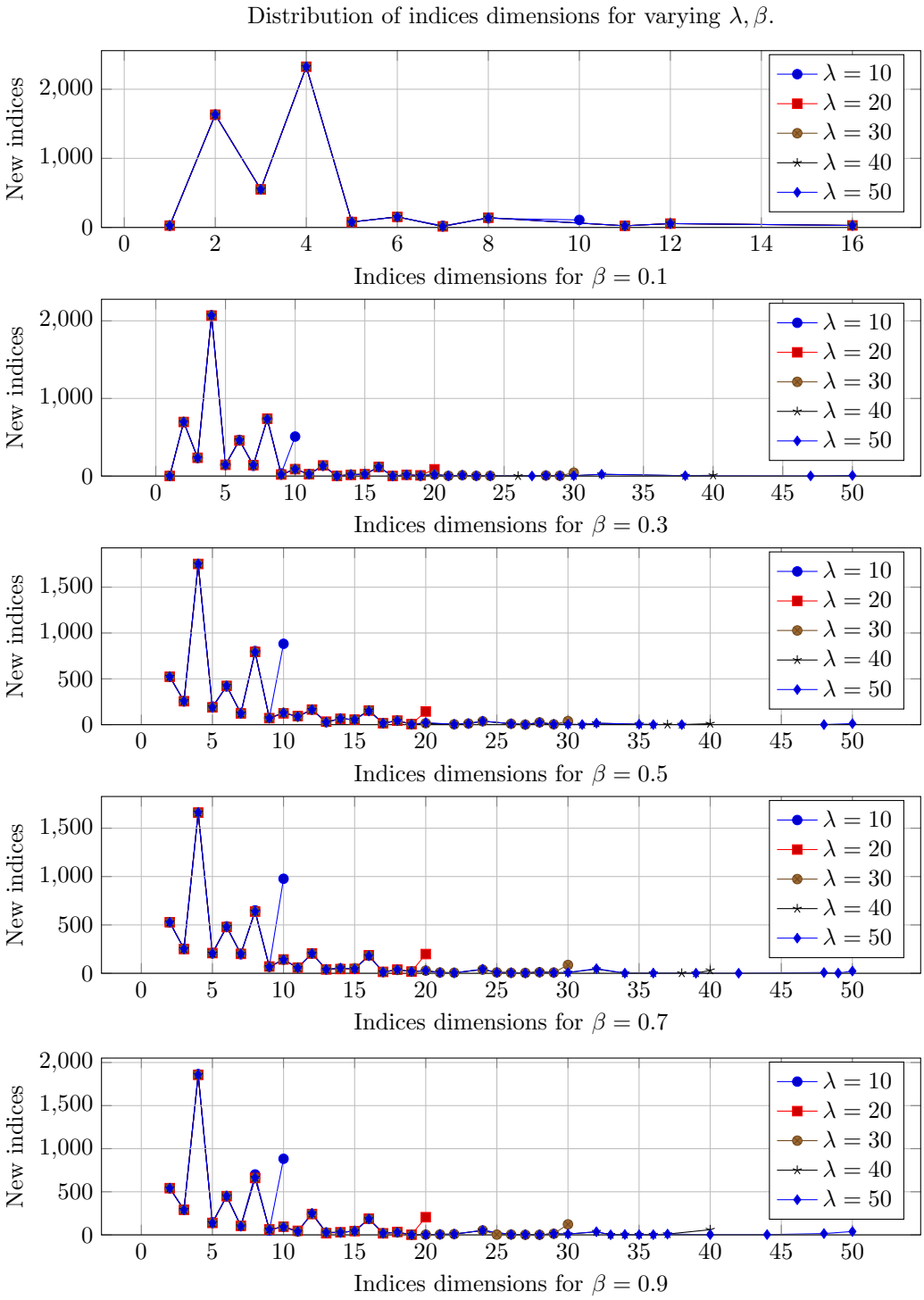

Figure 39: Different new indices dimensions introduced in the TN during the SVD-of-the-lighter-cycle's step in the MCE algorithm for simulations in figures 34, 35, 36, 37, 38 , for various β, λ and considering all the 20 samples collected together. As it can be clearly seen, despite the higher precision achievable increasing λ, the number of indices with dimension greater than 20 is very small. **Indeed what it is observed from simulations is that the precision is strongly influenced by few tensors for which a higher λ is necessary. For the most of the tensors a dimension at most equal to 10 is sufficient**.

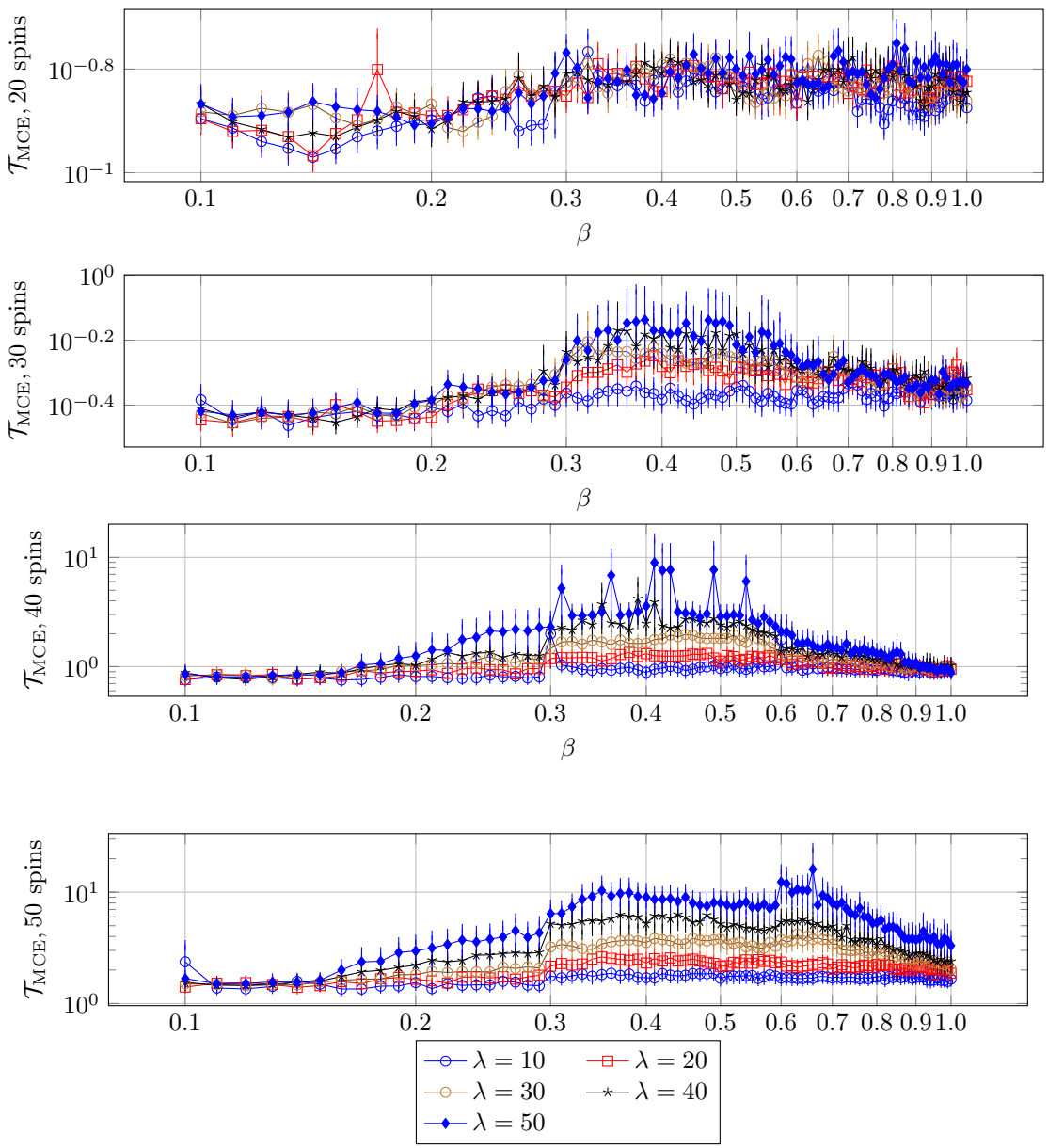

Figure 40: Temperature $T = 1/\beta$ dependance of the computational times $\mathcal{T}_{\mathrm{MCE}}$ for a ferromagnetic Ising model with zero fields, in the case of a number of spin $N \in \{20, 30, 40, 50\}$ on a regular graoh with degree 4. Errors bars are presented to depict the errors among 20 simulations. Except for some specific simulations in the third panel, i.e. for $N = 40$, for inverse temperatures in the interval $\{0.3, 0.6\}$, the computational times are slowly increasing with β after $\beta = 0.3$, and then slowly decrease after $\beta = 0.6$.

Performances and features of MCE algorithm from simulations results

We have analysed the Ising model, where χ is always equal to 2, because the couple-interaction tensors are matrices, i.e. they are already in the MPS form, while the copy tensors can be always written as product of smaller copy tensors with non-grown indices' dimensions, as we have seen in figure 6. The two parameters λ and ξ determine the properties of MCE algorithm:

1. λ selects the maximum index dimension allowed in the SVD step.

2. ξ controls the error on the representation, its role is to select a number of singular values $\leq \lambda$ such that the error remains below the given cutoff ξ: if the SVD proposes e.g. λ singular values to be kept in the representation of the matrix S, see figure 7, then the parameter ξ allows to control if λ singular values are necessary to obtain a sum of the squares of the discarded eigenvalues $\leq \xi$. If the precision is too high with respect to ξ the lowest singular values are removed such that **the remaining number of singular values is the lowest possible to achieve the desired precision.**

The temperature $T = 1/\beta$ has a unique influence, independent on the choice of the paramters λ, ξ, χ: for higher temperatures the contractions are easier and require computational times as depicted in figure 40. This is due to the reduced complexity in the tensors' form, indeed higher temperatures generates tensors's entries which are all small and similar between them. Indeed, recall that the sets of couplings and fields are extracted from a uniform distribution. Let's analyse the various cases:

1. **λ not given, ξ given:** λ is not given so the tensors in the SVD step can be generated with an arbitrary number of singular values, the only constraint is that the errors are $\leq \xi$. As it can be seen from figures 29, 30, 31, 32 a higher complexity of the underlying network, i.e. the degree of the regular graph, implies a higher ξ to achieve a better precision. As a consequence the computational times needed grow accordingly because bigger tensors are created in the SVD step and higher memory allocations are needed. **The computational times are always polynomial in the number of nodes** N and depends on the complexity of the underlying network and on the cutoff ξ chosen. The complexity of the network is controlled also by the tensors' entries, indeed in figure 33 the regular graph has degree 4, but the statistical model is a ferromagnetic Ising model with couplings all equal to 1 and random fields; this easier structure of the tensors in the network allows for better approximations with a cutoff $\xi = 10^{-3}$. The number of nodes N in figures 31, 32 is at most 50 because the number of memory allocations needed without a fixed value for λ is too high, a common computer is not able to perform the contraction for higher complexity of the underlying network. The precision is high because we are able to pay whatever cost in terms of singular values kept in the SVD step.

2. **Both λ and ξ given:** as it can be seen in figures 34, 35, 36, 37, 38 a choice $\lambda = 50, \xi = 10^{-3}$ ensures mean errors at most on the first decimal unit and higher precision can be reached with higher values of λ, still bounded. The errors are generated from at most 3 of the 180 simulations. Indeed errors are generated from a low number of tensors which are not approximated well and propagate errors during the contraction, which seems to to suggest that specific rare tensor structures are very difficult to approximate with the SVD method. From figure 39 it can be seen that few tensors use the maximum index dimension allowed due to the influence of ξ: some of these tensors need a high number of singular values to be approximated with the same precision of the other tensors using a number of singular values $< \lambda$. Indeed if the number of singular values kept in the approximation is lower than the maximum allowed λ, we are sure that the precision ξ is reached while we cannot make the same reasoning if the SVD-index dimension is the higher allowed.

Final analysis of MCE algorithm

The better choice of input parameters is a balance between a finite value of ξ and a finite value of λ which has to be chosen according to the specific physical symmetries of the model considered and to the graphical properties of the underlying network. In the simulations we have seen, the worst case of random Ising parameters and random regular graphs is considered to obtain the worst-case performances of MCE algorithm, at least for the Ising model.

In the next chapter we will se the performances of RRC algorithm, focusing on the scaling of the computational times needed to know if the pipeline transformation can be safely applied and in what cases it leads to better performances with respect to the MCE algorithm ones.
Independently of what we will see, the MCE algorithm gives its best if specific contraction heuristic are designed exploiting the physical symmetries and graphical properties of the network. The following analysis will give the differences in the performances of these two algorithms in the worst case, but this do not give information on their performances for all possible probabilistic graphical models. In the end we will see what is the better choice if we do not know anything about the properties of symmetries of the model we are analysing.

12 RRC algorithm absolute errors $\mathcal{A}_{\mathbf{RRC}}$ and execution times $\mathcal{T}_{\mathbf{RRC}}$

Let's analyze the performances of RRC algorithm considering the absolute errors $\mathcal{A}_{\mathrm{RRC}}$ and the execution times $\mathcal{T}_{\mathrm{RRC}}$ for a square lattice for varying grid size $\in \{10, 20, \ldots, 100\}$ and inverse temperature β, comparing the average magnetizations with Wolff algorithm, the latter executed with 1.000.000 samples. The results are presented in Figures 30 and 31.
In figure:

1. Figure 41, 42: $\boldsymbol{\beta = 0.1, \xi = +\infty, \lambda \in [5, 10, 15, 20]}$ with computational times needed.
2. Figure 44: $\boldsymbol{\beta = 0.1, \xi = +\infty, \lambda \in [5, 10, 15, 20, 25, 30, 35, 40]}$ with computational times needed.
3. Figure 45: distribution of dimension of indices in performing simulations in figures 41, 42, 44.
4. Figure 46: $\boldsymbol{\beta = 0.1, \xi = 10^{-3}, \lambda \in [5, 10, 15, 20]}$ with computational times needed.
5. Figure 47: $\boldsymbol{\beta = 0.5, \xi = 10^{-3}, \lambda \in [5, 10, 15, 20]}$ with computational times needed.
6. Figure 49: $\boldsymbol{\beta = 0.9, \xi = 10^{-3}, \lambda \in [5, 10, 15, 20]}$ with computational times needed.
7. Figure 50: distribution of dimension of indices in performing simulations in figures 46, 47, 49.

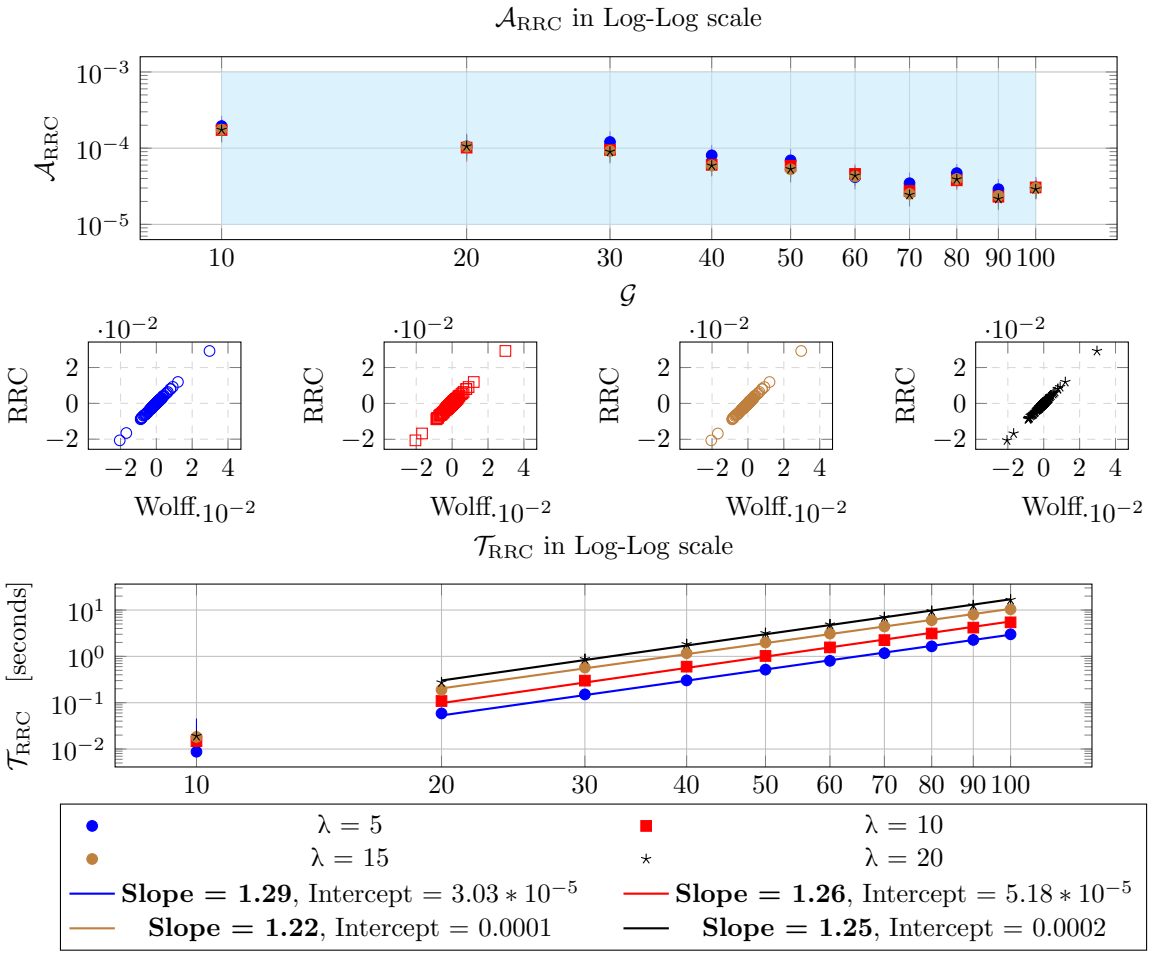

Figure 41: Input parameters: $\boldsymbol{\beta = 0.1, \xi = +\infty, \lambda \in [5, 10, 15, 20]}$.
Top figure: absolute errors $\mathcal{A}_{\text{RRC}}$ on the calculation of the average magnetization of an Ising model with $N = \mathcal{G}^2$ spins and *random* parameters $\{J, h\}$ on a square lattice with grid size $\mathcal{G}$, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various $\mathcal{G}$ values.
Central figures: in the 4 panels of RRC vs Wolff data, the entire dataset for all the 20 samples and $\mathcal{G}$ values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{RRC} = N^{\textbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

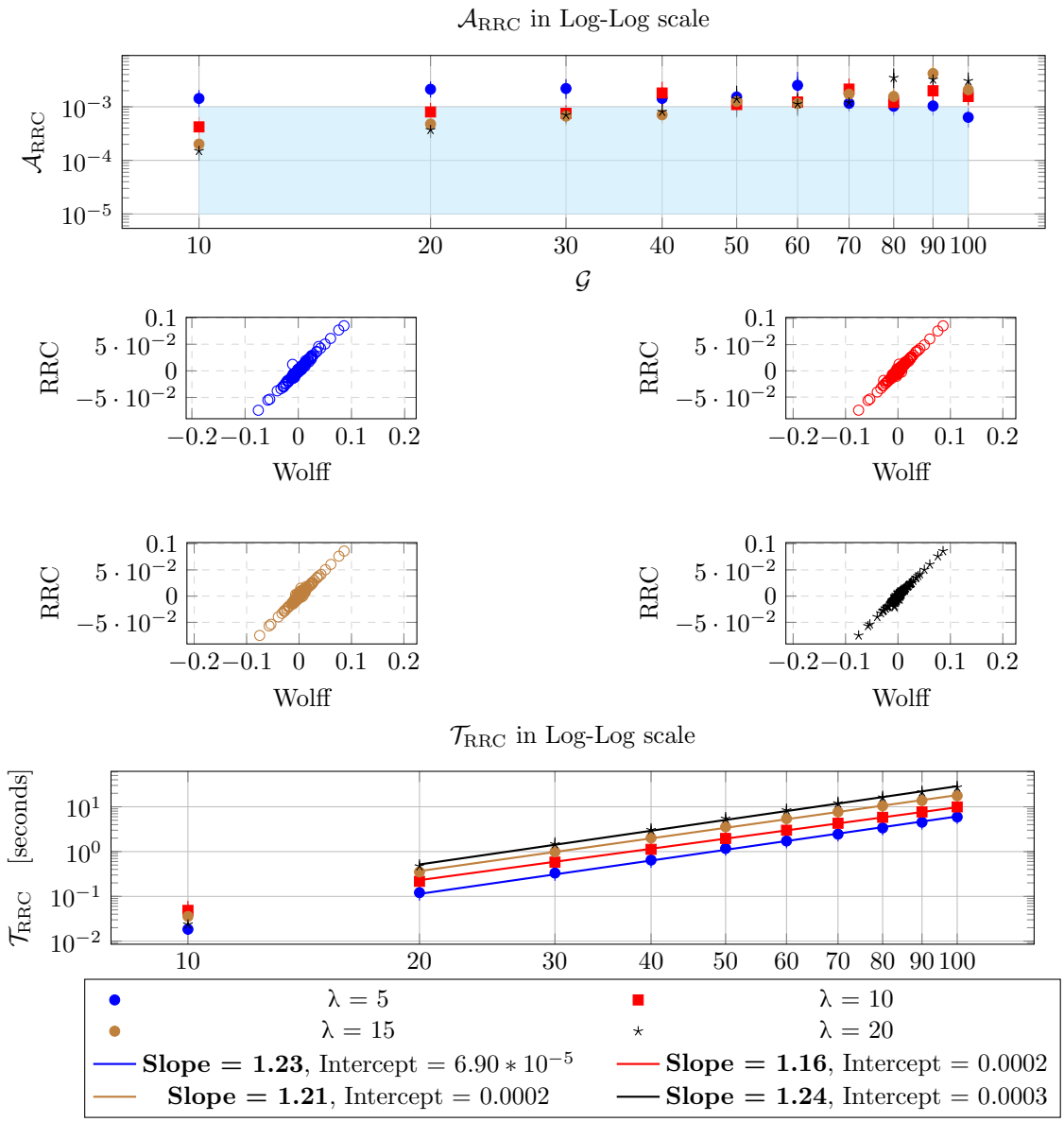

Figure 42: Input parameters: $\boldsymbol{\beta = 0.5, \xi = +\infty, \lambda \in [5, 10, 15, 20]}$.
Top figure: absolute errors $\mathcal{A}_{\mathrm{RRC}}$ on the calculation of the average magnetization of an Ising model with $N = \mathcal{G}^2$ spins and *random* parameters $\{J, h\}$ on a square lattice with grid size $\mathcal{G}$, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various $\mathcal{G}$ values.
Central figures: in the 4 panels of RRC vs Wolff data, the entire dataset for all the 20 samples and $\mathcal{G}$ values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{RRC} = N^{\mathbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

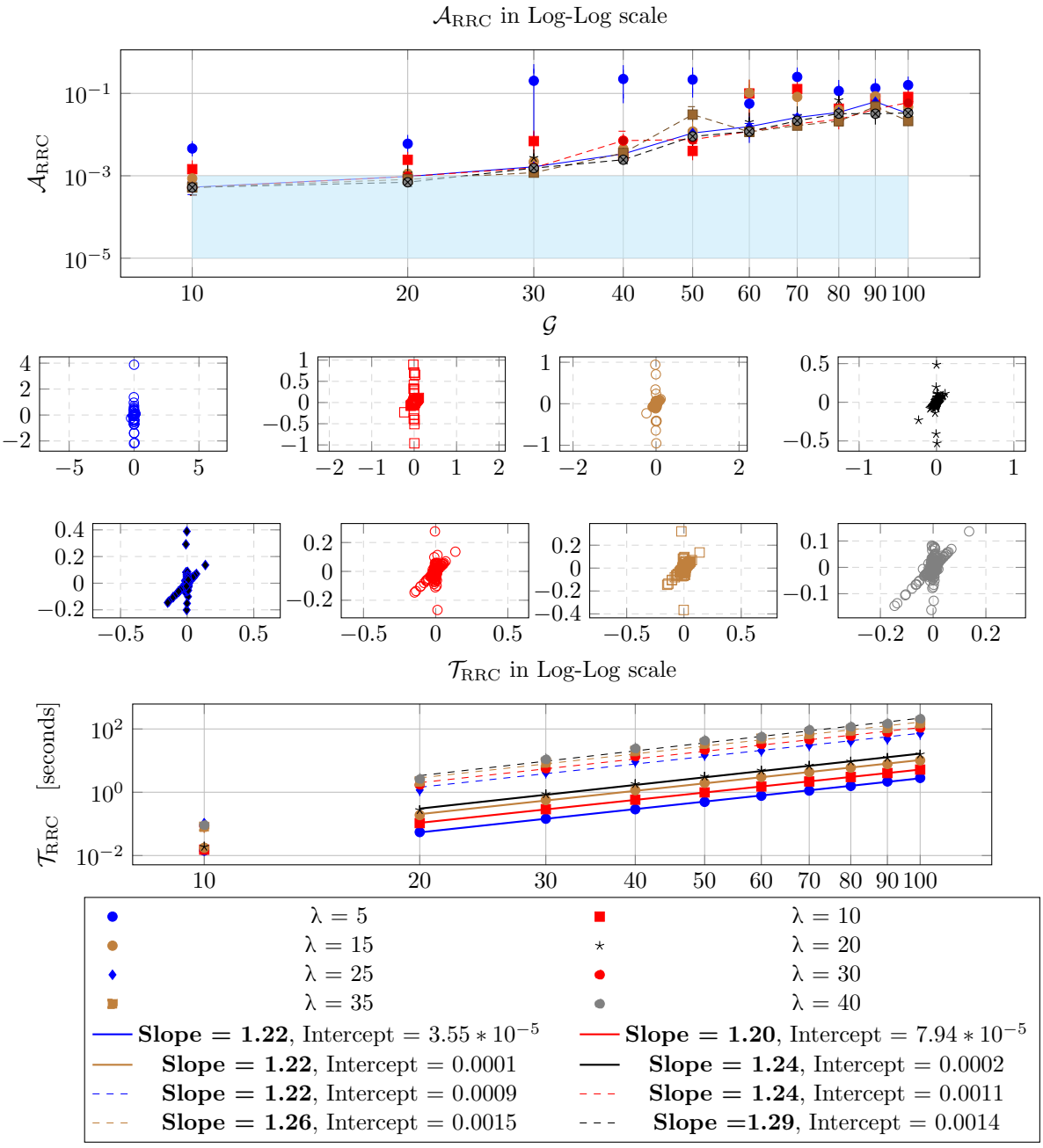

Figure 44: Input parameters: $\boldsymbol{\beta = 0.9, \xi = +\infty, \lambda \in [5, 10, 15, 20]}$.
Top figure: absolute errors $\mathcal{A}_{\text{RRC}}$ on the calculation of the average magnetization of an Ising model with $N = \mathcal{G}^2$ spins and *random* parameters $\{J, h\}$ on a square lattice with grid size $\mathcal{G}$, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various $\mathcal{G}$ values.
Central figures: in the 4 panels of RRC vs Wolff data, the entire dataset for all the 20 samples and $\mathcal{G}$ values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{RRC} = N^{\textbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

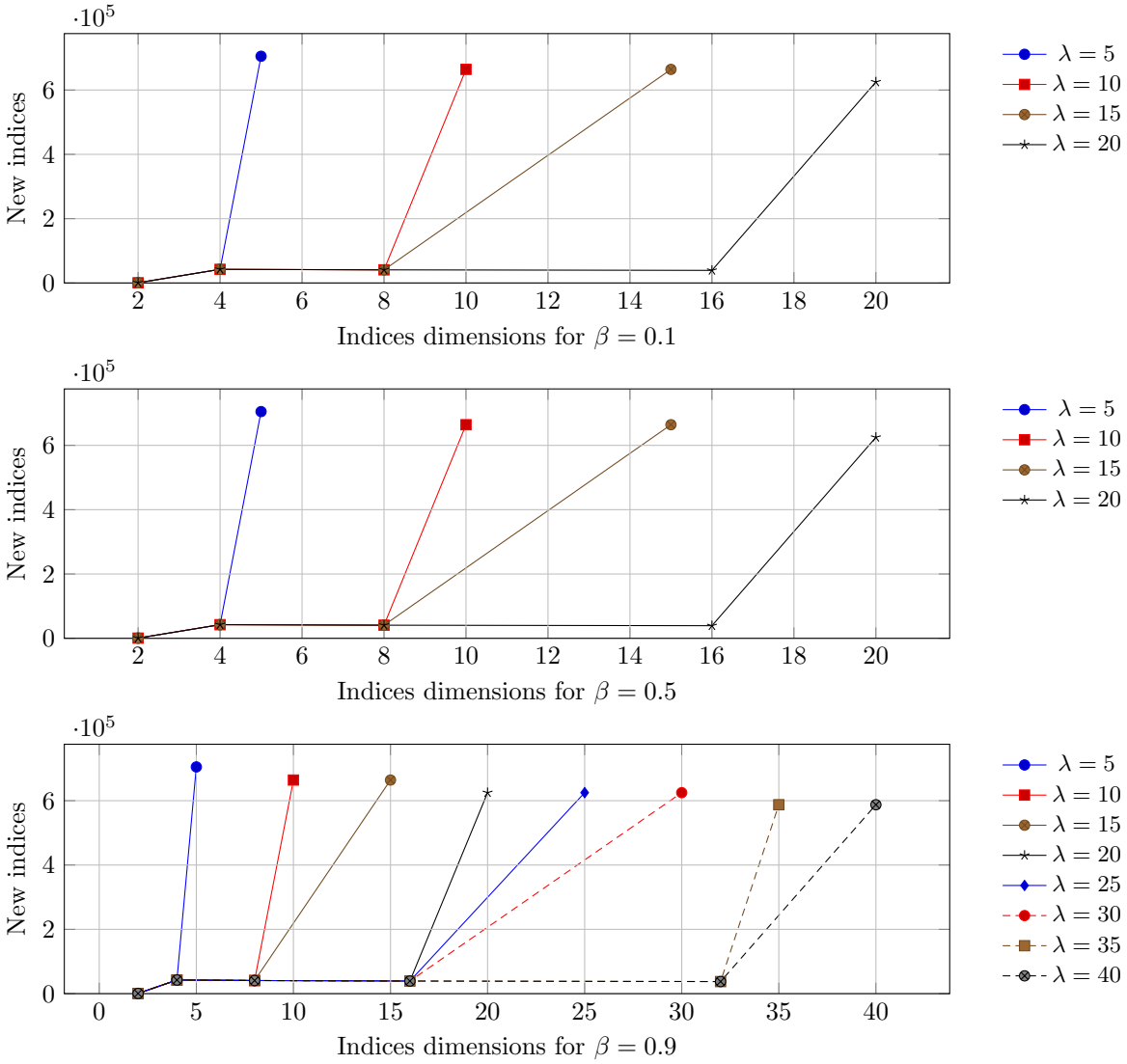

Figure 45: Different new indices dimensions introduced in the TN during the SVD-step in the RRC algorithm, for various β, λ and considering all the 20 samples collected together. From simulations for most of tensors, RRC algorithms choose the highest λ available. This behaviour is predictable, because the λ values allowed are small, i.e. they do not exceed 20.

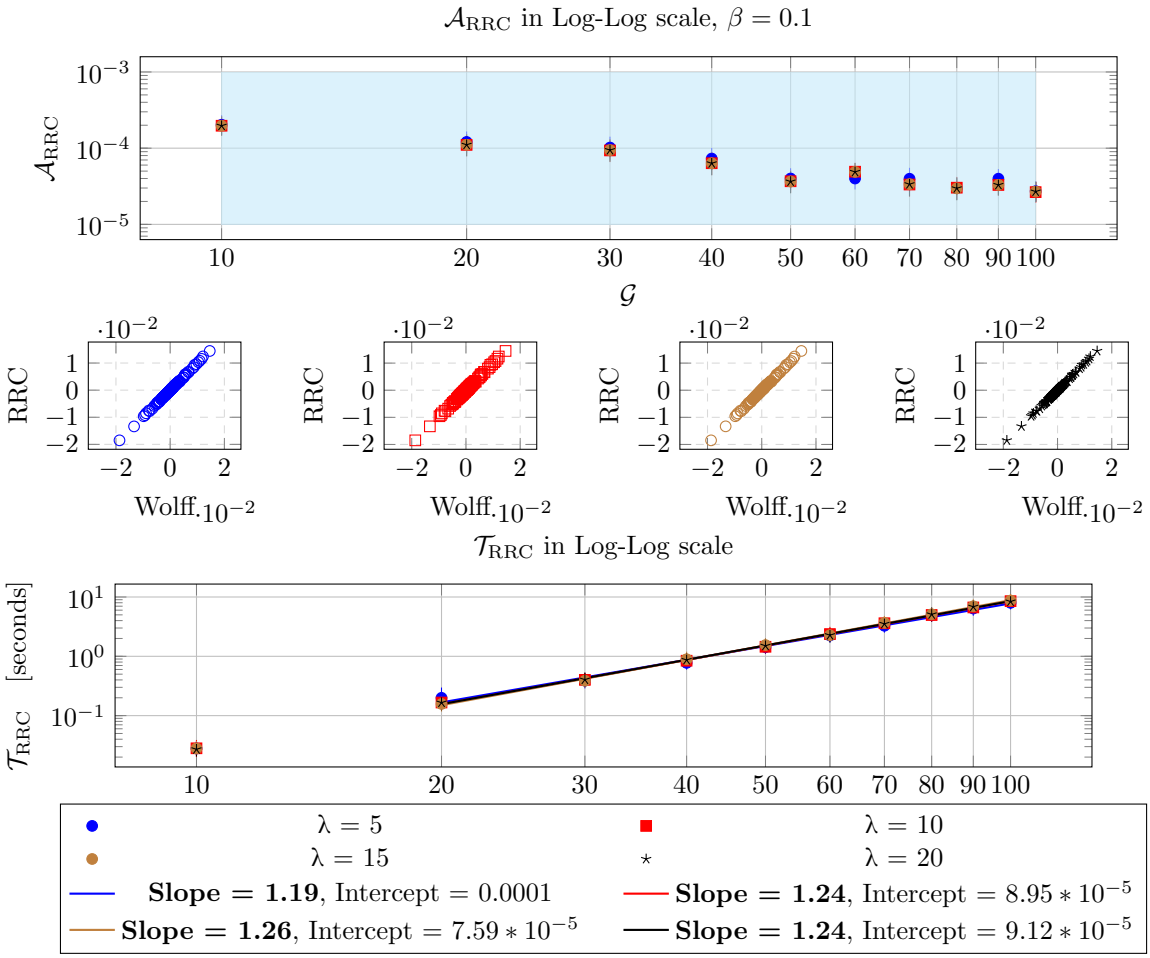

Figure 46: Input parameters: $\boldsymbol{\beta = 0.1, \xi = 10^{-3}, \lambda \in [5, 10, 15, 20], \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{RRC}}$ on the calculation of the average magnetization of an Ising model with $N = \mathcal{G}^2$ spins and *random* parameters $\{J, h\}$ on a square lattice with grid size $\mathcal{G}$, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various $\mathcal{G}$ values.
Central figures: in the 4 panels of RRC vs Wolff data, the entire dataset for all the 20 samples and $\mathcal{G}$ values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{RRC} = N^{\textbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

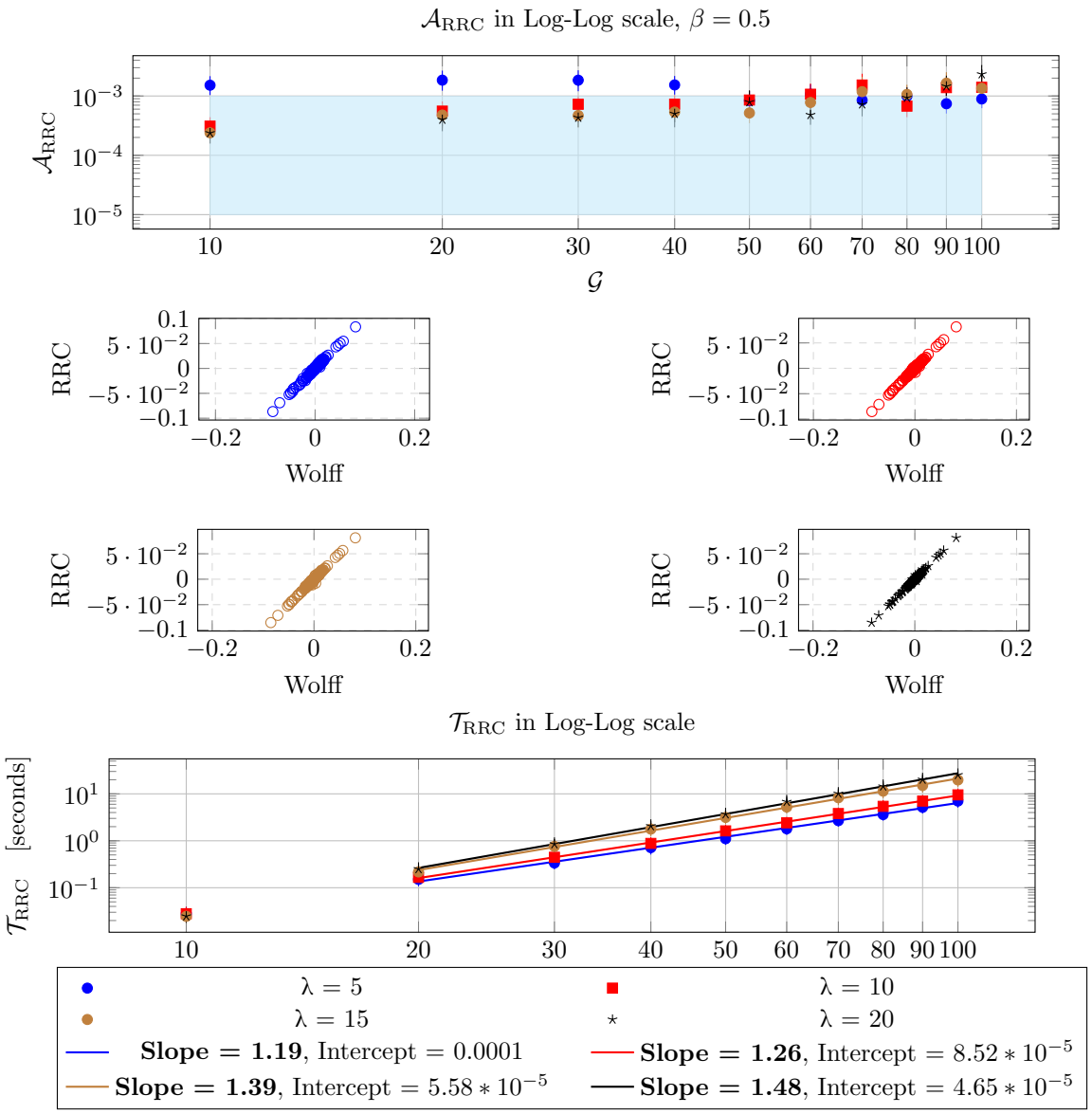

Figure 47: Input parameters: $\boldsymbol{\beta = 0.5, \xi = 10^{-3}, \lambda \in [5, 10, 15, 20], \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{RRC}}$ on the calculation of the average magnetization of an Ising model with $N = \mathcal{G}^2$ spins and *random* parameters $\{J, h\}$ on a square lattice with grid size $\mathcal{G}$, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various $\mathcal{G}$ values.
Central figures: in the 4 panels of RRC vs Wolff data, the entire dataset for all the 20 samples and $\mathcal{G}$ values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{RRC} = N^{\textbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

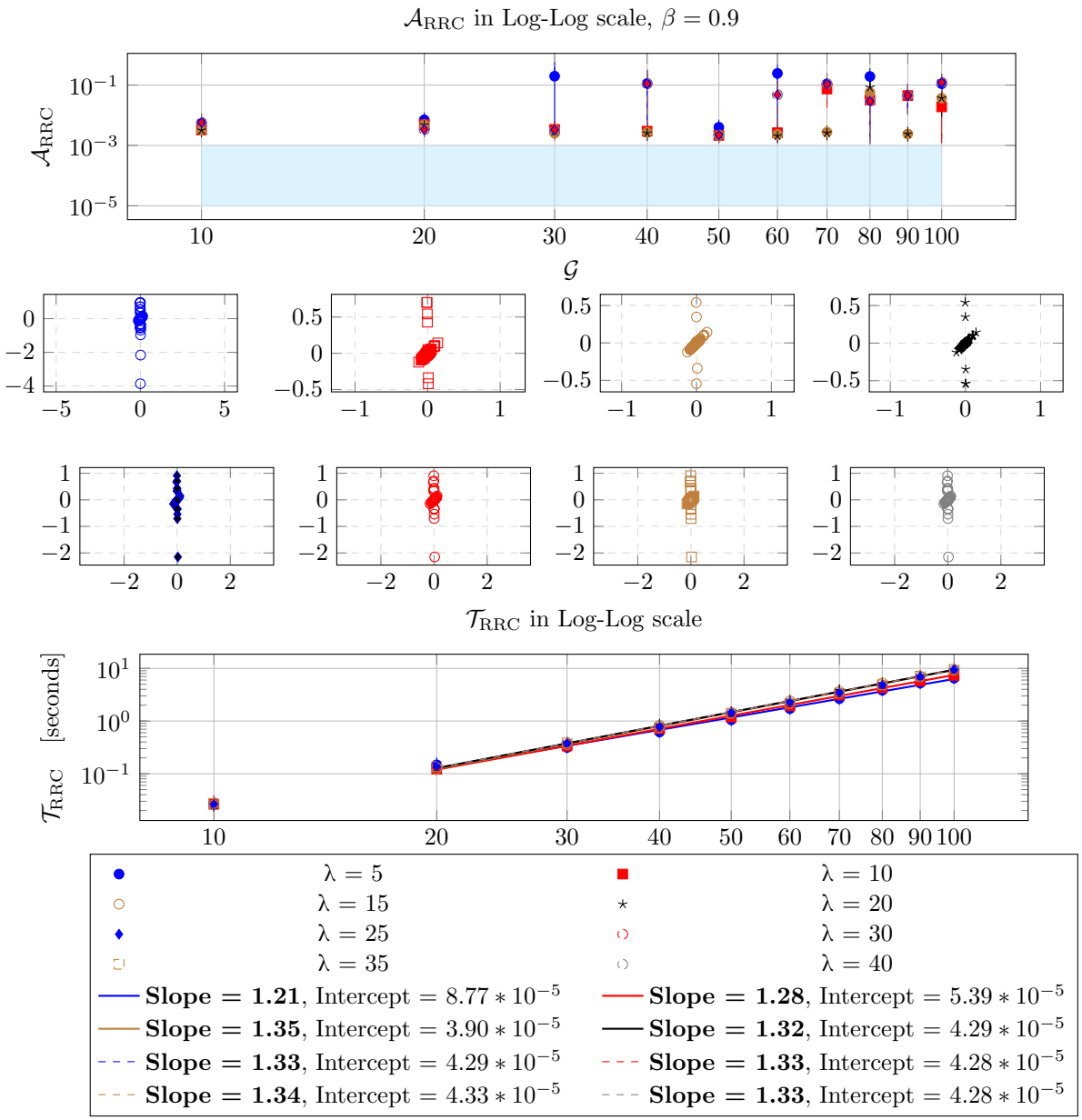

Figure 49: Input parameters: $\boldsymbol{\beta = 0.9, \xi = 10^{-3}, \lambda \in [5, 10, 15, 20], \chi = 2}$.
Top figure: absolute errors $\mathcal{A}_{\text{RRC}}$ on the calculation of the average magnetization of an Ising model with $N = \mathcal{G}^2$ spins and *random* parameters $\{J, h\}$ on a square lattice with grid size $\mathcal{G}$, using 20 samples and comparing the results with Wolff algorithm's average magnetization, for various $\mathcal{G}$ values.
Central figures: in the 4 panels of RRC vs Wolff data, the entire dataset for all the 20 samples and $\mathcal{G}$ values are presented for each value of λ, following the same legend.
Bottom figure: the linear fitting (linear in Log-Log scale) of $\mathcal{T}_{RRC} = N^{\textbf{Slope}}$ + intercept is done using the method of Ordinary Least Squares for the 20 samples of Ising model average magnetization calculation.

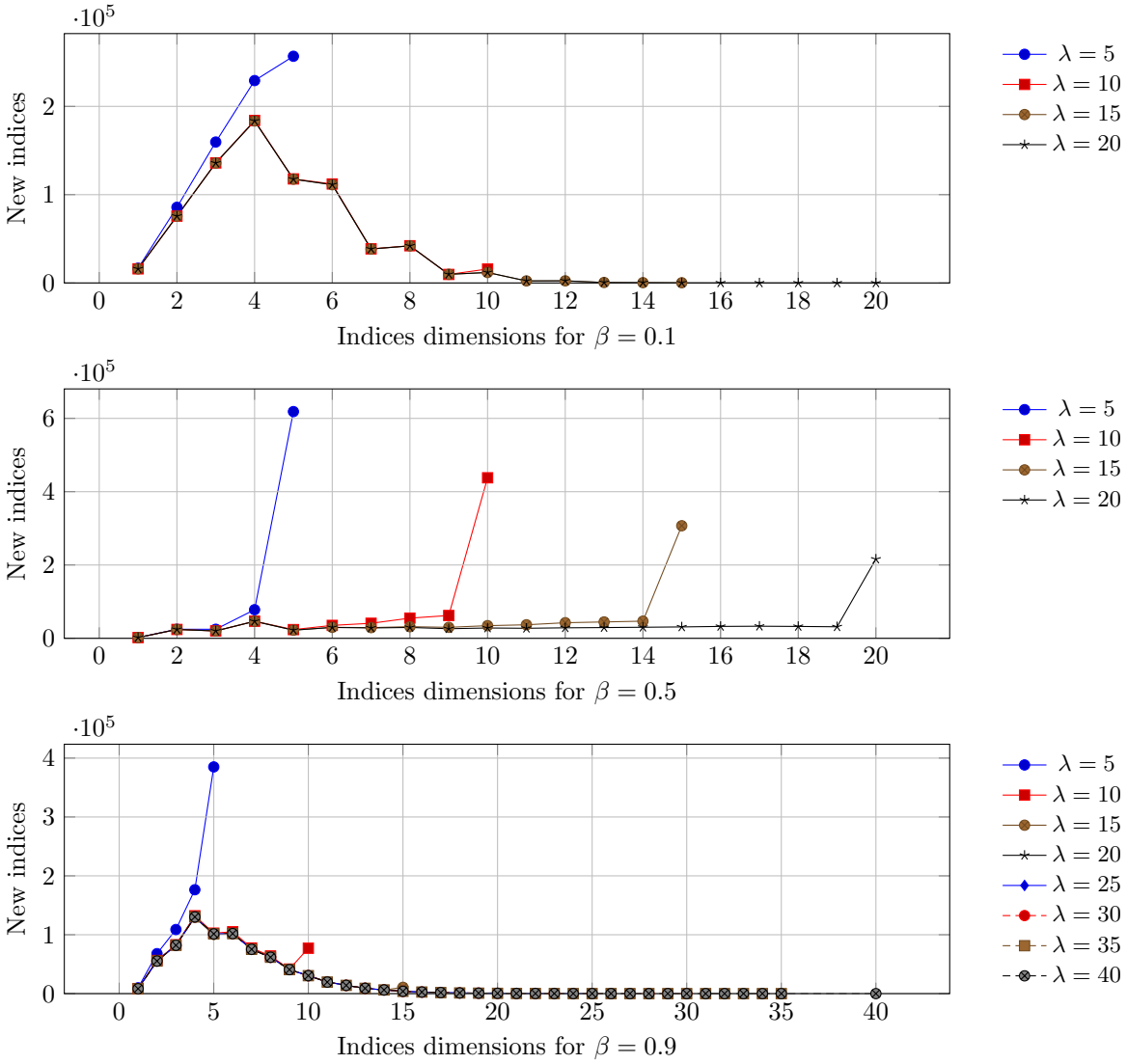

Figure 50: Different new indices dimensions introduced in the TN during the SVD-step in the RRC algorithm, for various β, λ and considering all the 20 samples collected together. From simulations for most of tensors, RRC algorithms choose the highest λ available. This behaviour is predictable, because the λ values allowed are small, i.e. they do not exceed 20.

Final considerations for RRC algorithm

Let's analyse the simulations of RRC algorithm:

1. **λ given, ξ not given:** the value needed for λ to obtain errors on the second decimal unit is 20, for $\lambda > 20$ the errors do not improve, in particular considering low temperatures. The computational times remain polynomial and bounded to $N^{1.50}$ at most in the simulations I've performed. As previously analysed, the absence of a finite value for ξ generates tensors with indices' dimensions which are most of the times equal to λ, as depicted in figure 45.

2. **Both λ and ξ given:** the errors and computational times remain almost the same as in the case of a non-finite ξ but the difference is presented in figure 50: the indices dimension are lower than the non-finite ξ case, so RRC algorithm prefers to work with crude approximations from the beginning and generates errors and computational times similar to the case of non-finite ξ, using a lower number of memory allocations. Also in this case, as it can be seen from simulations, when both ξ and λ are given, $\lambda = 20$ is sufficient and higher values of λ are not needed[a]. Indeed values greater than $\lambda = 20$ do not introduce improvements, as depicted in figures 46, 47, 49 .

[a]Clearly with $\lambda = +\infty$ the errors are equal to 0, but here we are looking for good balances between errors and computational resources.

Now that we have seen both algorithms' performances we can say

RRC algorithm is convenient to contract whatever TN through the pipeline transformation IF the underlying network and tensors' structures have no useful symmetries to exploit, so in absence of useful information. In a general case, it is fundamental to know symmetries of the probabilistic graphical model under examination, so that a specific heuristic can be developed, making clear the choice between using MCE algorithm or RRC algorithm, or in order to develop new contraction heuristics for both of them.

13 Biophysics application: the case of Boltzmann learning applied to the DCA method to infer native contacts in proteins

The direct coupling analysis (DCA) method for determining the native contacts of proteins through multiple sequence alignment (MSA) of homologous proteins relies on evolutionary conservation and statistical correlations. Native contacts, the specific interactions between amino acid that stabilize the three-dimensional structure of a protein, are fundamental for understanding protein function and the impact of mutations.
In the oncological context, mutations in oncogenes and tumor suppressor genes disrupt native contacts, altering protein function and contributing to carcinogenesis. Identifying native contacts allows for the distinction between driver mutations, which promote tumor growth, and passenger mutations, which are neutral. This distinction is used for the development of targeted therapies, e.g. tyrosine kinase inhibitors like Imatinib are designed to specifically bind to the mutated forms of kinase proteins, inhibiting their oncogenic activity [10].
Understanding native contacts also helps in predicting and overcoming drug resistance, providing insights into the molecular mechanisms underlying tumor growth and metastasis.
Strategies to counteract drug resistance can be developed by identifying how mutations affect protein structure and functions.
Starting from an MSA of a class of homologous proteins of interest, a statistical model can be obtained exploiting the maximum entropy principle, where the model's constraints involve the matching of frequency counts of single residues and pairs of residues across columns of the MSA. The model assigns a probability to each possible sequence of amino acids to be described by the alignment, and results in a Potts model. For the model parameters evaluation It can be used the MCE algorithm or Pipeline transformation + RRC algorithm, which solves efficiently the tensor network contraction problem associated with the steps of the Boltzmann learning method with gradient ascent. Once these parameters are obtained, a quantity called "direct information" between all couples of two sites in the primary structure of a protein can be calculated, resulting in a measure of direct correlations. The highest correlations will be associated to the native contacts: they can be used to identify how mutations alter protein structure and functions by comparing with the structure of the mutated proteins obtained with de novo protein sequencing using mass spectrometry, thereby allowing the design of drugs that specifically target the mutated regions of the proteins of interest.

13.a Introduction

The possible plan could be the following:

1. **Identification of Proteins of Interest:**
   Begin by identifying the protein of interest, often a key protein involved in cell growth regulation, DNA repair, or apoptosis. Examples include proteins encoded by oncogenes like RAS and EGFR or tumor suppressor genes like TP53 and RB1 [10].

2. **Collection of Homologous Sequences:**
   Use protein databases such as UniProt, GenBank, or Pfam [1] to collect homologous sequences retaining similar functions.

3. **Construction of Multiple Sequence Alignment (MSA):**
   Use software like Clustal Omega, MAFFT, or MUSCLE [1] to align the sequences of homologous proteins. MSA allows you to identify conserved regions and evolutionary variations indicative of native contacts and functional interactions.

4. **Analysis of Conservations and Correlations:**
   Analyze the MSA to identify conserved residues and pairs of residues showing evolutionary correlations. The strongest correlations indicate physical contacts between residues in the protein's three-dimensional structure [5].

5. **Prediction of Native Contacts:**
   Use the evolutionary correlation data to predict native contacts in the protein of interest. These contacts represent the interactions between amino acid residues that stabilize the protein's tertiary structure.

6. **Identification of Pathogenic Mutations:**
   Use native contacts to identify how mutations alter the protein's structure and function. De novo protein sequencing using mass spectrometry enables the determination of the amino acid sequence of a mutated protein. This technique isolates the protein, digests it into peptides, and uses tandem mass spectrometry to analyze the fragmentation patterns [10]. The resulting data are then interpreted to reconstruct the sequence directly from the protein sample, identifying any unknown mutation.

7. **Design of Targeted Drugs:**
   Design drugs that specifically interact with the mutated regions of proteins.

8. **Study of Drug Resistance:**
   Predict how secondary mutations might alter native contacts and contribute to drug resistance.

13.b State of the art of DCA

Direct Coupling Analysis (DCA) is a statistical framework designed to capture the variability within a group of phylogenetically related biological sequences.
E.g. the bacterial two-component signaling (TCS) proteins [2] have been used in the work of Morcos et al. [5] because of the large number of TCS protein sequences, which were already numbered [11].
When applied to a multiple sequence alignment (MSA) [1] of sequences of length N, this model assigns a probability to each possible sequence of the same length. This probability indicates the likelihood that a given sequence belongs to the same category as those in the MSA, such as a specific protein family.
In the first articles in this framework (e.g.: [12]), a message-passing algorithm was used to implement DCA. This method is a natural evolution of simple covariance analysis methods. Indeed, the main issue with covariance analysis is that correlations between interacting residues induce indirect correlations between non-interacting residues[20].
DCA aims to disentangle direct from indirect correlations.
In the work of Weigt et al. [12] the approach used is known as mpDCA, while in the work of Morcos et al. [5], mfDCA is used, which is an algorithm based on the mean-field approximation: this method is faster, allowing for the rapid analysis of many long protein sequences.

My proposal to find the native contacts

1. Find the statistical mechanics model P describing the probability of a specific sequence of residues referring to a MSA of proteins homologous to the one we are interested in. This model will turn out to be the Potts model, described by interaction parameters $\{J_{ij}\}_{i,j\in\{1,\dots,N\},i<j}$ and field parameters $\{h_i\}_{i\in\{1,\dots,N\}}$ where N is the length of proteins in the alignment.
2. Use the Boltzmann learning gradient ascent method to estimate the parameters of P obtained in the previous point via marginals evaluation of P.
3. Update the parameters in the Boltzmann learning steps via the contraction of the tensor network associated with the marginals evaluations using MCE algorithm or Pipeline transformation + RRC algorithm.

Denote a protein sequence of amino acids in the MSA by $\vec{A} = (A_1, A_2, \dots, A_N)$ with $A_i \in \{\text{gap}\} \cup \{20 \text{ amino acids}\}$.
For each family, the protein sequences are collected in one MSA denoted by

$$\{(A_1^a, \dots, A_N^a) | a = 1, \dots, M\}$$

where N denotes the number of MSA columns, i.e. the length of the aligned protein domains, and M is the number of sequences in the MSA.
Starting with an MSA of a large number of sequences of a given protein domain, extracted using Pfam's hidden Markov models (HMMs) [1], we have access to the basic quantities [5]:

1. The frequency count for a single MSA column i:

$$f_i(A) = \frac{1}{M_{\text{eff}} + \lambda} \left(\frac{\lambda}{q} + \sum_{a=1}^{M} \frac{1}{m_a} \delta_{A,A_i^a} \right)$$

   This is the relative frequency of finding amino acid A in column i.

[20] The indirect correlations are caused by an indirect effect mediated by one or more intermediate residues, e.g. if residue A interacts with residue B, and residue B interacts with residue C, there might be a correlation between A and C even if they do not interact directly.

2. The frequency count for pairs of MSA columns i, j:

$$f_{ij}(A,B) = \frac{1}{M_{\text{eff}} + \lambda}\left(\frac{\lambda}{q^2} + \sum_{a=1}^{M} \frac{1}{m_a}\delta_{A,A_i^a}\delta_{B,A_j^a}\right)$$

This is the frequency with which amino acids A, B co-appear in the same protein sequence in MSA columns i, j.

where:

- $\delta_{A,B}$ is the Kronecker delta, which equals 1 if $A = B$ and 0 otherwise.
- q is the number of different symbols: $q = |\{\text{gap}\} \cup \{20 \text{ aminoacids}\}| = 21$.
- λ is a pseudocount to avoid issues with small samples.
- m_a is the number of sequences with more than 80% identity to sequence a and it aims at correcting for the sampling bias i.e. sequences without similar sequences take weight one, and sequences featuring similar sequences are down-weighted.
- M_{eff} is the effective number of sequences, calculated as $M_{\text{eff}} = \sum_{a=1}^{M} \frac{1}{m_a}$.

To separate direct from indirect couplings, a statistical model $P(\{A_1, \ldots, A_N\})$ depending on protein sequences $\{A_1, A_2, \ldots, A_N\}$ is defined, which satisfies the constraints:

1. $\sum_{A_k|k\neq i} P(\{A_1, \ldots, A_N\}) = f_i(A_i) \qquad \forall i, A_i$
2. $\sum_{A_k|k\neq i,j} P(\{A_1, \ldots, A_N\}) = f_{ij}(A_i, A_j) \qquad \forall i, j, A_i, A_j$

Applying the maximum entropy principle imposing the constraints defined above we obtain a statistical model assigning a probability to a full sequence $\{A_1, A_2, \ldots, A_N\}$

$$P(\{A_1, \ldots, A_N\}) = \frac{1}{Z}\exp\left\{\sum_{i=1}^{N}\left(\sum_{j=i+1}^{N} J_{ij}(A_i, A_j) + h_i(A_i)\right)\right\}$$

where Z is

$$Z = \sum_{\{A_1,\ldots,A_N\}} \exp\left\{\sum_{i=1}^{N}\left(\sum_{j=i+1}^{N} J_{ij}(A_i, A_j) + h_i(A_i)\right)\right\}$$

The latter is a Potts model[21] with $q = 21$ on a fully connected graph, where all residues interact with each other and

1. J_{ij} represents an interaction term quantifying the compatibility between symbols at different columns i, j of the MSA.
2. h_i represents the propensity of a symbol to be found at a certain column i of the MSA.
3. Z is the partition function of the Potts model.
4. N is the length of protein sequences in the MSA.

Once the couplings J_{ij} and fields h_i are known, direct information (DI) is introduced: it measures the mutual information (MI) due to the direct coupling.
DI can be calculated using an isolated two-site model described by $P_{ij}^{(direct)} = P_{ij}^{(dir)}$:

$$P_{ij}^{(dir)}(A,B) = \frac{1}{Z_{ij}}\exp\left(J_{ij}(A,B) + \tilde{h}_i(A) + \tilde{h}_j(B)\right)$$

[21] Potts model on wikipedia

where the auxiliary fields $\tilde{h}_i$ are determined to match the empirical single-residue counts:

$$f_i(A) = \sum_B P_{ij}^{(dir)}(A,B)$$

$$f_j(B) = \sum_A P_{ij}^{(dir)}(A,B)$$

Finally, direct information is defined as the Kullback-Leibler divergence between P_{ij}^{dir} and $f_i f_j$:

$$DI_{ij} = \sum_{A,B} P_{ij}^{(dir)}(A,B) \ln\left(\frac{P_{ij}^{(dir)}(A,B)}{f_i(A) f_j(B)}\right)$$

To appreciate the difference between DI and MI, the latter based just on frequency counts f_i, f_{ij} as

$$MI_{ij} = \sum_{A,B} f_{ij}(A,B) \ln\left(\frac{f_{ij}(A,B)}{f_i(A) f_j(B)}\right)$$

consider the differences in the contacts predictions between MI and DI, given in figure 51, and made by Morcos et al. [5] for a specific family of protein domains that are homologous to Region 2 of a bacterial Sigma factor. These contact predictions were applied to the sequence of the SigmaE factor of the bacterium Escherichia coli, which is encoded by the rpoE gene, and whose structure has been resolved and deposited in the Protein Data Bank (PDB) with ID 1OR7.

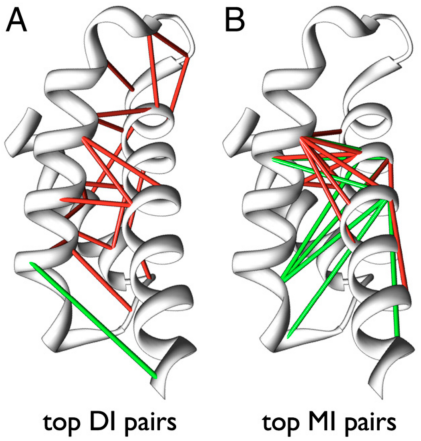

Figure 51: Analysis of Morcos et al [5] for contact predictions for the family of domains homologous to Region 2 of the bacterial Sigma factor (Pfam ID PF04542) mapped to the sequence of the SigmaE factor of E. coli (encoded by rpoE) (PDB ID 1OR7). The A panel shows the top 20 DI predictions while the B panel shows the top 20 MI predictions for residue–residue contacts. Each pair with distance < 8 Å is connected by a red link, and the more distant pairs are connected by the green links.

13.c Boltzmann Learning

The Boltzmann learning (BL) method with numerical gradient ascent can be applied to the Potts model for estimating the set of couplings $J = \{J_{ij}\}_{i,j\in\{1,\dots,N\},i<j}$ and the set of fields $h = \{h_i\}_{i\in\{1,\dots,N\}}$. Each $A \in \{\text{gap}\} \cup \{20 \text{ amino acids}\}$ is indicated by a number going from 1 to 21.
In the Potts model the probability of a configuration $\vec{s}$ where each s_i takes values in $\{1,\dots,q\}$ given the sets of parameters of the model J, h, is given by

$$P(\vec{s}\,|J,h) = \frac{1}{Z} \exp\left\{\sum_{i=1}^{N}\left(\sum_{j=i+1}^{N} J_{ij}(s_i,s_j) + h_i(s_i)\right)\right\}$$

The objective of the BL is to estimate the sets of parameters J and h that maximize the likelihood of the observed data, i.e. find J and h such that the probability of the observed data is maximized. The observed data are configurations of N amino acids $\vec{s}^{\,(o)} = \{s_1^{(o)}, s_2^{(o)}, \dots, s_N^{(o)}\}$ where the observation o is in the set of total observations O, which is the MSA.
The log likelihood of the observed data $\vec{s}^{\,(o)}, \forall o \in O$, is given by

$$\log \mathcal{L}(J,h) = \log \prod_{o \in O} P(\vec{s}^{\,(o)}|J,h) = \sum_{o \in O} \log P(\vec{s}^{\,(o)}|J,h)$$

To maximize $\log \mathcal{L}$, the derivatives with respect to the parameters in the sets J and h are considered. Indeed the partial derivative of $\log \mathcal{L}$ with respect to $J_{ij}(s_i, s_j)$ and $h_i(s_i)$ are given by

$$\begin{cases} \dfrac{\partial \log \mathcal{L}(J,h)}{\partial J_{ij}(s_i = A, s_j = B)} = \langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{data}} - \langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{model}} \\ \dfrac{\partial \log \mathcal{L}(J,h)}{\partial h_i(s_i = A)} = \langle \delta_{s_i,A} \rangle_{\text{data}} - \langle \delta_{s_i,A} \rangle_{\text{model}} \end{cases}$$

calculated $\forall i, j \in \{1, \dots, N\},\ i < j,\ \forall s_i, s_j \in \{1, \dots, 21\}$, where

1. $\langle \cdot \rangle_{\text{data}}$ denotes the average over the observed data.
2. $\langle \delta_{s_i,A} \rangle_{\text{data}} = f_i(A)$ where $f_i(A)$ is the frequency count for amino acid A in a single MSA column i.
3. $\langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{data}} = f_{ij}(A,B)$ where $f_{ij}(A,B)$ is the frequency count for pairs (A,B) of MSA columns i, j.
4. $\langle \cdot \rangle_{\text{model}}$ denotes the average with respect to the model's probability distribution.
5. $\langle \delta_{s_i,A} \rangle_{\text{model}} = P(s_i = A|J,h) = \sum_{s_k, k \neq i} P(s_1, \dots, s_{i-1}, A, \dots, s_N|J,h)$
6. $\langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{model}} = P(s_i = A, s_j = B|J,h) = \sum_{s_k, k \neq i,j} P(s_1, \dots, s_{i-1}, A, \dots, s_{j-1}, B, \dots, s_N|J,h)$

13.d Numerical gradient ascent

The numerical gradient ascent method is used to iteratively update the sets of parameters J and h using the calculated partial derivatives

$$\begin{cases} J_{ij}^{(t+1)}(s_i, s_j) = J_{ij}^{(t)}(s_i, s_j) + \eta \dfrac{\partial \log \mathcal{L}(J,h)}{\partial J_{ij}(s_i, s_j)} \\ h_i^{(t+1)}(s_i) = h_i^{(t)}(s_i) + \eta \dfrac{\partial \log \mathcal{L}(J,h)}{\partial h_i(s_i)} \end{cases}$$

calculated $\forall i, j \in \{1, \dots, N\},\ i < j,\ \forall s_i, s_j \in \{1, \dots, 21\}$, where η is the learning rate, a parameter that controls the step size of the update.

13.e Proposed Methodologies

The averages $\langle \cdot \rangle_{\text{data}}$ are easily computed from the observed MSA data.
However computing the averages $\langle \cdot \rangle_{\text{model}}$ is in general computationally intensive because it requires summing over all possible configurations of the system which are $q^N = 21^N$: for proteins of MSA-length equal to e.g. 70, means 21^{70} operations for the evaluation of a single marginal for a single step of the update procedure. Clearly facing this problem calculating exhaustively this sums is unfeasible. We can use the MCE algorithm or Pipeline transformation + RRC algorithm, indeed the sets of parameter of the Potts model can be evaluated in the following way:

1. **Initialization**: the sets of parameters J and h are initialised with random or predefined values.
2. **Gradient calculation**: compute the averages $\langle \delta_{s_i,A}\delta_{s_j,B}\rangle_{\text{model}}$ and $\langle \delta_{s_i,A}\rangle_{\text{model}}$ using MCE algorithm or Pipeline transformation + RRC algorithm on the underlying network, encoding all the possible useful biological information to make the contraction procedure as efficient as possible. Indeed in the case of MCE algorithm the two possible cycle weights definitions I've mentioned in the dedicated chapter are based on general properties of regular graphs but *they can be specifically designed* for the specific problem of interest, focusing on the properties of the protein's family we are analising with the MSA.
3. **Parameters update**: update the sets of parameters J and h using the gradient ascent procedure.
4. **Iteration**: repeat steps 2 and 3 until a chosen norm of the difference between the sets of parameters at step t and $t+1$ is below a chosen threshold.

In our case, the contraction problem involves the calculation of averages over the model parameters for the Boltzmann learning update procedure. In the language of tensor networks, this corresponds to a total contraction of the network associated with the model considering the addition of

1. For the calculation of $\langle \delta_{s_i,A}\delta_{s_j,B}\rangle_{\text{model}}$: two tensors encoding respectively $\delta_{s_i,A}$ and $\delta_{s_j,B}$ connected to copy tensors associated to s_i and s_j.
   In figure 52 it is depicted the graphical representation of the graph associated with the Potts model in the case of $N = 40$, when $i = 1$ and $j = 20$.
2. For $\langle \delta_{s_i,A}\rangle_{\text{model}}$: a tensor encoding $\delta_{s_i,A}$ connected to the copy tensor associated to s_i.

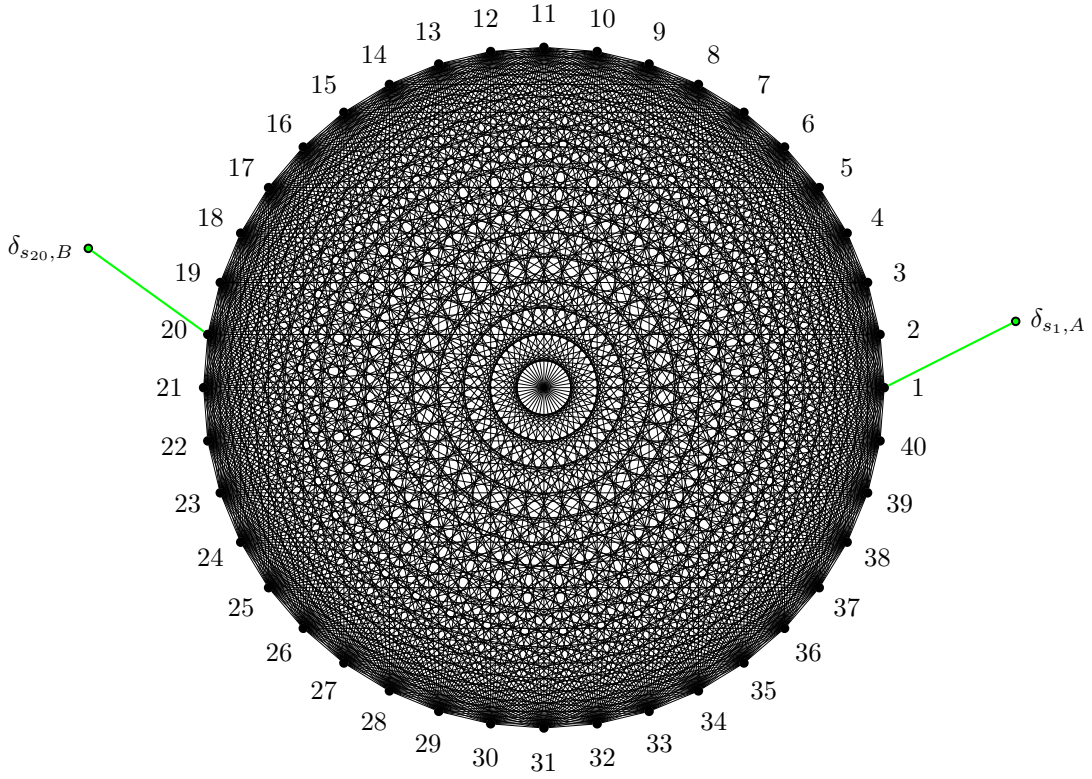

Figure 52: Marginal $\langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{model}}$ evaluation for the Boltzmann learning procedure in the case of a MSA with proteins length $N = 40$, and considering $i = 1$ and $j = 20$. The 40 nodes represents copy tensors associated with the spins present on nodes, while edges contain a tensor encoding the pairwise interaction + fields of the two spins connected by that edge. The two green nodes added are vectors with length equal to 21 and one single non-zero entry, the one corresponding to the amino acid selected by the delta function. The amino acids are represented by numbers going from 1 to 21, so if the amino acid selected by the delta function is A and the number representing it is 5, then the vector $\delta_{s_i,A}$ will be full of zeros with a single 1 in position 5.

Consider now $h_i(A)$ as a matrix where the rows labeled by i indicate the position in the protein sequence and the column label A indicate the specific symbol (taken from the list of 21 symbols) that is present. The same idea can be applied to $J_{ij}(A, B)$, which will result in a 4-order tensor with entries labeled by i, j, A, B.
Following the same reasoning we used for the Ising model, a small imaginary field $\tilde{h} = ix$ is added on the component of the original real set of fields h^0 corresponding to the entry (i, A) for the evaluation of $\langle \delta_{s_i,A} \rangle_{\text{model}}$. For the evaluation of $\langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{model}}$ a small imaginary coupling $\tilde{J} = ix$ is added on the component of the set of original real couplings J^0, corresponding to the entry (i, j, A, B).
Define $\mathbb{1}_i^A$ a matrix with entries all equal to 0 except the (i, A)-th entry which is set to 1.
Define $\mathbb{1}_{i,j}^{A,B}$ a 4-order tensor with entries all equal to 0 except the (i, j, A, B)-th entry, which is set to 1.
If x is chosen sufficiently small, the following expansions can be safely stopped to the first order.

Consider first the case of $\langle \delta_{s_i,A} \rangle_{\text{model}}$:

$$\log Z(J,h') = \log Z(J,h^0 + \tilde{h}\mathbb{1}_i^A) \simeq \log Z(J,h^0) + ix \left\{ \frac{\partial \log Z(J,h)}{\partial h_i(A)} \right\} \Big|_{h=h^0} =$$

$$= \log Z(J,h^0) + ix \frac{1}{Z(J,h^0)} \sum_{\{s_1,\ldots,s_N\}} \exp\left\{ \sum_{i=1}^{N} \left(\sum_{j=i+1}^{N} J_{ij}(s_i s_j) + h_i(s_i) \right) \right\} \delta_{s_i,A} =$$

$$= \log Z(J,h^0) + ix \langle \delta_{s_i,A} \rangle_{\text{model}}$$

$$\Rightarrow \langle \delta_{s_i,A} \rangle_{\text{model}} \simeq \frac{1}{x} \mathfrak{Im} \left\{ \log Z(J,h') \right\}$$

where $\mathfrak{Im}\{\cdot\}$ is the imaginary part.
Now consider $\langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{model}}$:

$$\log Z(J',h) = \log Z(J^0 + \tilde{J}\mathbb{1}_{i,j}^{A,B}, h) \simeq \log Z(J^0,h) + ix \left\{ \frac{\partial \log Z(J,h)}{\partial J_{ij}(A,B)} \right\} \Big|_{J=J^0}$$

$$= \log Z(J^0,h) + ix \frac{1}{Z(J^0,h)} \sum_{\{s_1,\ldots,s_N\}} \exp\left\{ \sum_{i=1}^{N} \left(\sum_{j=i+1}^{N} J_{ij}(s_i s_j) + h_i(s_i) \right) \right\} \delta_{s_i,A} \delta_{s_j,B}$$

$$= \log Z(J^0,h) + ix \langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{model}}$$

$$\Rightarrow \langle \delta_{s_i,A} \delta_{s_j,B} \rangle_{\text{model}} \simeq \frac{1}{x} \mathfrak{Im} \left\{ \log Z(J',h) \right\}$$

References

[1] Richard Durbin et al. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Apr. 1998. ISBN: 9780511790492. DOI: 10.1017/cbo9780511790492. URL: http://dx.doi.org/10.1017/cbo9780511790492.

[2] James A Hoch. "Two-component and phosphorelay signal transduction". In: *Current Opinion in Microbiology* 3.2 (Apr. 2000), pp. 165–170. ISSN: 1369-5274. DOI: 10.1016/s1369-5274(00)00070-9. URL: http://dx.doi.org/10.1016/s1369-5274(00)00070-9.

[3] Y. L. Loh and E. W. Carlson. "Efficient Algorithm for Random-Bond Ising Models in 2D". In: *Physical Review Letters* 97.22 (Nov. 2006). ISSN: 1079-7114. DOI: 10.1103/physrevlett.97.227205. URL: http://dx.doi.org/10.1103/PhysRevLett.97.227205.

[4] Igor L. Markov and Yaoyun Shi. "Simulating Quantum Computation by Contracting Tensor Networks". In: *SIAM Journal on Computing* 38.3 (2008), pp. 963–981. DOI: 10.1137/050644756. eprint: https://doi.org/10.1137/050644756. URL: https://doi.org/10.1137/050644756.

[5] Faruck Morcos et al. "Direct-coupling analysis of residue coevolution captures native contacts across many protein families". In: *Proceedings of the National Academy of Sciences* 108.49 (Nov. 2011). ISSN: 1091-6490. DOI: 10.1073/pnas.1111471108. URL: http://dx.doi.org/10.1073/pnas.1111471108.

[6] I. V. Oseledets. "Tensor-Train Decomposition". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317. DOI: 10.1137/090752286. eprint: https://doi.org/10.1137/090752286. URL: https://doi.org/10.1137/090752286.

[7] N. Robertson and P.D. Seymour. "Graph Minors .XIII. The Disjoint Paths Problem". In: *Journal of Combinatorial Theory, Series B* 63.1 (1995), pp. 65–110. ISSN: 0095-8956. DOI: https://doi.org/10.1006/jctb.1995.1006. URL: https://www.sciencedirect.com/science/article/pii/S0095895685710064.

[8] Neil Robertson and P.D. Seymour. "Graph minors. I. Excluding a forest". In: *Journal of Combinatorial Theory, Series B* 35.1 (1983), pp. 39–61. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(83)90079-5. URL: https://www.sciencedirect.com/science/article/pii/0095895683900795.

[9] Neil Robertson and P.D. Seymour. "Graph Minors. XX. Wagner's conjecture". In: *Journal of Combinatorial Theory, Series B* 92.2 (2004). Special Issue Dedicated to Professor W.T. Tutte, pp. 325–357. ISSN: 0095-8956. DOI: https://doi.org/10.1016/j.jctb.2004.08.001. URL: https://www.sciencedirect.com/science/article/pii/S0095895604000784.

[10] R. C. Sobti, Nirmal K. Ganguly, and Rakesh Kumar. *Handbook of Oncobiology: From Basic to Clinical Sciences*. Singapore: Springer, 2024. ISBN: 9789819962631. DOI: 10.1007/978-981-99-6263-1.

[11] Luke E. Ulrich and Igor B. Zhulin. "The MiST2 database: a comprehensive genomics resource on microbial signal transduction". In: *Nucleic Acids Research* 38.suppl$_1$ (Nov. 2009), pp. D401–D407. ISSN: 1362-4962. DOI: 10.1093/nar/gkp940. URL: http://dx.doi.org/10.1093/nar/gkp940.

[12] Martin Weigt et al. "Identification of direct residue contacts in protein–protein interaction by message passing". In: *Proceedings of the National Academy of Sciences* 106.1 (Jan. 2009), pp. 67–72. ISSN: 1091-6490. DOI: 10.1073/pnas.0805923106. URL: http://dx.doi.org/10.1073/pnas.0805923106.