Entropy Estimation in Cluster Graphs

Katerine M. Sadie *, Johan A. du Preez, Willie Brink

Posted Date: 29 May 2026

doi: 10.20944/preprints202605.2092.v1

Keywords: probabilistic graphical model; cluster graph; entropy estimation

 doi:10.20944/preprints202605.2092.v1

Article

Entropy Estimation in Cluster Graphs

Katerine M Sadie *, Johan A du Preez and Willie Brink

Stellenbosch University, South Africa
* Correspondence: 21639361@sun.ac.za

Abstract

Probabilistic graphical models (PGMs) provide a powerful framework for modelling complex systems, but inference over loopy graphs requires approximate methods whose accuracy depends on how factors are clustered in the graphical representation. Existing factor clustering methods rely on the number of variables in a cluster as a proxy for memory cost and informational content—a loose upper bound that leads to suboptimal merging decisions. We address this limitation by proposing an efficient algorithm for estimating the joint entropy of a group of clusters without explicitly multiplying out the constituent factors, thereby avoiding the exponential computational cost that makes exact computation prohibitive. The algorithm integrates naturally with both static and dynamic graph restructuring methods, and reduces to the Kikuchi entropy approximation when applied to the complete graph. Experiments on models with up to 24 variables demonstrate that the algorithm produces accurate (when compared to ideal junction tree performance) entropy estimates across diverse model types, with errors remaining within tight bounds. Scalability is further validated on a substantially larger model defined over 2640 random variables. These results confirm that accurate entropy estimation is achievable wherever reliable probabilistic inference is possible, and that the proposed estimation algorithm yields objectively close approximations, thereby supporting improved clustering decisions in PGM structuring algorithms.

Keywords: probabilistic graphical model; cluster graph; entropy estimation

1. Introduction

Probabilistic graphical models (PGMs) can be used to model complex systems and perform inference on sets of random variables. While tree-structured PGMs allow for exact inference, their use is limited in practice due to exponential blow-up in complexity. Instead, for computational tractability, we often use loopy PGMs. Various graphical structures can be used when performing approximate inference over loopy graphs; the choice of structure influences the quality (i.e. accuracy) of the result [1]. Factor graphs [2] and simple cluster graphs (CGs) [3, p. 346; 4] reuse the original set of factors (potential functions) directly, placing each factor into its own node. More sophisticated graphical structures—such as those arising from the junction tree (JT) algorithm (which enables exact inference) [5], and the join graph structuring algorithm (which yields a loopy approximation to the JT) [6]—group several of these factors together in the nodes of a cluster graph, thereby enabling more accurate inference. There have also been attempts to dynamically restructure the graphical structure as inference progresses [7]. These more advanced graphical structuring techniques require a metric for deciding which factors should be clustered together. In essence, such a metric is used to limit the number of random variables that will be present in the union of the factors in the same cluster. This is important, since the number of random variables in a cluster acts as a loose upper bound for the memory footprint of the cluster. Due to the exponential blow-up in the number of states of such a joint distribution, this memory footprint in turn is an important consideration when clustering factors.

The number of variables in a distribution is also a loose upper bound on the entropy of that distribution (as we show in Section 2.1). From our perspective, instead of using this loose upper bound,

 doi:10.20944/preprints202605.2092.v1

a more refined metric for factor clustering should be based directly on estimating the joint entropy of a proposed cluster. In fact, we can use the entropy to answer two questions: how closely does a set of factors associate (lower entropy means they are more informative); and how large will the memory footprint of the joint be (which turns out to also be a particular entropy calculation, as we show in Section 3.2).

The challenge, however, is to estimate the entropies without first multiplying the factors in the cluster (which can be prohibitively expensive). In this paper we present an efficient algorithm for estimating the joint entropy of a set of clusters in a cluster group. This algorithm can be incorporated into advanced graphical model structuring methods, or used in other application that require entropy.

The computation of entropy in PGMs is a natural quantity of interest, appearing explicitly in the Bethe and Kikuchi free energy approximations [1], as well as playing a role in model selection [3]. Several works have addressed the computation of entropy and Kullback–Leibler divergence in PGMs, although these are focused primarily on Bayesian networks [8; 9; 10, p. 348]. For tree-structured PGMs, the entropy follows directly from the factorisation of the joint probability distribution [10, p. 348–349]; no such equation exists for cluster graphs. When applied to the complete graph (that is, the cluster group containing all clusters), our entropy estimation reduces to the Kikuchi approximation of entropy in a region graph, which is exact in trees and serves as an approximation on loopy graphs [1]. The novelty in our algorithm lies in the cluster group approach, which applies to arbitrary groupings of clusters, allowing for a more refined metric during cluster merging. Further, the algorithm to estimate the number of non-zero probabilities in a factor product—without explicitly multiplying out the factors—through a particular entropy calculation is, to the best of our knowledge, novel. Related work [5,6] use the number of variables as a proxy for this.

The remainder of the paper is structured as follows: In Section 2 we provide preliminaries and notation. In Section 3 we develop our novel entropy estimation method through a small example, and formalise it as an algorithm. In Section 4 we evaluate the quality of the entropy estimations through two experiments over large caches of models—reporting our estimation error through histograms, percentile intervals, and correlation coefficients—as well as a third experiment to demonstrate the scalability on a large real-world model. Finally, in Section 5 we summarise our findings and outline directions for future work.

2. Preliminaries

2.1. Probabilistic Graphical Models

A probabilistic graphical model (PGM) comprises a set of probability distributions and a graph representing the structure. The graph is made up of nodes (each representing one or more random variables) and edges (representing probabilistic relationships between the adjacent nodes) [11]. For a comprehensive coverage of PGMs, the reader is referred to the books of Koller and Friedman [3] and Bishop [11, Ch. 8].

Let $\boldsymbol{X}$ denote a set of random variables taking values $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$. A factor $\phi(\boldsymbol{X})$ is a function, with $\text{scope}[\phi] = \boldsymbol{X}$, mapping $\boldsymbol{x}$ to the non-negative real numbers [3, p. 104]. A factor can be seen as a generalisation of a probability distribution. Factors can be connected to one another in a PGM; our focus is on a class of PGMs called cluster graphs.

A cluster graph (CG) [3, p. 346] (variously also referred to as a junction graph [12, p. 281] or join graph [6]), defined for a set of factors Φ over a set of variables $\boldsymbol{X}$, is an undirected graph where

- each node i has an associated cluster $C_i \subseteq \boldsymbol{X}$,
- each edge between a pair of clusters C_i and C_j (with $i < j$) has an associated separation set (sepset for short) $S_{i,j} \subseteq C_i \cap C_j$,
- the graph is family preserving: each factor $\phi \in \Phi$ has an associated cluster C_i such that $\text{scope}[\phi] \subseteq C_i$, and
- the graph satisfies the running intersection property (RIP): for any pair of clusters containing a common variable, there is a unique path between the clusters containing that common variable.

 doi:10.20944/preprints202605.2092.v1

We will use the following notation for a general CG: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with clusters $\boldsymbol{C} = \{C_i \mid i \in \mathcal{V}\}$ and sepsets $\boldsymbol{S} = \{S_{i,j} \mid (i,j) \in \mathcal{E}\}$. Within the class of CGs, there are a few different graphical structures:

- The junction tree (JT) [3, p. 348; 5, p. 330] is constructed using variable elimination or the JT algorithm. The graph has a tree structure, and while inference on this structure can be performed exactly, due to exponential blow-up it is typically infeasible to construct and use in practice.
- The factor graph [2; 11, p. 360; 13]—variously also known as a Tanner graph [14; 15, p. 315] or Bethe graph [3, p. 405]—is a (loopy) bipartite graph, with a subset of nodes corresponding to the variables and another corresponding to the factors. Each sepset (implicitly) consists of a single variable. Inference is only approximate, hampered by the fact that message passing uses univariate marginal distributions for the sepsets, thus preventing inter-variable dependency information from propagating [3, p. 406].
- The general CG is a generalisation of the factor graph, moving beyond the limitations of single-variable sepsets [3, p. 406]. Construction algorithms include the layered trees running intersection property (LTRIP) algorithm [4] and the join-graph structuring algorithm [6]. The graph structure is typically loopy (although in rare cases it may be tree-structured) with multivariate sepsets. Although inference is again approximate, it improves on factor graphs in terms of both accuracy and speed [4].

Given a CG, there are several message passing algorithms we can use for inference, chief among them the Shafer-Shenoy [16] and Lauritzen-Spiegelhalter [17] algorithms. For a particular algorithm, some considerations include the message passing schedule and the choice of convergence criteria; we will provide details where relevant. In line with standard practice [18], we use sum-marginalisation and sum-normalisation for loopy-structured CGs, and their max counterparts for tree-structured CGs.

Through marginalisation and factor multiplication, message passing iteratively refines the cluster beliefs ($\boldsymbol{\Psi} = \{\Psi_i \mid i \in \mathcal{V}\}$ and sepset beliefs $\boldsymbol{\psi} = \{\psi_{i,j} \mid (i,j) \in \mathcal{E}\}$). In a tree structure, the beliefs are exactly the marginal distributions over the relevant variables [3, p. 400]. In a loopy structure, the beliefs are approximations, referred to as pseudo-marginals.

A key property of message passing is that beliefs represent a reparameterisation of the original distribution. Specifically, if Φ is the set of factors defining a distribution over the random variables $\boldsymbol{X}$, then at every iteration we have [3, p. 396]

$$\tilde{p}(\boldsymbol{X}) = \prod_{\phi \in \Phi} \phi = \frac{\prod_{i \in \mathcal{V}} \Psi_i(C_i)}{\prod_{(i,j) \in \mathcal{E}} \psi_{i,j}(S_{i,j})}, \tag{1}$$

where $\tilde{p}(\boldsymbol{X})$ denotes the unnormalised joint probability distribution. A CG reaches calibration when all adjacent clusters agree on the marginal belief of their shared sepset. That is, for each pair of clusters C_i and C_j with sepset $S_{i,j}$,

$$\sum_{C_i \setminus S_{i,j}} \Psi_i(C_i) = \sum_{C_j \setminus S_{i,j}} \Psi_j(C_j), \tag{2}$$

where $C_i \setminus S_{i,j}$ is the set difference between C_i and $S_{i,j}$.

We now shift the focus to entropy, before describing the proposed algorithm for entropy estimation in a CG.

2.2. Entropy

Entropy is a measure of the degree of uncertainty of a random variable. Let X be a discrete random variable that takes on values $x \in \mathcal{X}$, with probability distribution $p(x)$. The entropy of X is defined as [19]

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log(p(x)). \tag{3}$$

 doi:10.20944/preprints202605.2092.v1

We are free to choose the base of the logarithm, but it is customary to make it either 2 (in which case the entropy is measured in bits) or e (in which case the entropy is measured in nats); throughout this paper we use the former.

Equation (3) generalises to the case of multiple variables. Let $\boldsymbol{X}$ denote a set of variables that takes on values $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$, with probability distribution $p(\boldsymbol{x})$. The entropy of $\boldsymbol{X}$ is defined as

$$H(\boldsymbol{X}) = -\sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{X}}} p(\boldsymbol{x}) \log(p(\boldsymbol{x})). \tag{4}$$

From this we want to point out a special case that we will find useful in Section 3.2. Let us assume that there are N assignments to $\boldsymbol{X}$ that occur with equal probability $\frac{1}{N}$, and all other cases have probability equal to zero. Equation (4) then reduces to

$$H(\boldsymbol{X}) = \log(N). \tag{5}$$

Thus, in this case we have a direct relationship between the number of non-zero probabilities (i.e. the table size N) and the entropy of the distribution.

2.3. Entropy When the PGM Has a Tree Structure

Let $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ denote a tree-structured CG (such as a JT). The probability distribution over its variables $\boldsymbol{X}$ can be factorised as

$$p(\boldsymbol{X}) = \frac{\prod_{i\in\mathcal{V}} p(C_i)}{\prod_{(i,j)\in\mathcal{E}} p(S_{i,j})}, \tag{6}$$

and the joint entropy of $\boldsymbol{X}$ is then given by [10]

$$H(\boldsymbol{X}) = \sum_{i\in\mathcal{V}} H(C_i) - \sum_{(i,j)\in\mathcal{E}} H(S_{i,j}). \tag{7}$$

Importantly, each entropy component in Equation (7) is computed from the true marginal distribution of the relevant variables (whether cluster or sepset). We now pose the question: what is to be done when the system is too large for a tree-structured CG, and the graph structure is loopy? In this case Equation (6) does not represent a factorisation in terms of the true marginal distributions. However, looking to Equation (1), it still factorises (exactly) in terms of the cluster beliefs ($\boldsymbol{\Psi}$) and sepset beliefs ($\boldsymbol{\psi}$), where the beliefs serve as pseudo-marginals—that is, approximations to the corresponding marginal distributions.

We use this reparameterisation property in the next section, where we investigate entropy in PGMs with loopy structures.

3. Entropy Estimation

3.1. Entropy When the PGM Has a Loopy Structure

Equation (7) provides an exact solution for calculating entropy when the PGM has a tree structure, but does not hold when the graph has loops. In this section we explain how to deal with such a situation, and ultimately present our algorithm for entropy estimation.

Running example:

We use a small loopy-structured PGM as a running example to illuminate our reasoning. Suppose we have four factors $\phi_0(v_0, v_3, v_4)$, $\phi_1(v_0, v_2, v_3)$, $\phi_2(v_1, v_2, v_3)$ and $\phi_3(v_1, v_3, v_4)$ defining a joint probability distribution p over random variables $\boldsymbol{X}$. These are linked up in a structure as shown in Figure 1.

 doi:10.20944/preprints202605.2092.v1

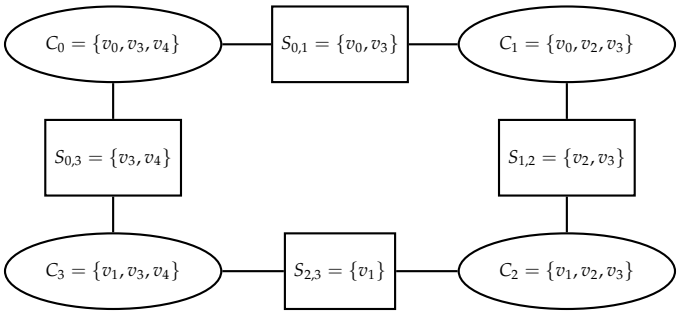

Figure 1. A simple cluster graph (CG) $\mathcal{G}$. Note that, in order to prevent breaking the RIP with a cycle in v_3, we have $S_{2,3} \neq C_2 \cap C_3$.

Since Equation (7) applies to JTs, a possible solution is to use the JT algorithm [3, p. 348; 5, p. 330] to transform the loopy graph into an equivalent tree structure $\mathcal{T} = (\mathcal{V}, \mathcal{E})$. Such a structure is shown in Figure 2.

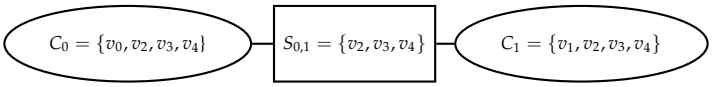

Figure 2. A simple junction tree (JT) $\mathcal{T}$. Note that the clusters now operate on larger sets of variables.

To calculate the (exact) total entropy, we use Equation (7). For this, we would need to perform message passing on the tree until calibration (which, for a tree, is one forward and backward sweep), and extract the beliefs (exact marginals). The entropy is then given by

$$\begin{aligned} H_p(\boldsymbol{X}) &= \sum_{i \in \mathcal{V}} H_p(C_i) - \sum_{(i,j) \in \mathcal{E}} H_p(S_{i,j}) \\ &= H_p(C_0) + H_p(C_1) - H_p(S_{0,1}). \end{aligned} \tag{8}$$

Note that the JT algorithm may result in a change in the number of clusters, as well as larger clusters. In many cases of interest, the exponential growth in computational cost that results from larger clusters in the JT makes it computationally infeasible. This then forces us back to the loopy graph of Figure 1. Knowing that loopy inference will result in pseudo-marginals, we propose calculating the entropy from these, instead of the (often unavailable) true marginals:

$$\hat{H}_p(\boldsymbol{X}) = \sum_{i \in \mathcal{V}} H_{\Psi_i}(C_i) - \sum_{(i,j) \in \mathcal{E}} H_{\psi_{i,j}}(S_{i,j}). \tag{9}$$

This can be shown to be a special case of the Kikuchi approximation of entropy in a region graph—the clusters being the outer regions, and the sepsets being the inner regions [1].

Similarly, for an individual cluster's entropy, we propose using the pseudo-marginal and calculating the entropy directly on it, that is,

$$\hat{H}_p(C_i) = H_{\Psi_i}(C_i). \tag{10}$$

As our interest includes grouping clusters, we need to be able to estimate the entropy of such a group. Suppose we want to estimate the entropy of cluster group $\boldsymbol{C}' = \{C_0, C_1\}$ in the context of the underlying probability distribution.

In Figure 3, the subgraph $\mathcal{G}'$ of $\mathcal{G}$ (from Figure 1) induced by $\boldsymbol{C}'$ is shown.

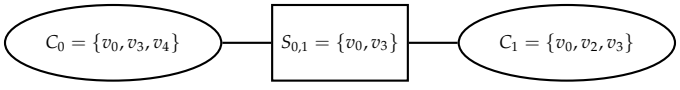

Figure 3. The subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ of $\mathcal{G}$ (from Figure 1) induced by the cluster group $\boldsymbol{C}' = \{C_0, C_1\}$.

 doi:10.20944/preprints202605.2092.v1

Notice that this is a valid CG, as it adheres to the RIP. We propose a two-stage approach for estimating the entropy of the cluster group. First, we perform message passing until calibration on the original graph $\mathcal{G}$ (from Figure 1), yielding the pseudo-marginals for all clusters and sepsets. Second, we identify the relevant pseudo-marginals from the induced subgraph $\mathcal{G}'$ (from Figure 3), and apply Equation (9) to obtain

$$\begin{aligned}\hat{H}_p(\boldsymbol{C}') &= \sum_{i\in\mathcal{V}'} H_{\Psi_i}(C_i) - \sum_{(i,j)\in\mathcal{E}'} H_{\psi_{i,j}}(S_{i,j}) \\ &= H_{\Psi_0}(C_0) + H_{\Psi_1}(C_1) - H_{\psi_{0,1}}(S_{0,1}).\end{aligned} \tag{11}$$

The subtraction of $H_{\psi_{0,1}}(S_{0,1})$ can be understood as compensating for variables in common in the adjacent clusters, which would otherwise be overcounted in the entropy calculation (echoing an inclusion-exclusion approach).

Now suppose we were interested in estimating the entropy of cluster group $\boldsymbol{C}' = \{C_2, C_3\}$ in the context of the underlying probability distribution. If we were to follow the same approach as before, our induced subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ would be as in Figure 4.

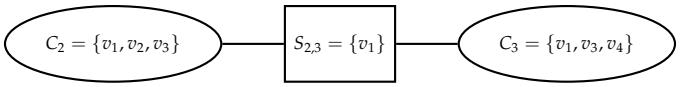

Figure 4. The subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ of $\mathcal{G}$ (from Figure 1) induced by the cluster group $\boldsymbol{C}' = \{C_2, C_3\}$.

Here we run into a problem. While both C_2 and C_3 contain variable v_3, it is not in $S_{2,3}$ due to the RIP in the original graph $\mathcal{G}$. Since the sepset is not the complete overlap, clearly we would be subtracting too little, that is, "undercount" $H_{\psi_{2,3}}(S_{2,3})$ in Equation (7). An underlying structural indication of this problem is that the induced subgraph does not adhere to the RIP.

However, during the CG construction process, it is somewhat arbitrary where to constrict a sepset to achieve the RIP, as long as it is done somewhere. We could have, without detriment, constructed the CG as in Figure 5.

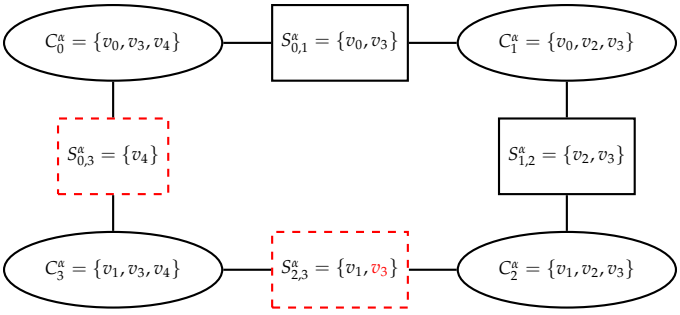

Figure 5. An alternative CG $\mathcal{G}^\alpha$. The only difference between $\mathcal{G}$ and $\mathcal{G}^\alpha$ is the sepset v_3 is excluded from—$S_{0,3}$ or $S_{2,3}$.

In this case, $S_{2,3}^\alpha$ is equal to the full overlap between C_2^α and C_3^α, so the induced subgraph of $\mathcal{G}^\alpha$ is a valid CG (adhering to the RIP), and Equation (9) can be applied. But the question is, can we achieve this without performing the full recalibration process—constructing $\mathcal{G}^\alpha$, performing message passing until calibration, extracting the induced subgraph, and applying Equation (9)?

The key observation is that $\mathcal{G}$ and $\mathcal{G}^\alpha$ are both valid CGs (adhering to the RIP) with identical clusters. If we assume that the resulting cluster beliefs are sufficiently close, we can exploit the

 doi:10.20944/preprints202605.2092.v1

structural similarity of the CGs to obtain the relevant subgraph. We can modify the reparameterisation (Equation (1)) without changing the original distribution, as follows:

$$\begin{aligned}\tilde{p}(\boldsymbol{X}) &= \phi_0(v_0,v_3,v_4)\phi_1(v_0,v_2,v_3)\phi_2(v_1,v_2,v_3)\phi_3(v_1,v_3,v_4) && \text{(original)}\\ &= \frac{\Psi_0(v_0,v_3,v_4)\Psi_1(v_0,v_2,v_3)\Psi_2(v_1,v_2,v_3)\Psi_3(v_1,v_3,v_4)}{\psi_{0,1}(v_0,v_3)\psi_{1,2}(v_2,v_3)\psi_{2,3}(v_1)\psi_{0,3}(v_3,v_4)} && \text{(reparameterised)}\\ &= \frac{\Psi_0(v_0,v_3,v_4)\Psi_1(v_0,v_2,v_3)\frac{\Psi_2(v_1,v_2,v_3)}{\psi_{2,3}(v_1)}\Psi_3(v_1,v_3,v_4)}{\psi_{0,1}(v_0,v_3)\psi_{1,2}(v_2,v_3)\psi^{\dagger}_{2,3}(v_1,v_3)\psi_{0,3}(v_3,v_4)} && \text{(reinterpreted)} \qquad (12)\end{aligned}$$

where $\psi^{\dagger}_{2,3}(v_1,v_3)$ is a uniform distribution over the (now extended) random variables $S^{\dagger}_{2,3}=\{v_1,v_3\}$. Note that division by a uniform distribution will not change the shape of $\tilde{p}(\boldsymbol{X})$. That is, if we reparameterise by replacing $\Psi_2(v_1,v_2,v_3)$ with a new (non-normalised) distribution $\Psi^{\dagger}_2(v_1,v_2,v_3)=\frac{\Psi_2(v_1,v_2,v_3)}{\psi_{2,3}(v_1)}$, and also replacing the original sepset belief $\psi_{2,3}(v_1)$ with a uniform sepset belief $\psi^{\dagger}_{2,3}(v_1,v_3)$, it will leave the original $\tilde{p}(\boldsymbol{X})$ intact. We should not try to recalibrate the (implied) new graph though, since the extended $S^{\dagger}_{2,3}$ sepset now breaks the RIP requirement. However, we can extract from this (implied) new graph the subgraph induced by the cluster group $\boldsymbol{C}'=\{C_2,C_3\}$, as shown in Figure 6.

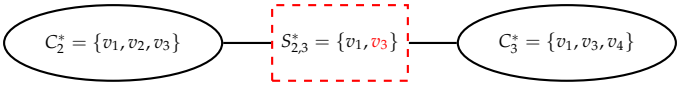

Figure 6. The recalibration graph $\mathcal{G}^*=(\mathcal{V}^*,\mathcal{E}^*)$ of $\mathcal{G}$ corresponding to cluster group $\boldsymbol{C}'=\{C_2,C_3\}$.

This subgraph does satisfy the RIP, and further recalibration on it will (due the reparameterisation property in Equation (1)) also preserve its $\frac{\Psi^{\dagger}_2(v_1,v_2,v_3)\Psi_3(v_1,v_3,v_4)}{\psi^{\dagger}_{2,3}(v_1,v_3)}$ contribution to Equation (12). Thus, the updated approach for estimating the entropy of the cluster group works as follows: as before, we perform message passing until calibration on the original graph $\mathcal{G}$ (from Figure 1) yielding the pseudo-marginals for all clusters and sepsets. We obtain the induced subgraph ($\mathcal{G}'$) and corresponding beliefs, and then construct the recalibration graph $\mathcal{G}^*$. Starting with "factor functions" $\Psi^{\dagger}_2(v_1,v_2,v_3)=\frac{\Psi_2(v_1,v_2,v_3)}{\psi_{2,3}(v_1)}$, and $\Psi_3(v_1,v_3,v_4)$ (and a uniform distribution for the sepset), we perform message passing on $\mathcal{G}^*$ until calibration, and then apply Equation (9), yielding

$$\begin{aligned}\hat{H}_p(\boldsymbol{C}') &= \sum_{i\in\mathcal{V}^*} H_{\Psi^*_i}(C_i) - \sum_{(i,j)\in\mathcal{E}^*} H_{\psi^*_{i,j}}(S_{i,j})\\ &= H_{\Psi^*_2}(C_2) + H_{\Psi^*_3}(C_3) - H_{\psi'^*_{2,3}}(S'_{2,3}),\end{aligned} \qquad (13)$$

where Ψ^*_i and $\psi^*_{i,j}$ denote the beliefs after recalibration.

From this, we extrapolate to the more general situation involving two or more clusters that are not already internally linked in an RIP-satisfying configuration. First, we collect the set of cluster and sepset beliefs corresponding to the cluster group $\boldsymbol{C}'$ for which we want to estimate the entropy. We divide out the relevant sepset beliefs, build a new (RIP-compliant) CG with these factors using LTRIP or any other valid cluster graph construction technique, and recalibrate it. The entropy estimate can then be obtained directly by using Equation (7) as before.

We now formalise this process in an algorithm for estimating the joint entropy of the variables contained in any subset of clusters present in a (potentially larger) PGM. Note that if this PGM has a tree structure, the algorithm will provide the exact (correct) entropy [10, p. 349]. In a loopy-structured PGM, the result will be an approximation to the true entropy. The algorithm is applicable to systems of discrete random variables only; continuous random variables (or mixtures of discrete and continuous variables) would require extensions suited to differential entropy.

There are two special cases of the cluster group $\boldsymbol{C}'$. The first is when we have a single cluster ($\boldsymbol{C}'=\{C_i\}$), and the second when we have the group consisting of the full set of clusters ($\boldsymbol{C}'=\boldsymbol{C}$). It is easily verified that the recalibration steps are redundant in these cases.

 doi:10.20944/preprints202605.2092.v1

Algorithm 1 Estimating the entropy of a subset of clusters in a cluster graph (CG).

Require: A CG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with clusters $\boldsymbol{C}$, and a cluster group $\boldsymbol{C}' \subseteq \boldsymbol{C}$.
Initial inference:
Perform message passing on $\mathcal{G}$ until calibration.
Obtain the cluster beliefs $\boldsymbol{\Psi}$ and sepset beliefs $\boldsymbol{\psi}$.
Filter according to $\boldsymbol{C}'$:
Construct the subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ induced by $\boldsymbol{C}'$.
Filter the cluster beliefs using $\mathcal{G}'$: $\boldsymbol{\Psi}' \leftarrow \{\Psi_i \mid i \in \mathcal{V}'\}$.
Filter the sepset beliefs using $\mathcal{G}'$: $\boldsymbol{\psi}' \leftarrow \{\psi_{i,j} \mid (i,j) \in \mathcal{E}'\}$.
Recalibration:
Divide away the sepset beliefs: for each $(i,j) \in \mathcal{E}'$ (with $i < j$), set $\Psi_i \leftarrow \frac{\Psi_i}{\psi_{i,j}}$.
Construct the recalibration CG $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ with the resulting set of cluster beliefs acting as its factor functions.
Perform message passing on $\mathcal{G}^*$ until calibration.
Obtain the cluster beliefs $\boldsymbol{\Psi}^*$ and sepset beliefs $\boldsymbol{\psi}^*$.
Entropy of cluster group $\boldsymbol{C}'$:
Estimate the entropy of the cluster group $\boldsymbol{C}'$ as

$$\hat{H}(\boldsymbol{C}') = \sum_{i \in \mathcal{V}^*} H_{\Psi_i^*}(C_i) - \sum_{(i,j) \in \mathcal{E}^*} H_{\psi_{i,j}^*}(S_{i,j}). \tag{14}$$

3.2. Using Entropy to Estimate Probability Table Sizes

In the following, we are specifically limiting ourselves to sparse representations where only non-zero probabilities are recorded in the probability table (for a dense representation, the table size will always be maximal, which rapidly becomes untenable). When we consider which clusters in a PGM to merge, it is crucial to have a decent prior estimate of how large the resultant probability table will be. Using a direct estimate of the entropy (such as provided by Algorithm 1) as a proxy for table size will not suffice, since there may be many small probabilities present. Each of these may not contribute much to the entropy, but still requires a separate probability to be recorded in the probability table.

We also note that when factors in a cluster share variables—the typical situation during factor merging—the exact size of the resulting factor product depends on which variable assignments have non-zero probabilities across those factors. For instance, it is possible for a factor product to become completely empty if the constituent factors fully disagree on the valid assignments to their overlapping variables.

We approach untying this knot as follows:

- We first note that the *number* of non-zero probabilities in the resultant factor product will not change if we vary any of the non-zero probabilities in the constituent factors (i.e. by setting them to different non-zero values). Therefore, if in each factor we set all its non-zero probabilities to a uniform value—we call this a (partially) uniform value, since the zero probabilities remain unaffected—the *size* of the factor product's sparse probability table will remain unchanged.
- Secondly, we notice that after making these factors (partially) uniform, the resultant factor product will also be (partially) uniform. All its non-zero probabilities will have the same fixed value.
- Thus, if we can calculate the entropy of this (partially) uniform factor product, then with Equation (5) we can calculate the number of non-zero probabilities in its sparse probability table.
- Algorithm 2 estimates this required entropy value, even before performing the factor merging, i.e. without multiplying out all the factors to get the factor product.

3.3. Limitations and Assumptions

Loopy inference inherently introduces error in the entropy estimation. After message passing, the probabilities are more extreme or overconfident (due to feedback loops) [3, p. 429], which causes the

 doi:10.20944/preprints202605.2092.v1

Algorithm 2 Estimating the sparse probability table size for the joint of a cluster of factors.

Require: The subset of factors whose joint probability table size is needed.
Graph construction:
1: Make all factors (partially) uniform by setting the non-zero probabilities to a fixed uniform value.
2: Build a CG from these factors. Note: the fewer loops in this PGM, the more accurate the result will be. A factor graph will result in the least accurate size estimate, whereas a JT (which is typically not feasible to construct) will provide an exact value. We choose to use the LTRIP algorithm for building the graph, which balances accuracy with computational tractability.
Total entropy of joint:
3: Estimate the entropy using Algorithm 1, with the cluster group C' equal to the subset of factors whose joint probability table size is needed.
Table size of joint:
4: Determine the estimated number of non-zero probabilities in the joint distribution using Equation (5).

individual cluster and sepset entropies to be underestimated. Since we calculate the total entropy by subtracting sepset entropies from cluster entropies, we face two possibilities for error: the underestimated cluster entropies reduce the total, while the underestimated sepset entropies (being subtracted) increase it. The net result depends on which effect dominates—if the clusters dominate, the total entropy is underestimated; if the sepsets dominate, the total entropy is underestimated. Thus, like loopy inference, our entropy estimates are generally good but remain approximate.

4. Experiments and Results

To evaluate the quality of the entropy estimations returned by Algorithm 1, we perform two sets of experiments using distinct sets of models. The first experiment (Section 4.1) evaluates the algorithm on a large cache of LDPC codes, which feature binary variables and the same variable count across all models. The second experiment (Section 4.2) demonstrates generalisation to randomly generated models with diverse characteristics: mixed variable domains (binary and ternary), varying model sizes (in terms of variable count, cluster count, and connectivity), and less structured probability distributions. In Section 4.3 we validate the scalability of the algorithm on a substantially larger model.

In the first two experiments, the following methodology is used per model:

- *Graph construction:* An initial CG is created using LTRIP [4] and, if the model size allows it, a JT is constructed for ground truth validation.
- *Message passing:* Message passing is performed using the Lauritzen-Spiegelhalter algorithm [17] with an asynchronous sweep schedule [3, p. 408]. The convergence criteria are sepset belief calibration (no sepset beliefs change between the forward and backward sweep, i.e. messages in opposite direction are in agreement) and variable value agreement (all clusters agree on the maximum likelihood assignment).
- *Measurements:* We only consider cases where message passing reaches convergence according to the above criteria. For various configurations of cluster groups $\boldsymbol{C}'$, we measure the estimated entropy $\hat{H}(\boldsymbol{C}')$ and, where the models are sufficiently small to allow for JT construction, the true entropy $H(\boldsymbol{C}')$. The entropy estimation error (that is, the deviation from the ground truth entropy) is then calculated as

$$\delta_{\hat{H}(\boldsymbol{C}')} = \hat{H}(\boldsymbol{C}') - H(\boldsymbol{C}'). \tag{15}$$

We use the (direct) entropy estimation error, as it provides a consistent measure across both near-zero and large entropy values. Relative error would inflate artificially as the true entropy approaches zero.

 doi:10.20944/preprints202605.2092.v1

All experiments were performed on a desktop PC with an Intel i7-13700K CPU (24 cores) and 64 GB DDR5 RAM running Ubuntu 20.04. A replication package is available on GitHub [20], built on top of the C++ PGM library `EMDW` [21].

4.1. Experiment 1: LDPC Codes

We make use of low density parity check (LDPC) codes as models for our first experiment. Below we provide a minimal description of LDPC codes, only sufficient to enable understanding how we use them in the context of PGMs and entropy. For more detail the reader is referred to Johnson's introduction [22].

4.1.1. Description

LDPC codes are used in digital communication systems for error correction. Transmission over a noisy channel may introduce bit errors that corrupt the transmitted message sequence. LDPC codes work by appending check bits (an encoding of the message bits) to form the transmitted sequence. This allows the receiver to decode the received sequence and correct bit errors. An LDPC code comprises a set of even-parity constraints, specifying which combination of bits need to sum (modulo 2) to 0. These constraints can be implemented as factors in a PGM, which we use to construct a CG. The PGM is defined over two sets of random variables: the (unknown) transmitted bits and the (observed) received bits. Each cluster in the CG corresponds to an even-parity constraint, and the sepsets depend on the choice of graphical structure.

If a code has k message bits, $n-k$ check bits, and minimum Hamming distance (MHD) $d_{\min}$, it is denoted an $(n, k, d_{\min})$ code. The MHD is directly proportional to the error correction capability, i.e. the number of errors the code can fix under perfect inference (approximate inference may occasionally cause decoding failures), through [22, p. 9]

$$d_{\text{fix}} = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor. \tag{16}$$

In the face of d_{fix} or fewer errors, LDPC codes are designed to decode the received sequence to the original transmitted sequence. In these cases, the decoded transmitted sequence has high probability (close to one), corresponding to a low entropy (close to zero)—a measurable property, useful for evaluating the quality of our entropy estimation in scenarios where the model complexity no longer allows comparison to ground truth. We highlight this in the experiment below, as well as in the demonstration in Section 4.3.

4.1.2. Experimental Setup

This experiment makes use of a cache of 26,903 unique $(24, 12, 5)$ LDPC codes [23], each consisting of 12 message bits and 12 check bits with an error correction capability of 2 bits, and each small enough to permit the construction of a JT. Every model is evaluated across eight error conditions, with $0, 1, \ldots,$ or 7 corrupted bits in the transmitted sequence. For each condition, entropy is estimated for four selected cluster group configurations (with the cluster indices randomly selected): a single cluster ($\boldsymbol{C}' = \{C_i\} \mid i \in \mathcal{V}$), a cluster pair ($\boldsymbol{C}' = \{C_i, C_j\} \mid i, j \in \mathcal{V}$), a cluster trio ($\boldsymbol{C}' = \{C_i, C_j, C_k\} \mid i, j, k \in \mathcal{V}$), and all clusters ($\boldsymbol{C}' = \boldsymbol{C}$).

4.1.3. Results

In Figure 7, box and whisker plots of the entropy estimation error $\delta_{\hat{H}(\boldsymbol{C}')}$ (Equation (15)) for the cache of $(24, 12, 5)$ LDPC codes are shown. There is commonality across the cluster group configurations based on the number of corrupted bits, with three clear regimes.

1. At most $d_{\text{fix}} = 2$ errors: the cases of 0 and 1 corrupted bits have no visible box, and the case of 2 corrupted bits has a small interquartile range, centred close below zero. This is consistent with

 doi:10.20944/preprints202605.2092.v1

the low-entropy regime discussed above: when decoding succeeds, both the estimated and true entropy are themselves near zero.

2. 3–4 errors: the boxes grow substantially, indicating increasing variance in the entropy estimation error as the number of corrupted bits exceed the correction capability. Both the JT and CG may still occasionally decode correctly.
3. 5 or more errors: a clear upward trend in the entropy estimation error as the number of corrupted bits increase. This directly reflects the inability to decode the received sequence—the more errors we introduce, the more competing hypotheses there are.

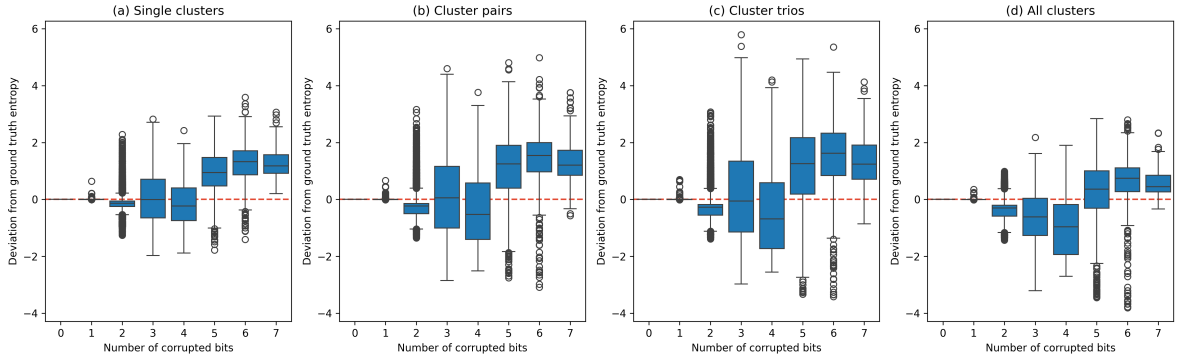

Figure 7. Box and whisker plots of the entropy estimation error for the cache of $(24, 12, 5)$ low density parity check (LDPC) codes.

Figure 8 shows 30-bin histograms of the entropy estimation errors for the cache of $(24, 12, 5)$ LDPC codes. Across the cluster group configurations, the medians and modes are very close to zero, and the plots are skewed to the left. This indicates that, on average, the estimated entropy is lower than the true entropy (which aligns with our discussion in Section 3.3). Each of the $16\% - 84\%$ percentile intervals (PIs) (68% of the data, similar to one standard deviation from the mean) is very narrow, with the 84% PI coinciding with the median almost exactly. While the $2.5\% - 97.5\%$ PIs (95% of the data, similar to two standard deviations from the mean) are wider, they remain confined to approximately $[-1, 1]$. To contextualise the scale of the errors: for a model defined over 24 binary random variables, the maximum joint entropy occurs when the distribution is uniform, yielding $H = \log_2(2^{24}) = 24$ bits. An entropy estimation error within ± 1 (less than 5%), therefore, indicates good accuracy. The PIs, along with the Pearson correlation coefficients between the estimated and true entropies, are reported in Table 1, and further validate the quality of our entropy estimates.

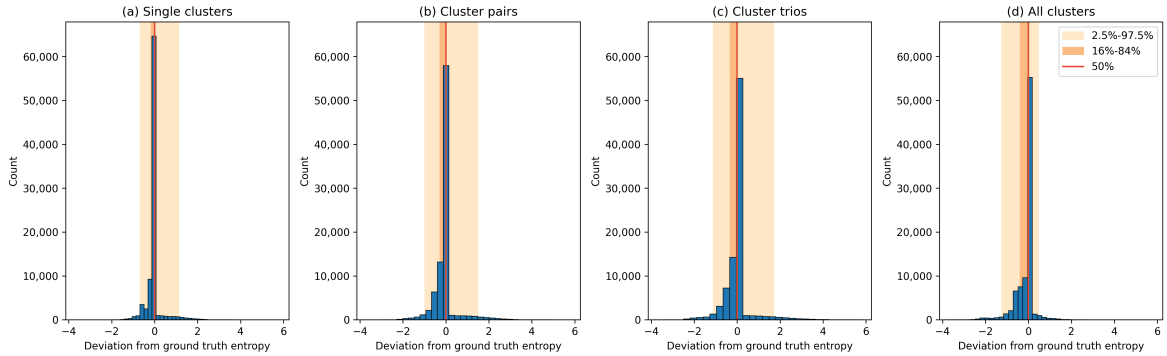

Figure 8. Histograms of the entropy estimation error for the cache of $(24, 12, 5)$ LDPC codes.

 doi:10.20944/preprints202605.2092.v1

Table 1. Entropy estimation error percentile intervals (PIs) and Pearson correlation coefficients for the cache of $(24, 12, 5)$ low density parity check (LDPC) codes.

Cluster group configuration	Percentile intervals			Pearson correlation coefficient
	50%	**[16%, 84%]**	**[2.5%, 97.5%]**	
Single clusters	-0.0015	$[-0.16, 0.0]$	$[-0.68, 1.1]$	0.88
Cluster pairs	-0.0021	$[-0.27, 0.0]$	$[-0.98, 1.5]$	0.91
Cluster trios	-0.0030	$[-0.33, 0.0]$	$[-1.1, 1.7]$	0.92
All clusters	-0.0033	$[-0.38, 0.0]$	$[-1.2, 0.47]$	0.94

4.2. Experiment 2: Random Models

To demonstrate the generalisability of the entropy estimation algorithm, we consider for the next experiment a cache of randomly generated models.

4.2.1. Description

A key difference between these models and those in the previous experiment is that here we have random factors (instead of even-parity factors). We further have a combination of binary and ternary variables (instead of exclusively binary variables). The number of variables and clusters also vary. The models are generated by uniform random selection of the number of variables, number of clusters, variable sparsity (proportion of clusters a variable belongs to) and probability sparsity (how many of the assignments have a non-zero probability). Finally, each of the non-zero probabilities are (uniformly) randomly generated, and normalised to ensure a valid probability distribution.

4.2.2. Experimental Setup

This experiment makes use of 5498 randomly generated models, each small enough to permit the construction of a JT. Each model is randomly parameterised, with the number of variables drawn from $[10, 21]$, the number of clusters from $[5, 10]$, the variable sparsity from $[0.1, 0.6]$, and the probability sparsity from $[0.2, 0.8]$. For each random model, we again estimate the entropy for four cluster group configurations (with the cluster indices randomly selected): a single cluster ($\boldsymbol{C}' = \{C_i\} \mid i \in \mathcal{V}$), a cluster pair ($\boldsymbol{C}' = \{C_i, C_j\} \mid i, j \in \mathcal{V}$), a cluster trio ($\boldsymbol{C}' = \{C_i, C_j, C_k\} \mid i, j, k \in \mathcal{V}$), and all clusters ($\boldsymbol{C}' = \boldsymbol{C}$).

4.2.3. Results

In Figure 9, 30-bin histograms of the entropy estimation error $\delta_{\hat{H}(\boldsymbol{C}')}$ for the cache of random models are shown. Across all configurations, the modes are close to zero, and the medians are close to (but slightly above) zero. Further, the plots are skewed to the right, indicating that, on average, the entropy estimation is larger than the true entropy. The 16%–84% PIs vary across the configuration, with the narrowest being the single clusters, followed by all clusters, cluster pairs, and cluster trios the widest. The 2.5%–97.5% PIs follow a similar order: all clusters is the narrowest, then single clusters, cluster pairs, and cluster trios the widest. Over all configurations, the entropy estimation error is confined to approximately $[-1, 2]$, again demonstrating good accuracy, and confirming the applicability of the entropy estimation algorithm to a selection of more general models.

The PIs, along with the Pearson correlation coefficients between the estimated and true entropies, are reported in Table 2.

 doi:10.20944/preprints202605.2092.v1

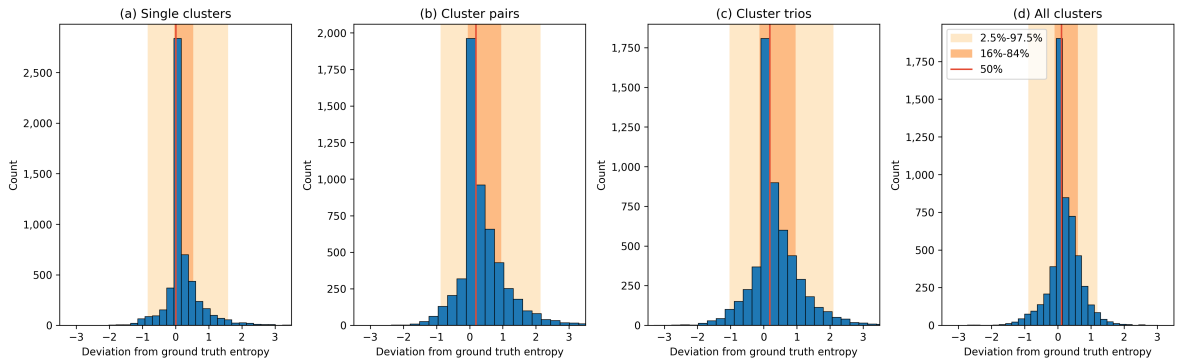

Figure 9. Histograms of the entropy estimation error for the cache of random models.

Table 2. Entropy estimation error PIs and Pearson correlation coefficients for the cache of random models.

Cluster group configuration	Percentile intervals			Pearson correlation coefficient
	50%	**[16%, 84%]**	**[2.5%, 97.5%]**	
Single clusters	0.0057	$[-0.031, 0.52]$	$[-0.82, 1.6]$	0.96
Cluster pairs	0.19	$[-0.042, 0.93]$	$[-0.87, 2.1]$	0.96
Cluster trios	0.18	$[-0.110, 0.94]$	$[-1.00, 2.1]$	0.97
All clusters	0.11	$[-0.083, 0.59]$	$[-0.87, 1.2]$	0.99

4.3. Experiment 3: Scaling to a Larger PGM

In the previous two experiments we compared estimated entropies against the ground truth values provided by a JT. However, the exponential blow-up in JT probability table sizes restricts this to fairly small models. It turns out that the computational requirement grows roughly exponentially with the scope of the largest cluster in the PGM (see Appendix B for details). This motivated us to demonstrate at least one substantially larger example, where a loopy-structured graph in combination with restricted cluster sizes makes the algorithm feasible.

In the absence of a JT, we need some other indication of the entropy estimation quality, and LDPC codes present a possible avenue for this. From Section 4.1.1, we know that a received message with d_{fix} or fewer errors should, with perfect inference, decode to the correct sequence, i.e. the transmitted sequence. Thus, the decoded transmitted sequence should have high probability, corresponding to a low total entropy.

To demonstrate the scalability of the entropy estimation algorithm, we use a $(2640, 1320, \sim 220)$ LDPC code [24, p. 30]. This code consists of 2640 binary random variables (1320 message bits and 1320 check bits), across 1320 parity check factors. From the number of message bits we calculate the number of valid transmitted sequences to be $2^{1320} \approx 10^{397}$. The exponentially larger space of possible received sequences consists of $2^{2640} \approx 10^{795}$ sequences. The MHD of this code is not known, but through extensive testing we have determined 220 to be an upper bound on the MHD, giving an error correction capability of $d_{\text{fix}} = 109$ corrupted bits per transmitted sequence of 2640 bits. For received sequences with at most 109 corrupted bits, inference should assign high probability to the correct transmitted sequence, corresponding to a near-zero total entropy.

To confirm this, we applied Algorithm 1 to $10,000$ simulated received sequences for each of $0, 5, 10, \ldots, 105$ and 109 corrupted bits. The resultant entropy estimates for the configuration of all clusters ($\boldsymbol{C}' = \boldsymbol{C}$) are distributed in the range $[-10^{-18}, 0]$. This is within round-off error of the zero entropy we expect.

5. Discussion

The results of the three experiments suggest that Algorithm 1 yields good quality entropy estimations. In Experiments 1 and 2, we demonstrate the algorithm's accuracy on models with up to 24

 doi:10.20944/preprints202605.2092.v1

variables, with errors remaining within tight bounds across diverse model types. We also demonstrate, using a substantially larger model (defined over 2640 random variables) that entropy estimation remains viable, provided that we are able to perform inference on the model.

Our entropy estimation algorithm inherits the limitations of loopy inference. However, the experimental results validate the core assumption that good inference leads to accurate entropy estimation.

Whereas purge-and-merge [4] and other factor clustering methods make use of worst-case approximations (maximal entropy) we obtain a much closer approximation without the typical computational cost of factor multiplication, which allows for improved cluster merging decisions. The entropy estimation algorithm can be incorporated into various algorithms for static and dynamic graph restructuring, as well as entropy-based applications in other PGM contexts.

Some possible avenues for future work include extensions to continuous random variables or mixtures, investigating the effect of cluster group size on accuracy, and investigating performance on even larger models.

Author Contributions: Conceptualization, K.M.S. and J.A.d.P.; methodology, K.M.S. and J.A.d.P.; software, K.M.S. and J.A.d.P.; writing—original draft preparation, K.M.S.; writing—review and editing, J.A.d.P. and W.B.; visualization, K.M.S.; supervision, J.A.d.P. and W.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: A replication package is available on https://doi.org/10.5281/zenodo.20417153.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this paper:

CG	Cluster graph
JT	Junction tree
LDPC	Low density parity check
LTRIP	Layered trees running intersection property
MHD	Minimum Hamming distance
PGM	Probabilistic graphical model
PI	Percentile interval
RIP	Running intersection property

Appendix A. THE layered Trees Running Intersection Property (LTRIP) Algorithm [4]

In Algorithm A1, the LTRIP algorithm is set out. We also illustrate some steps in the algorithm with a small example, shown in Figure A1. The distance measure we use in step 1 of the algorithm is the modified Jaccard distance:

$$d_{C_i,C_j} = -\log \frac{|C_i \cap C_j|}{\min(|C_i|, |C_j|)}, \tag{A1}$$

where $|C_i|$ is the number of variables in cluster C_i. This emphasises large (relative) overlap between adjacent clusters.

Since each individual layer is a tree structure, the final graph is guaranteed to not contain cycles in any particular variable, thereby satisfying the RIP criterion.

 doi:10.20944/preprints202605.2092.v1

Algorithm A1 The LTRIP algorithm

1: Specify a distance measure d_{C_i,C_j} that describes how closely two clusters associate.
2: For a given set of clusters, compile a set $\mathcal{V}$ of all the variables in them. For our running example, $\mathcal{V} = \{v_0, v_1 \dots v_4\}$.
3: **for** each $v \in \mathcal{V}$, in separate layers, construct a *tree* as follows:
4: Using d_{C_i,C_j}, specify an edge weight between all pairs of clusters containing v.
5: Use the Prim-Jarník algorithm [25] to construct a minimum spanning tree connecting all clusters containing v. In our running example, Figures A1(**a**)–A1(**c**) each shows one such tree (in total there will be five such layers, corresponding to the five random variables).
6: **end for**
7: Overlay these layers (each a tree) and merge the sepsets occurring between corresponding clusters. For our running example this produces the graph in Figure A1(**d**).

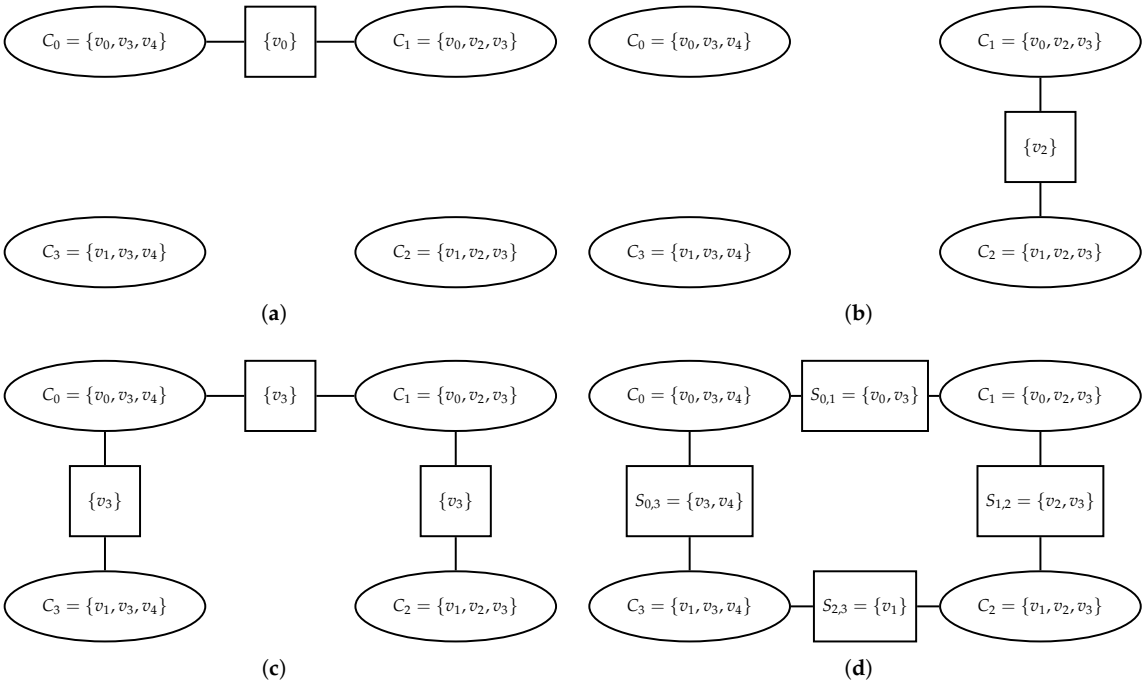

Figure A1. How the LTRIP algorithm creates a cluster graph (CG) by overlaying layered tree structures. (**a**) The v_0 layer. (**b**) The v_2 layer. (**c**) The v_3 layer. Note that the tree structure precludes a v_3 link between C_2 and C_3, which would have violated the running intersection property (RIP) by causing a cycle in v_3. (**d**) The overlay of all the layers in the LTRIP algorithm. Note the reduced sepset between C_2 and C_3 which ensures the RIP.

Appendix B. Computational Considerations

To determine the computational cost of estimating the entropy in a graph, we distinguish between the message passing and the entropy calculation steps. Let A_{C_i} be the number of non-zero probabilities for cluster C_i, and similarly $A_{S_{i,j}}$ for sepset $S_{i,j}$.

Appendix B.1. Message passing (graph calibration)

Passing a belief update message [3, Sec. 10.3] from cluster C_i to cluster C_j, involves a sepset belief update

$$\psi'_{i,j} = \sum_{\setminus S_{i,j}} \Psi_i, \tag{A2}$$

 doi:10.20944/preprints202605.2092.v1

where $\backslash S_{i,j}$ indicates we marginalise out all variables not in $S_{i,j}$, and a cluster belief update

$$\Psi_j' = \Psi_j \frac{\psi_{i,j}'}{\psi_{i,j}}. \tag{A3}$$

The marginalisation in Equation (A2) requires $A_{C_i} - A_{S_{i,j}}$ additions [3, p. 297]. Note that a sepset is always a subset of both adjacent clusters. Further, Equation (A3) requires A_{C_j} multiplications and $A_{S_{i,j}}$ divisions [3, p. 107]. Combined, therefore, a single belief update message from C_i to C_j requires $A_{C_i} + A_{C_j}$ arithmetic operations.

To find the total cost of making one belief update sweep over all the edges in the graph, we must sum this cost over all edges present. In this, a cluster C will appear $\mathrm{degr}_{\mathrm{CG}}(C)$ times, where $\mathrm{degr}(C)$ denotes the number of edges of cluster C. From this we get the total computational cost (number of arithmetic operations) of a single message passing sweep over the cluster graph as:

$$\chi_{\mathrm{CG}} = \sum_i \mathrm{degr}_{\mathrm{CG}}(C_i) A_{C_i}. \tag{A4}$$

The number of sweeps to reach convergence is not known. When message passing converges, it typically takes fewer than some tens of sweeps, but it may not converge at all. To manage this, we set a maximum number of sweeps before assuming non-convergence. This limits the total number of arithmetic operations to be proportional to the single sweep of Equation (A4).

Appendix B.2. Entropy Calculation

From Equation (4) we see that the entropy calculation at a node (cluster or sepset) consisting of N non-zero probabilities requires N log operations, N multiplications, and $N - 1$ additions. The log operations will dominate, with the multiplication and addition comparatively negligible. Further, the contributions from the sepset terms in Equation (9) are dwarfed by the exponentially larger cluster contributions. Thus, counting only the dominant log operations and only the cluster terms, the computational cost (number of log operations) of the entropy calculation is approximately

$$\chi_{H_{\mathrm{CG}}} \approx \sum_i A_{C_i}. \tag{A5}$$

Appendix B.3. Computational Cost Comparison

Although log operations are more expensive than the primitive arithmetic operations involved in message passing [26], the entropy calculation is performed only once on the calibrated beliefs. In our experience, the computational cost of message passing—and the multiple sweeps being necessary for convergence—will dominate the entropy calculation.

References

1. Yedidia, J.; Freeman, W.; Weiss, Y. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory* **2005**, *51*, 2282–2312.
2. Kschischang, F.; Frey, B.; Loeliger, H.A. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory* **2001**, *47*, 498–519.
3. Koller, D.; Friedman, N. *Probabilistic Graphical Models: Principles and Techniques*; MIT Press, 2009.
4. Streicher, S.; du Preez, J.A. Graph Coloring: Comparing Cluster Graphs to Factor Graphs. In Proceedings of the Proceedings of the ACM Multimedia 2017 Workshop on South African Academic Participation, New York, NY, USA, 2017; SAWACMMM '17, pp. 35–42.
5. Aji, S.; McEliece, R. The Generalized Distributive Law. *IEEE Transactions on Information Theory* **2000**, *46*, 325–343.
6. Mateescu, R.; Kask, K.; Gogate, V.; Dechter, R. Join-Graph Propagation Algorithms. *The Journal of Artificial Intelligence Research* **2010**, *37*, 279–328. https://doi.org/10.1613/jair.2842.

 doi:10.20944/preprints202605.2092.v1

7. Streicher, S.F.; du Preez, J.A. Strengthening Probabilistic Graphical Models: The Purge-and-Merge Algorithm. *IEEE Access* **2021**. https://doi.org/10.1109/ACCESS.2021.3124760.
8. Moral, S.; Cano, A.; Gómez-Olmedo, M. Computation of Kullback–Leibler Divergence in Bayesian Networks. *Entropy* **2021**, *23*, 1122. https://doi.org/10.3390/e23091122.
9. Scutari, M. Entropy and the Kullback–Leibler Divergence for Bayesian Networks: Computational Complexity and Efficient Implementation. *Algorithms* **2024**, *17*, 24. https://doi.org/10.3390/a17010024.
10. Darwiche, A. *Modeling and Reasoning with Bayesian Networks*; Cambridge University Press, 2009.
11. Bishop, C.M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*, first ed.; Springer, 2007.
12. McEliece, R.J.; Yildirim, M. Belief Propagation on Partially Ordered Sets. In Proceedings of the Mathematical Systems Theory in Biology, Communications, Computation, and Finance; Rosenthal, J.; Gilliam, D.S., Eds., New York, NY, 2003; pp. 275–299.
13. Frey, B.J.; Kschischang, F.R.; Loeliger, H.A.; Wiberg, N. Factor graphs and algorithms. In Proceedings of the Proceedings of the Annual Allerton Conference on Communication Control and Computing. University of Illinois, 1997, Vol. 35, pp. 666–680.
14. Tanner, R. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory* **1981**, *27*, 533–547. https://doi.org/10.1109/TIT.1981.1056404.
15. Tomlinson, M.; Tjhai, C.J.; Ambroze, M.A.; Ahmed, M.; Jibril, M. *Error-correction coding and decoding: Bounds, Codes, decoders, analysis and applications*, 1st ed.; Springer Nature: Cham, Switzerland, 2017.
16. Shafer, G.R.; Shenoy, P.P. Probability Propagation. *Annals of Mathematics and Artificial Intelligence* **1990**, *2*, 327–351.
17. Lauritzen, S.L.; Spiegelhalter, D.J. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society. Series B, Methodological* **1988**, *50*, 157–224.
18. MacKay, D.J. *Information theory, inference and learning algorithms*; Cambridge University Press: Cambridge, UK, 2003.
19. Cover, T.M.; Thomas, J.A. *Elements of Information Theory*; Wiley-Interscience: USA, 2006.
20. Sadie, K.; du Preez, J. Replication package. https://github.com/emdw-contributor/emdw, 2026.
21. contributors, T.E. EMDW. https://github.com/emdw-contributor/emdw, 2026.
22. Johnson, S.J. Introducing low-density parity-check codes. *University of Newcastle, Australia* **2006**, *1*.
23. contributors, T.E. EMDW. https://github.com/emdw-contributor/emdw, 2026.
24. Li, J.; Lin, S.; Abdel-Ghaffar, K.; Ryan, W.E.; Costello, D.J. *LDPC code designs, constructions, and unification*, 1st ed.; Cambridge University Press: New York, NY, USA, 2017.
25. Prim, R.C. Shortest connection networks and some generalizations. *The Bell System Technical Journal* **1957**, *36*, 1389–1401. https://doi.org/10.1002/j.1538-7305.1957.tb01515.x.
26. Fog, A. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. https://www.agner.org/optimize/instruction_tables.pdf, 2025.