Shapeshifter: A Dynamic Graph-Restructuring Algorithm for Improved Inference in Loopy-Structured PGMs

Katerine M. Sadie 21639361@SUN.AC.ZA
Johan A. du Preez DUPREEZ@SUN.AC.ZA
Willie Brink WBRINK@SUN.AC.ZA
Stellenbosch University, Stellenbosch, South Africa

Editors: Gustau Camps-Valls, Manuele Leonelli and Gherardo Varando

Abstract

Inference in probabilistic graphical models (PGMs) is sensitive to the choice of graph structure used for message passing. While junction trees allow for exact inference, their use is practically infeasible due to exponential blow-up in cluster sizes. Loopy-structured PGMs, like factor graphs or cluster graphs, are computationally tractable alternatives, but the gap between the performance of such graphs and the ideal performance of junction trees still exists. We propose SHAPESHIFTER, an algorithm that dynamically restructures a cluster graph during inference, by merging cluster pairs based on disagreement and redundancy criteria. We evaluate on two distinct domains: error correction and constraint satisfaction. The results show that SHAPESHIFTER offers a practical middle ground: improved inference when compared to existing PGM-based approaches—closing the performance gap by a substantial margin—without the prohibitive computational expense of junction trees.

Keywords: Probabilistic graphical models, dynamic graph structure, inference

1. Introduction

Probabilistic graphical models (PGMs) have been used to represent complex systems and perform inference in various domains (Wright, 1921; Gallager, 1962; Geman and Geman, 1984; Rabiner, 1989; Heckerman et al., 1992). While there are a myriad of graph structures that can be used for inference, the quality of said inference is sensitive to the choice of graph structure (Yedidia et al., 2005). Junction trees allow for exact inference but suffer from exponential blow-up. Loopy-structured PGMs are computationally tractable alternatives but are not guaranteed to converge, and when they do, may yield suboptimal results (Koller and Friedman, 2009, ch. 11). In this paper we are concerned with a specific class of PGMs called cluster graphs (CGs), which includes factor graphs[1] (FGs) (Kschischang et al., 2001), general CGs (Koller and Friedman, 2009, p. 346) and junction trees (JTs) (Aji and McEliece, 2000). We will also restrict our attention to discrete random variables. Given a set of factors, some graph structures—such as FGs and CGs constructed using LTRIP (Streicher and du Preez, 2017)—have a one-to-one correspondence between the original factors and the clusters of the graph. Others—such as CGs constructed using the join-graph structuring algorithm (Mateescu et al., 2010) and JTs—group various factors together in the clusters of the graph. The purge-and-merge algorithm (Streicher and du Preez, 2021) pairs factor grouping with repeated graph construction and inference, and is most effective when the

[1] Not to be confused with the unrelated factor tree structure (Bloemeke and Valtorta, 1998).

© 2026 K.M. Sadie, J.A. du Preez & W. Brink.

set of valid assignments is small. At each iteration, message passing is performed, zero-probability variable assignments are extracted from the beliefs to update the original factors, and a completely new CG is constructed, resetting the inference process.

Methods that involve factor merging, such as join-graph structuring and purge-and-merge, use the number of variables in the resultant joint probability distribution as a metric to bound the resulting probability table size. Instead of the number of variables, we make use of the sparse probability table size estimation algorithm of Sadie et al. (2026) to estimate the number of non-zero probabilities in a joint probability distribution *without* explicitly multiplying out the factors. Building on this, we propose SHAPESHIFTER, an algorithm that dynamically restructures any valid CG through cluster merging. In a single application, SHAPESHIFTER identifies candidate cluster pairs belonging to one of two categories, both subject to a size constraint on the number of non-zero probabilities in the joint. The first category targets pairs in disagreement, and the second targets pairs with substantial redundancy. SHAPESHIFTER is intended to be applied repeatedly during inference, alternating with rounds of message passing; merges are performed such that message passing can continue seamlessly on the dynamically updated graph. Our evaluations indicate that this approach offers a practical middle ground: substantially improved inference quality when compared to other approaches on the same problems, without the prohibitive computational cost of exact inference.

The remainder of the paper is structured as follows. In Section 2 we cover some preliminaries and notation. In Section 3 we present our graph restructuring algorithm, and demonstrate the graphical effects it has on a small example. In Section 4 we evaluate the inference quality of the dynamic graph structure on two distinct domains: error correction (channel decoding) and constraint satisfaction (Sudoku puzzles). Finally, in Section 5 we summarise our results.

2. Background and Notation

A probabilistic graphical model (PGM) consists of a set of probability distributions and a graphical representation of the dependency structure. A node represents one or more variables, and an edge represents some probabilistic relationship between the two nodes it connects (Bishop, 2007).

Let $\boldsymbol{X}$ be a set of random variables that can take on values $\boldsymbol{x} \in \mathcal{X}$. A factor $\phi(\boldsymbol{X})$ is a function mapping $\boldsymbol{x}$ to the non-negative reals (Koller and Friedman, 2009, p. 104), with $\boldsymbol{X}$ its scope. While there are various choices for how factors can be connected to one another in a PGM, we restrict our attention to cluster graphs (CGs).

A CG (also known as a junction or join graph) (Koller and Friedman, 2009, p. 346; McEliece and Yildirim, 2003, p. 281; Mateescu et al., 2010) is an undirected graph defined for a set of factors $\boldsymbol{\Phi}$ over a set of variables $\boldsymbol{X}$, subject to the following: a node i corresponds to a cluster $C_i \subseteq \boldsymbol{X}$, and an edge (i, j) with $i < j$ corresponds to a separation set (sepset) $S_{i,j} \subseteq \boldsymbol{X}$. Further, a CG must adhere to family preservation, which requires that each factor $\phi_k(\boldsymbol{X}_k) \in \boldsymbol{\Phi}$ belongs to exactly one cluster C_i such that $\text{scope}[\phi_k] = \boldsymbol{X}_k \subseteq C_i$. Lastly, the running intersection property (RIP) must hold, which requires that for any variable X with $X \in C_i$ and $X \in C_j$, there is exactly one path between C_i and C_j that contains X in

every sepset along the path. Equivalently, the subgraph induced by any variable X (that is, formed by retaining only the clusters and sepsets containing X) is a tree.

We use $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to denote a CG $\mathcal{G}$ with nodes $\mathcal{V}$ and edges $\mathcal{E}$. The clusters are denoted $\boldsymbol{C} = \{C_i \text{ with } i \in \mathcal{V}\}$, and sepsets $\boldsymbol{S} = \{S_{i,j} \text{ with } (i,j) \in \mathcal{E}\}$. Given a spanning tree of the graph, adding any edge not in the tree closes the unique path already connecting its two nodes into a cycle—called a fundamental cycle. These form a (non-unique) basis for the graph's cycle space, in that every cycle can be expressed in terms of them (Gross and Yellen, 2005). The number of fundamental cycles in a connected graph with $|\mathcal{V}|$ nodes and $|\mathcal{E}|$ edges is (Gross and Yellen, 2005)

$$n_{\text{cycles}} = |\mathcal{E}| - |\mathcal{V}| + 1. \tag{1}$$

Graphs with more and shorter cycles tend to exhibit poorer inference quality (Weiss, 2000).

There are three types of CGs relevant to this paper. A factor graph (FG), also referred to as a Tanner or Bethe graph (Kschischang et al., 2001; Bishop, 2007, p. 360; Frey et al., 1997; Tanner, 1981; Tomlinson et al., 2017, p. 315; Koller and Friedman, 2009, p. 405) is bipartite, with two node categories: factor nodes and variable nodes. Each factor node corresponds to a factor, and each variable node is connected to all factor nodes whose scope includes the variable. FGs have single-variable sepsets, which may also hamper inference. General CGs, such as those constructed using the join-graph structuring algorithm (Mateescu et al., 2010) or LTRIP (Streicher and du Preez, 2017), generalise FGs by removing the limitation on the sepset size: each cluster corresponds to one or more factors, and sepsets are typically multivariate. A junction tree (JT) (Koller and Friedman, 2009, p. 348; Aji and McEliece, 2000, p. 330) is a CG whose graph structure is a tree. While inference is exact in JTs, they suffer from exponential blow-up in cluster sizes, and can only be used in select cases where the model is sufficiently small.

Message passing is an iterative process that refines the cluster beliefs $\boldsymbol{\Psi} = \{\Psi_i \text{ with } i \in \mathcal{V}\}$ and sepset beliefs $\boldsymbol{\psi} = \{\psi_{i,j} \text{ with } (i,j) \in \mathcal{E}\}$, as a means to obtain marginal and conditional distributions over the variables of the model. These beliefs are simply a reparameterisation of the original distribution defined by the set of factors $\boldsymbol{\Phi}$ over the set of variables $\boldsymbol{X}$, through

$$\tilde{p}(\boldsymbol{X}) = \prod_{\phi_k \in \boldsymbol{\Phi}} \phi_k(\boldsymbol{X}_k) = \frac{\prod_{i \in \mathcal{V}} \Psi_i(C_i)}{\prod_{(i,j) \in \mathcal{E}} \psi_{i,j}(S_{i,j})}, \tag{2}$$

with $\tilde{p}(\boldsymbol{X})$ the unnormalised joint probability distribution. There are various approaches to inference through message passing (Lepar and Shenoy, 1998), and we choose to use the Lauritzen–Spiegelhalter (belief update) algorithm (Lauritzen and Spiegelhalter, 1988). When performed on a loopy-structured CG, convergence is not guaranteed (Koller and Friedman, 2009, ch. 11). We say a CG is calibrated once every pair of connected clusters agrees on their shared sepset's belief, and has reached assignment agreement once every cluster agrees on the most probable assignment.

3. Shapeshifter

The SHAPESHIFTER algorithm works by merging cluster pairs, thereby reducing the number of fundamental cycles in the graph, with the aim of improving inference quality: Section 3.1 covers which clusters to merge, Section 3.2 the mechanics of a single merge, Section 3.3

the formal algorithm, and Section 3.4 a detailed example. Note that SHAPESHIFTER, as formalised in Section 3.3, describes a single application: one pass over the current CG that identifies and performs a set of cluster merges. In practice, SHAPESHIFTER is applied repeatedly, alternating with rounds of message passing; the number of sweeps between applications is a tunable parameter, and we find that a small number (e.g. 3 to 5) works well.

3.1. Which Clusters to Merge

Candidate cluster pairs fall into one of two categories. *Disputing* merges target connected pairs that are largely uncalibrated, i.e. those where the Kullback–Leibler divergence between the normalised cluster beliefs, marginalised to their shared sepset, exceeds a specified threshold δ_D. Intuitively, such clusters are in dispute about the sepset belief, and merging them resolves the disagreement directly. *Structural* merges target overlapping pairs—connected or not—where the increase in entropy (computed over the clusters' normalised beliefs) from merging, relative to the larger of the two cluster entropies, falls below a specified threshold δ_H. This measures redundancy between the two clusters: a small increase means the less-informative (lower entropy) cluster's information is already largely captured by the more-informative (higher entropy) one. Formally, this increase is the conditional entropy of the less-informative cluster given the more-informative one, i.e. the uncertainty that remains once the other's assignment is known.

Candidate pairs in both categories are subject to a size constraint—the factor product resulting from the merge must remain sufficiently small for computational tractability—where the number of non-zero probabilities in the joint must fall below a specified threshold δ_N. Note that this bounds the joint's count of non-zero probabilities directly, rather than the number of variables in a cluster: a merged cluster may therefore grow arbitrarily wide, provided the resulting joint remains sufficiently sparse. In practice, δ_N is a tunable, system-dependent parameter, chosen according to the memory available for storing a cluster's belief. Crucially, both the number of non-zero probabilities and the entropy of the resulting joint can be estimated without explicitly computing the factor product (Sadie et al., 2026).

Candidates from each category are greedily selected after sorting: each pair is considered in turn, and accepted unless either of its clusters already appears in a previously accepted pair. Disputing pairs are sorted by descending Kullback–Leibler divergence, prioritising pairs with the most disagreement about their shared variables. Structural merges are sorted first by the number of edges eliminated (descending), then by entropy increase (ascending). The number of edges eliminated is equal to the number of clusters neighbouring *both* clusters in the pair, plus 1 if the pair itself is connected in the graph. This ordering directly targets fundamental cycle reduction (Equation 1): every merge decreases the node count by 1, so a merge that eliminates only a single edge—a connected pair with no common neighbours—leaves the number of fundamental cycles unchanged. Each additional edge eliminated (i.e. each additional common neighbour) removes one further fundamental cycle, which is why structural merges are prioritised by the number of edges eliminated. The non-overlap constraint ensures message passing can propagate the effect of the changes to the graph before another application of SHAPESHIFTER. This is deliberately more aggressive than a strictly sequential greedy algorithm (merge one pair, recompute, repeat), since each

merge only shrinks the cluster count by 1—too slow for large graphs. Repeated applications still adapt to the consequences of previous merges, just at the granularity of a full application rather than after every individual merge.

3.2. How to Merge the Clusters

Accepted pairs are merged one at a time. Let C_i and C_j (with $i < j$) denote an accepted pair; note that such a pair need not be connected in the graph. Throughout this section, whenever two clusters are not connected, we take the sepset between them to be empty and its belief to be 1 so the expressions below apply unchanged. The two clusters are combined into the lower-index cluster, $C'_i = C_i \cup C_j$, with C_j and $S_{i,j}$ removed. The merged cluster's belief is initialised as

$$\Psi'_i(C'_i) = \frac{\Psi_i(C_i)\Psi_j(C_j)}{\psi_{i,j}(S_{i,j})}. \tag{3}$$

This preserves the reparameterisation of the original distribution: it can be verified directly that replacing the individual components $\Psi_i(C_i)$, $\Psi_j(C_j)$, and $\psi_{i,j}(S_{i,j})$ with $\Psi'_i(C'_i)$ leaves Equation 2 unchanged.

RIP Restoration: Let $X \in (C_i \cap C_j) \setminus S_{i,j}$ be a variable shared by the clusters but absent from their sepset prior to merging. Such an X is absent precisely because its presence would have created a cycle in X through C_i and C_j, violating the RIP; after merging, this newly created cycle must be eliminated to restore the RIP. We do this by removing X from an appropriate sepset on one side of the merge. Let C_m be connected to C_i such that $X \in S_{i,m}$ and C_m lies on the induced cycle of X. We set $S'_{i,m} = S_{i,m} \setminus \{X\}$, and update its belief as well as the cluster belief of C_m[2]. The sepset belief is updated by marginalising the merged cluster's belief,

$$\psi'_{i,m}(S'_{i,m}) = \sum_{C'_i \setminus S'_{i,m}} \Psi'_i(C'_i), \tag{4}$$

and the cluster belief is updated as

$$\Psi'_m(C_m) = \Psi_m(C_m)\,\frac{\psi'_{i,m}(S'_{i,m})}{\psi_{i,m}(S_{i,m})}. \tag{5}$$

Sepset Transfer: A cluster C_k that was connected to C_j before the merge needs to transfer its sepset $S_{j,k}$ to C_i. The sepset is updated as $S'_{i,k} = S_{i,k} \cup S_{j,k}$, and $S_{j,k}$ is removed. The beliefs of sepset $S_{i,k}$ and cluster C_k are updated[2], through

$$\psi'_{i,k}(S'_{i,k}) = \sum_{C'_i \setminus S'_{i,k}} \Psi'_i(C'_i), \tag{6}$$

and

$$\Psi'_k(C_k) = \Psi_k(C_k)\,\frac{\psi'_{i,k}(S'_{i,k})}{\psi_{i,k}(S_{i,k})\psi_{j,k}(S_{j,k})}. \tag{7}$$

Again, this preserves the reparameterisation in Equation 2.

[2] Following the belief update algorithm (Lauritzen and Spiegelhalter, 1988).

3.3. Algorithm Listings

We formalise SHAPESHIFTER in Algorithm 1, with subroutines in Algorithms 2 and 3.

Algorithm 1: SHAPESHIFTER

Input: CG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with beliefs $\mathbf{\Psi}$ and $\boldsymbol{\psi}$, parameters { `runBoth`, δ_D, δ_H, δ_N}
Output: Updated CG $\mathcal{G}$ and beliefs $\mathbf{\Psi}$ and $\boldsymbol{\psi}$
Note: a sepset between clusters that are not connected is taken to be empty, with belief 1

$\mathcal{M} \leftarrow \emptyset$ `// initialise merges`
$\mathcal{M} \leftarrow$ DISPUTINGMERGES$(\mathcal{G},\ \mathbf{\Psi},\ \boldsymbol{\psi},\ \mathcal{M},\ \delta_D,\ \delta_N)$ `// compile merges`
if `runBoth` or $\mathcal{M} = \emptyset$ **then** $\mathcal{M} \leftarrow$ STRUCTURALMERGES$(\mathcal{G},\ \mathbf{\Psi},\ \boldsymbol{\psi},\ \mathcal{M},\ \delta_H,\ \delta_N)$
for $(i, j) \in \mathcal{M}$ **do** `// perform merges`
  $C'_i \leftarrow C_i \cup C_j$
  $\Psi'_i(C'_i) \leftarrow \frac{\Psi_i(C_i)\Psi_j(C_j)}{\psi_{i,j}(S_{i,j})}$
  for $X \in [(C_i \cap C_j) \setminus S_{i,j}]$ **do** `// RIP restoration`
    Let C_m be connected to C_i such that $X \in S_{i,m}$ and C_m is on the induced cycle
    $S'_{i,m} \leftarrow S_{i,m} \setminus \{X\}$
    $\psi'_{i,m}(S'_{i,m}) \leftarrow \sum_{C'_i \setminus S'_{i,m}} \Psi'_i(C'_i)$
    $\Psi'_m(C_m) \leftarrow \Psi_m(C_m)\ \frac{\psi'_{i,m}(S'_{i,m})}{\psi_{i,m}(S_{i,m})}$
  end
  for C_k connected to C_j **do** `// sepset transfer`
    $S'_{i,k} \leftarrow S_{i,k} \cup S_{j,k}$
    $\psi'_{i,k}(S'_{i,k}) \leftarrow \sum_{C'_i \setminus S'_{i,k}} \Psi'_i(C'_i)$
    $\Psi'_k(C_k) \leftarrow \Psi_k(C_k)\ \frac{\psi'_{i,k}(S'_{i,k})}{\psi_{i,k}(S_{i,k})\psi_{j,k}(S_{j,k})}$
    Remove $S_{j,k}$ and $\psi_{j,k}(S_{j,k})$
  end
  Remove C_j, $\Psi_j(C_j)$, $S_{i,j}$, and $\psi_{i,j}(S_{i,j})$
end
return $(\mathcal{G}, \mathbf{\Psi}, \boldsymbol{\psi})$

Algorithm 2: DISPUTINGMERGES

Input: CG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with beliefs $\mathbf{\Psi}$ and $\boldsymbol{\psi}$, merges $\mathcal{M}$, parameters $\{\delta_D, \delta_N\}$
Output: Updated $\mathcal{M}$

$\mathcal{S} \leftarrow \emptyset$ `// initialise suitable merges`
for $(i, j) \in \mathcal{E}$ **do** `// compile suitable merges`
  $D \leftarrow \mathrm{KL}\Big(\sum_{C_i \setminus S_{i,j}} \Psi_i(C_i) \;||\; \sum_{C_j \setminus S_{i,j}} \Psi_j(C_j)\Big)$
  $N \leftarrow$ TABLESIZE$\Big(\Psi_i(C_i), \Psi_j(C_j)\Big)$ (Sadie et al., 2026, alg. 2)
  if $D > \delta_D$ and $N < \delta_N$ **then** insert (i, j) into $\mathcal{S}$ `// store suitable merge`
end
Sort $\mathcal{S}$ on D (in decreasing order) `// sort suitable merges`
for $(i, j) \in \mathcal{S}$ **do** `// filter suitable merges`
  if neither i nor j in any element of $\mathcal{M}$ **then** insert (i, j) into $\mathcal{M}$
end
return $\mathcal{M}$

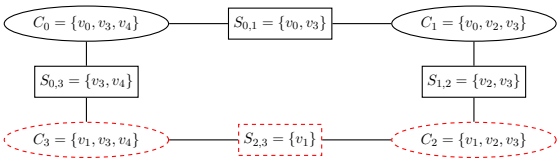
Algorithm 3: structuralMerges

Input: CG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with beliefs $\mathbf{\Psi}$ and $\boldsymbol{\psi}$, merges $\mathcal{M}$, parameters $\{\delta_H, \delta_N\}$
Output: Updated $\mathcal{M}$

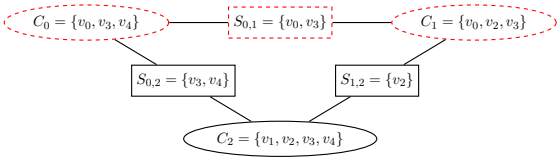
$\mathcal{S} \leftarrow \emptyset$ // initialise suitable merges
for (i, j) where $C_i \cap C_j \neq \emptyset$ **do** // compile suitable merges
  $N \leftarrow \text{tableSize}\Big(\Psi_i(C_i), \Psi_j(C_j)\Big)$ (Sadie et al., 2026, alg. 2)
  $\hat{H}(C_i \cup C_j) \leftarrow \text{estJointEntropy}\Big(\mathcal{G}, \{C_i, C_j\}\Big)$ (Sadie et al., 2026, alg. 1)
  $H \leftarrow \hat{H}(C_i \cup C_j) - \max\Big(H_{\Psi_i}(C_i), H_{\Psi_j}(C_j)\Big)$
  $n_{\text{elim}} \leftarrow \Big|\text{common neighbours of } C_i \text{ and } C_j\Big| + \Big[C_i \text{ connected to } C_j\Big]$[3]
  if $H < \delta_H$ and $N < \delta_N$ **then** insert (i, j) into $\mathcal{S}$ // store suitable merge
end
Sort $\mathcal{S}$, first on n_{elim} (in decreasing order), then on H (in increasing order) // sort suitable merges
for $(i, j) \in \mathcal{S}$ **do** // filter suitable merges
  if neither i nor j in any element of $\mathcal{M}$ **then** insert (i, j) into $\mathcal{M}$
end
return $\mathcal{M}$

3.4. Detailed Example

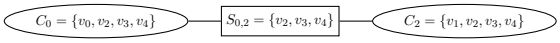
We use a small model to illustrate the mechanics of the SHAPESHIFTER algorithm. Suppose we have four factors $\phi_0(v_0, v_3, v_4)$, $\phi_1(v_0, v_2, v_3)$, $\phi_2(v_1, v_2, v_3)$, and $\phi_3(v_1, v_3, v_4)$, defining a joint probability distribution $\tilde{p}$ over the variables $\{v_0, v_1, v_2, v_3, v_4\}$. In Figure 1(*a*) we provide a CG representation of the model. Suppose our set of accepted merges is $\mathcal{M} = \{(2, 3), (0, 1)\}$.



(*a*) A simple cluster graph (CG), where, in order to adhere to the RIP for variable v_3, $S_{2,3} \neq C_2 \cap C_3$.



(*b*) The resultant CG after merging C_2 and C_3.



(*c*) The resultant CG after merging C_0 and C_1.

Figure 1: Illustration of the merge mechanics of the SHAPESHIFTER algorithm.

We start by merging C_2 and C_3, indicated by the dashed red ellipses in Figure 1(*a*). C_2 is updated as $C_2' = C_2 \cup C_3 = \{v_1, v_2, v_3, v_4\}$, and its belief is updated as per Equation 3. To restore the RIP we remove v_3 from sepset $S_{1,2}$, yielding $S_{1,2}' = S_{1,2} \setminus \{v_3\} = \{v_2\}$. We update its belief, and that of C_1, using Equations 4 and 5. Then $S_{0,3}$ is transferred to C_2' and becomes $S_{0,2}' = S_{0,3} = \{v_3, v_4\}$, with its belief updated as per Equation 6. We update

[3] where $\big[P\big] = 1$ if P is true and 0 otherwise

the belief of C_0 using Equation 7. Finally, both C_3 and $S_{2,3}$ and their corresponding beliefs are removed, and we are left with Figure 1(*b*).

We now merge C_0 and C_1, as indicated by the dashed red ellipses in Figure 1(*b*). C_0 is updated as $C_0' = C_0 \cup C_1 = \{v_0, v_2, v_3, v_4\}$, and its belief is updated as per Equation 3. There were no shared variables absent from $S_{0,1}$, so the RIP remains unaffected. We transfer $S_{1,2}$ to C_0', and since $S_{0,2}$ already exists, we merge the two to be $S_{0,2}' = S_{0,2} \cup S_{1,2} = \{v_2, v_3, v_4\}$, with the belief updated as per Equation 6 and that of C_2 as per Equation 7. Finally, both C_1 and $S_{0,1}$ and their corresponding beliefs are removed, and we are left with Figure 1(*c*).

For a demonstration of multiple applications of SHAPESHIFTER on a larger model, refer to Appendix A.

4. Experimental Setup and Results

We evaluate SHAPESHIFTER on two structurally distinct domains—error correction (channel decoding) and constraint satisfaction (Sudoku puzzles)—chosen specifically to demonstrate its role as a general-purpose inference component, rather than a domain-specific solver. As baselines, we compare against an FG and an LTRIP CG across both domains, and against a JT where tractable. Purge-and-merge is omitted, as it is limited to constraint satisfaction and does not apply to error correction. Since SHAPESHIFTER is agnostic to the initial choice, for convenience we use an LTRIP CG for its starting graph.

All methods share the same underlying inference process: message passing on a CG using the belief update rule (Lauritzen and Spiegelhalter, 1988) (Section 2), with convergence defined by both calibration and assignment agreement. For SHAPESHIFTER, this message passing is interleaved with repeated applications of the algorithm itself (Section 3); we begin with a small number of sweeps, followed by a single application of SHAPESHIFTER, and repeat until convergence. SHAPESHIFTER requires minimal problem-specific configuration: in the error-correction experiments, both disputing and structural merges are enabled (`runBoth=true`), while in the Sudoku experiments, structural merges are only performed in the absence of disputing merges (`runBoth=false`). This distinction reflects the different natures of the problems (Sudoku puzzles have a unique solution, while error correction allows for multiple possible decodings) and was guided by empirical observation during development.

All experiments were performed on an i7-13700K desktop PC with 24 threads and 64 GB DDR5 RAM.

4.1. Error Correction

4.1.1. DESCRIPTION

We make use of binary linear error-correction codes for this experiment, and provide only a brief description of how they work. Such codes, which include low-density parity-check codes, are used in modern communication systems. These systems transmit bit sequences over noisy channels, which can introduce bit errors and corrupt the received message. Each code comprises a set of even-parity constraints which translate directly to the factors with which we build the PGM. For more information, the reader is referred to Johnson (2006).

There are two sets of variables involved in decoding: the original transmitted bits, which are unknown, and the received bits, which are observed. The decoding process consists of conditioning on the received bits as evidence in the PGM, then performing inference through message passing, with the goal of recovering the original transmitted sequence. There are two outcomes: either all bit errors are corrected, or not. In the latter case, we have a frame error, where a frame denotes a single transmitted sequence. The frame error rate (FER) is the fraction of transmitted sequences that result in a frame error, and serves as our measure of inference quality. Different codes have different error-correction capabilities. The JT—which allows for exact inference—will recover the original transmitted sequence provided the number of errors in the received sequence does not exceed the code's correction capability.

The channel model determines the mechanics and probability of a bit error—we use the binary symmetric channel (MacKay, 2003, ch. 1)—and the signal-to-noise ratio (SNR) governs the reliability of the channel, with higher SNR corresponding to fewer expected bit errors.

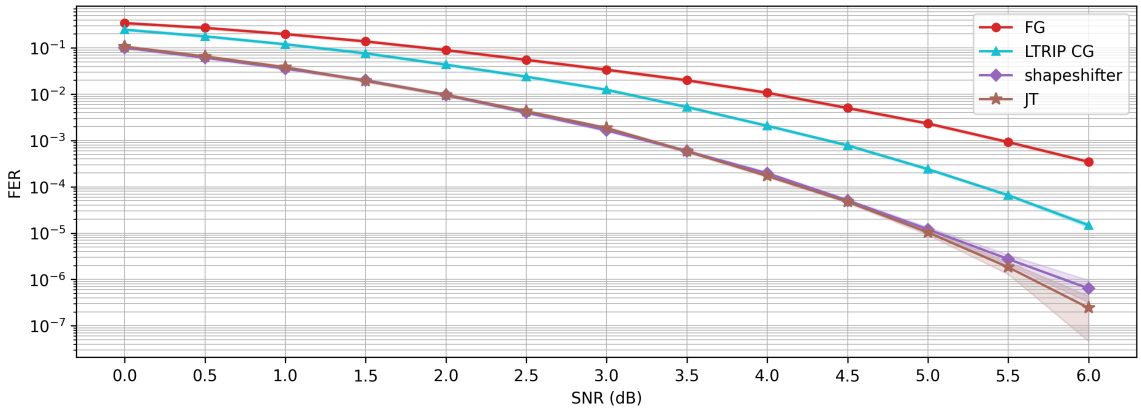
4.1.2. Experimental Setup

We evaluate on two popular codes: the Golay $(23, 12, 7)$ code and the $(120, 60, 8)$ 5G New Radio code. The Golay code defines a model over 23 unobserved and 23 observed variables across 11 factors (average 8.3 variables per factor after observation), and is sufficiently small to permit JT construction, enabling a direct comparison of SHAPESHIFTER against exact inference. The 5G NR code defines a model over 120 unobserved and 120 observed variables across 60 factors (average 6.7 variables per factor after observation). In this case, the JT is computationally infeasible; thus we use this model to test the scalability of SHAPESHIFTER.

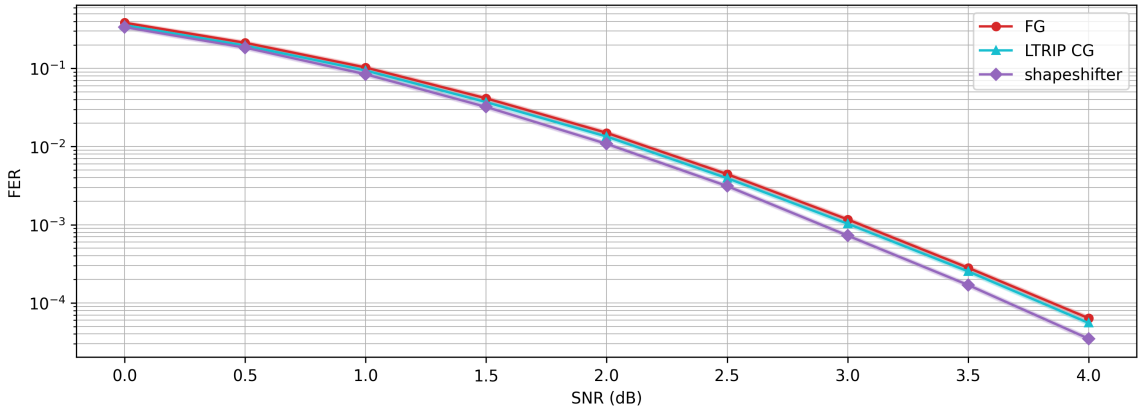
We test SNR values in $\{0, 0.5, 1, \ldots, 6\}$ dB for the Golay code and $\{0, 0.5, 1, \ldots, 4\}$ dB for the 5G NR code, and for each value transmit sequences until 1000 frame errors have occurred or 25 million frames have been transmitted (whichever comes first), to report the resulting FER. In Figure 2, we use shading to visualise two-standard-deviation confidence intervals of the FER using a Bayesian posterior, as detailed in Appendix B.

4.1.3. Results

Figure 2 shows the FER curves and confidence intervals for both codes. In Figure 2(a), there is a clear performance gap between the FG, LTRIP CG, and JT, which further confirms that inference quality is sensitive to the choice of graph structure. SHAPESHIFTER not only improves on the performance of the other loopy-structured graphs, but closely tracks the JT, indicating near-exact inference quality on the Golay code. In Figure 2(b), SHAPESHIFTER consistently achieves a lower FER value than the two static graph baselines, while the FG and LTRIP CG remain almost indistinguishable. This suggests that our dynamic restructuring can yield meaningfully improved inference, even at scales where exact inference is computationally infeasible.



(a) Frame error rates for the $(23, 12, 7)$ Golay code.



(b) Frame error rates for the $(120, 60, 8)$ 5G NR code.

Figure 2: The FERs of the FG, LTRIP CG, SHAPESHIFTER (and JT in (a)) for decoding inference on two error-correction codes. The shading denotes the two-standard-deviation confidence intervals of the FER (as per Appendix B); in (b) it is too narrow to be visible. SHAPESHIFTER demonstrates lower FER—that is, improved inference quality—when compared to the FG and LTRIP CG, and tracks the JT in (a).

4.2. Sudoku Puzzles

4.2.1. DESCRIPTION

We make use of Sudoku puzzles for this experiment. For details on these puzzles, the reader is referred to Bauke (2008). Sudoku puzzles offer a starkly different problem structure to error correction: rather than a noisy channel model, the problem is one of pure constraint satisfaction over a large discrete search space. A standard Sudoku puzzle consists of a 9×9 grid with some cells filled in (clues), where the objective is to determine the remaining values subject to the constraint that each row, column, and 3×3 sub-block contains each

of the digits 1–9 exactly once. A puzzle with exactly one solution must contain at least 17 clues (McGuire et al., 2014). The solving process consists of conditioning on the observed cells as evidence in the PGM and performing inference through message passing, with the goal of recovering a complete, constraint-satisfying assignment. There are two outcomes: either all the cells' values are uniquely determined, in which case the puzzle is solved, or they are not. Crucially, this check has no access to the puzzle's true solution: a puzzle is deemed solved only once the completed grid is directly verified against the Sudoku constraints. The fraction of puzzles solved (i.e. accuracy) serves as our measure of inference quality. Although a JT is computationally infeasible for standard 9×9 Sudoku, it would perform exact inference and therefore solve all puzzles, establishing a theoretical upper bound of 100% accuracy.

4.2.2. Experimental Setup

We evaluate on three standard Sudoku datasets: the collection of all known valid and unique 17-clue puzzles compiled by Royle (2012); the widely used top95 collection (Stertenbrink, 2005); and a collection of puzzles considered by its compiler to be among the most difficult (Champagne (pseud.), 2019), a characterisation we found to hold empirically. The Royle collection consists of 49 151 puzzles, the top95 collection of 95, and the Champagne collection of 1 000.

For each puzzle, we construct the PGM and perform inference through message passing. The inference procedure depends to a small extent on randomised initialisation, and we therefore allow every method up to three attempts before declaring a puzzle unsolved. We report the fraction of puzzles solved, as well as the execution-time quartiles (for solved puzzles) accumulated across all attempts.

4.2.3. Results

Table 1 summarises the accuracy and execution-time quartiles (for solved puzzles) across all three datasets. For the Royle collection, there is a large performance gap between methods: the LTRIP CG substantially outperforms the FG (in both accuracy and time), while SHAPESHIFTER matches the theoretical upper bound of 100% accuracy at a negligible increase in median solve time over the LTRIP CG. For the top95 collection, the performance gap becomes more striking: the LTRIP CG achieves roughly a tenfold increase in accuracy over the FG, and SHAPESHIFTER improves on the LTRIP CG by a similar margin, again matching the theoretical upper bound of 100% accuracy. This additional accuracy comes at the cost of a roughly threefold increase in median solve time—expected, given the additional computation involved in restructuring the graph—though solve times remain comfortably tractable. For the Champagne collection, almost none of these harder puzzles are solved by the FG or LTRIP CG; they each achieve 0.10% accuracy. SHAPESHIFTER achieves 100% accuracy on this dataset, at the cost of a substantially larger median solve time—which is still tractable when compared to the prohibitive cost of a JT. Across all three datasets, the upper quartile (Q3) solve times remain well within practical limits.

Table 1: Accuracy and execution-time quartiles (for solved puzzles) for Sudoku solving.

Dataset	Method	Accuracy (%)	Q1 (s)	Median (s)	Q3 (s)
Royle (49 151)	FG	45.96	0.23	0.28	0.34
	LTRIP CG	76.48	0.14	0.17	0.20
	SHAPESHIFTER	100.00	0.14	0.18	0.22
top95 (95)	FG	1.05	0.12	0.12	0.12
	LTRIP CG	10.53	0.10	0.11	0.11
	SHAPESHIFTER	100.00	0.17	0.30	0.61
Champagne (1 000)	FG	0.10	0.19	0.19	0.19
	LTRIP CG	0.10	0.08	0.08	0.08
	SHAPESHIFTER	100.00	6.69	11.97	23.28

5. Conclusion

Loopy-structured PGMs are tractable alternatives to exact inference, but are not guaranteed to converge, and when they do, may yield suboptimal results (Koller and Friedman, 2009, ch. 11). Compounding this, inference quality is sensitive to the choice of graph structure (Yedidia et al., 2005). This has motivated a range of upfront *static* design choices, including FGs and LTRIP CGs, each with some trade-off between computational cost and inference quality. At the extreme, the junction tree eliminates cycles entirely and guarantees exact inference, but is infeasible in practice due to the exponential growth in cluster sizes. The key insight in SHAPESHIFTER is that message passing on a loopy-structured graph can facilitate *dynamic* restructuring of the graph, through merging of cluster pairs.

SHAPESHIFTER identifies candidate pairs belonging to one of two categories: pairs that are in disagreement, and pairs with significant redundancy. Beyond that, merges with a greater number of edge eliminations are prioritised, thereby reducing the number of cycles in the graph. Merges are structured to preserve both the objectives of the original PGM and the current beliefs in the system, allowing message passing to continue seamlessly on the dynamically updated graph that moves progressively closer to a tree structure.

We evaluated SHAPESHIFTER on two structurally distinct domains: error correction and constraint satisfaction. The results strongly suggest that, while it is not a silver bullet, SHAPESHIFTER offers a practical middle ground—closing the performance gap relative to the junction tree by a substantial margin, without the prohibitive computational cost.

Some avenues remain for future work. The current implementation of SHAPESHIFTER is restricted to discrete random variables, but it could be extended to continuous random variables (or mixtures of discrete and continuous variables). Further, the candidate pair selection and sorting mechanisms could be optimised to prioritise the graph evolving into a tree structure, and the algorithm could be extended to merge more than two clusters at once.

Acknowledgments

The work of Katerine M. Sadie was supported by a Google DeepMind PhD scholarship.

References

Srinivas Aji and Robert McEliece. The Generalized Distributive Law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

Heiko Bauke. Passing Messages to Lonely Numbers. *Computing in Science and Engineering*, 10(2):32–40, 2008. doi: 10.1109/MCSE.2008.60.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, first edition, 2007. ISBN 0387310738.

Mark Bloemeke and Marco Valtorta. A hybrid algorithm to compute marginal and joint beliefs in bayesian networks and its complexity. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 16–23, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 155860555X.

Champagne (pseud.). Potential Hardest Sudoku Database (ph_1910). http://forum.enjoysudoku.com/the-hardest-sudokus-new-thread-t6539.html, 2019.

Brendan J. Frey, Frank R. Kschischang, Hans-Andrea Loeliger, and Niclas Wiberg. Factor Graphs and Algorithms. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 35, pages 666–680. University of Illinois, 1997.

Robert Gallager. Low-Density Parity-Check Codes. *IRE Transactions on Information Theory*, 8(1):21–28, 1962. doi: 10.1109/TIT.1962.1057683.

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Akti Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. CRC Press, Boca Raton, FL, United States of America, third edition, 2013. URL https://stat.columbia.edu/~gelman/book/.

Stuart Geman and Donald Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984. doi: 10.1109/TPAMI.1984.4767596.

J.L. Gross and J. Yellen. *Graph Theory and Its Applications*. Textbooks in Mathematics. CRC Press, Boca Raton, FL, United States of America, second edition, 2005. ISBN 9781584885054.

David Heckerman, Eric Horvitz, and Bharat Nathwani. Toward Normative Expert Systems: Part I The Pathfinder Project. *Methods of Information in Medicine*, 31:90–105, 07 1992. doi: 10.1055/s-0038-1634867.

Sarah J Johnson. Introducing Low-Density Parity-Check Codes. *University of Newcastle, Australia*, 1, 2006.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. ISBN 0262013192.

Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

Steffen L. Lauritzen and David J. Spiegelhalter. Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems. *Journal of the Royal Statistical Society. Series B, Methodological*, 50(2):157–224, 1988.

Vasilica Lepar and Prakash P. Shenoy. A Comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 328–337, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

David J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003.

Robert Mateescu, Kalev Kask, Vibhav Gogate, and Rina Dechter. Join-Graph Propagation Algorithms. *The Journal of Artificial Intelligence Research*, 37:279–328, 2010. doi: 10.1613/jair.2842.

Robert J. McEliece and Muhammed Yildirim. Belief Propagation on Partially Ordered Sets. In Joachim Rosenthal and David S. Gilliam, editors, *Mathematical Systems Theory in Biology, Communications, Computation, and Finance*, pages 275–299, New York, NY, 2003. Springer New York. ISBN 978-0-387-21696-6.

Gary McGuire, Bastian Tugemann, and Gilles Civario. There Is No 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration. *Experimental Mathematics*, 23(2):190–217, 2014. doi: 10.1080/10586458.2013.870056.

Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. doi: 10.1109/5.18626.

Gordon Royle. Minimum Sudoku. https://web.archive.org/web/20120722180233/http://mapleta.maths.uwa.edu.au/~gordon/sudokumin.php, 2012.

Katerine M. Sadie, Johan A. du Preez, and Willie Brink. Entropy Estimation in Cluster Graphs. *Algorithms*, 19(10), 2026. ISSN 1999-4893. doi: 10.3390/a19100889. URL https://www.mdpi.com/1999-4893/19/10/889.

Guenter Stertenbrink. Sudoku (top95). http://magictour.free.fr/top95, 2005.

Simon F. Streicher and Johan A. du Preez. Graph Coloring: Comparing Cluster Graphs to Factor Graphs. In *Proceedings of the ACM Multimedia 2017 Workshop on South African Academic Participation*, SAWACMMM '17, pages 35–42, New York, NY, USA, 2017. Association for Computing Machinery.

Simon F. Streicher and Johan A. du Preez. Strengthening Probabilistic Graphical Models: The Purge-and-Merge Algorithm. *IEEE Access*, 2021. doi: 10.1109/ACCESS.2021.3124760.

Robert M. Tanner. A Recursive Approach to Low Complexity Codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981. doi: 10.1109/TIT.1981.1056404.

Martin Tomlinson, Cen Jung Tjhai, Marcel A. Ambroze, Mohammed Ahmed, and Mubarak Jibril. *Error-Correction Coding and Decoding: Bounds, Codes, Decoders, Analysis and Applications*. Springer Nature, Cham, Switzerland, 1st edition, 2017. ISBN 3319511025.

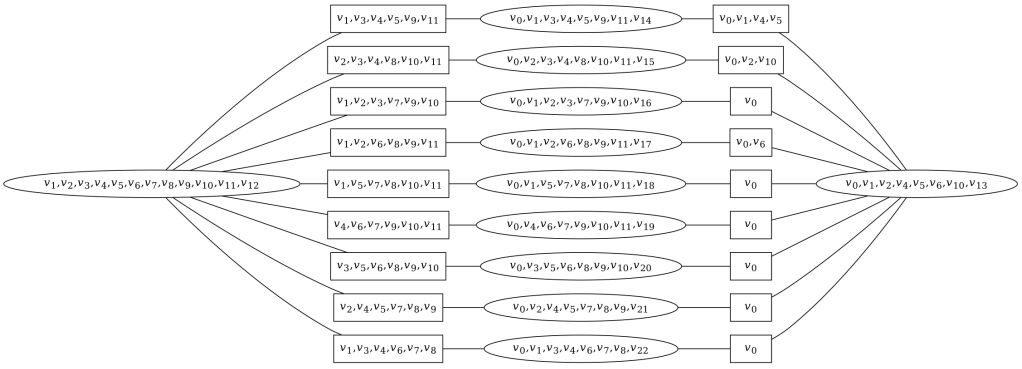
Yair Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1):1–41, 2000. ISSN 0899-7667.

Sewall Wright. Correlation and Causation. *Journal of Agricultural Research*, 20(7):557–585, 1921.

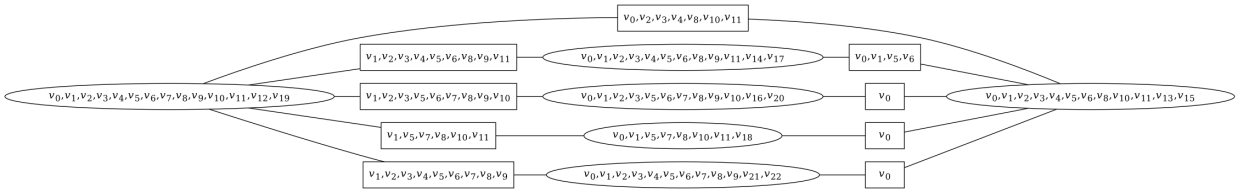
Jonothan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

Appendix A. Cluster Graph Evolution on the Golay $(23, 12, 7)$ Error-Correction Code

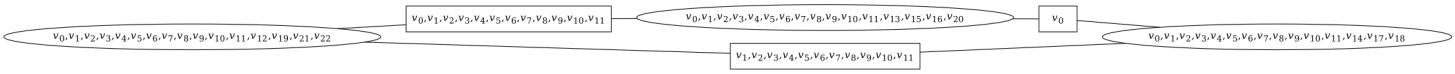
In Figure 3 we demonstrate the evolution of the SHAPESHIFTER algorithm on the Golay $(23, 12, 7)$ code. The reader can verify that both the cluster and sepset sizes grow, and the number of cycles decreases as we repeatedly apply the algorithm. This results in improved inference performance, which in the context of error correction translates to lower frame error rates (FERs).



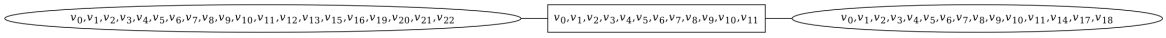
(a) The original CG, before message passing or SHAPESHIFTER.



(b) After the first application of SHAPESHIFTER.



(c) After the second application of SHAPESHIFTER.



(d) After the third application of SHAPESHIFTER, the graph has a tree structure.

Figure 3: Evolution of the Golay $(23, 12, 7)$ model's CG through 3 applications of the SHAPESHIFTER algorithm, interleaved with message passing.

Appendix B. A Bayesian Estimate for Frame Error Rates

We view the FER estimation problem through the lens of a series of Bernoulli trials, with each trial (frame transmission) having one of two outcomes: success (frame error) or failure (no frame error). We denote the unknown probability of a frame error p_{err}, and choose a conjugate Beta(α, β) prior for it.

If there are n_{err} observed frame errors after n trials, then the posterior distribution for p_{err} will be Beta$(\alpha + n_{\text{err}}, \beta + n - n_{\text{err}})$ (Gelman et al., 2013, p. 35). A Bayesian estimate of the FER is the mean of this posterior,

$$\frac{\alpha + n_{\text{err}}}{\alpha + \beta + n}. \tag{8}$$

Further, the variance is

$$\frac{(\alpha + n_{\text{err}})(\beta + n - n_{\text{err}})}{(\alpha + \beta + n)^2(\alpha + \beta + n + 1)}, \tag{9}$$

with the standard deviation quantifying the uncertainty in the estimate. For the purpose of this paper, we use $\alpha = \beta = 0.1$.