NEURAL CLUSTER TREE ELIMINATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Neural Cluster Tree Elimination (NCTE), a method for approximate probabilistic inference over discrete graphical models. The scheme approximates hard-to-compute messages over tree decompositions of the graphical model using neural networks. It extends Neural Bucket Elimination (NeuroBE)—an earlier approach restricted to bucket trees and to the bucket elimination scheme. By merging buckets into clusters, NCTE replaces a long chain of neural approximators with fewer networks, improving efficiency by requiring fewer networks to be trained and boosting accuracy by reducing error accumulation. We provide several strategies for transitioning from a bucket tree to a cluster tree to optimize the tradeoff between time and accuracy. Our empirical evaluation demonstrates the benefits and competitiveness of this approach for computing the partition function of a graphical model.

1 INTRODUCTION

Probabilistic graphical models (PGMs) compactly represent structured joint distributions over discrete variables by factorizing them into local factors. This explicit structure enables efficient probabilistic reasoning, which is central to automated decision-making and AI. However, exact inference—such as computing the partition function Z via variable elimination or Bucket Elimination (BE)—is computationally hard, scaling exponentially with the model's induced width.

Bucket elimination is a message passing algorithm that processes variables along a given ordering. For each variable, it takes the factors in its bucket and computes a message that is placed in one of the buckets downstream (Dechter, 1999a). BE is efficient when its messages, whose sizes scale exponentially with induced width, fit in memory; however, when a bucket's width is prohibitively large, an approximation scheme is needed. One popular approximation is *weighted mini-bucket elimination.* The scheme bounds the number of variables in each mini-bucket by an i-bound parameter i, so the messages it generates have scope at most $i-1$ (Dechter & Rish, 2002; Liu & Ihler, 2011).

Recent schemes like Neural Bucket Elimination (NeuroBE) (Agarwal et al., 2022) replace large exact messages with neural network (NN) approximators. While effective, NeuroBE incurs substantial overhead because it trains a separate NN for every high-width bucket in the elimination ordering. Furthermore, passing messages through a long chain of NN approximators leads to compounding approximation errors.

We introduce *Neural Cluster Tree Elimination (NCTE)*, which extends NeuroBE from bucket trees to arbitrary tree decompositions. By merging adjacent buckets into super-buckets or clusters (Dechter, 1996; Dechter & Fattah, 2001; Dechter, 2019), NCTE replaces a fine-grained chain of NN approximations with fewer NNs approximating larger clusters' messages. Merging a cluster with its neighbor can increase the cluster's size and decrease the separator size—the number of variables shared by neighboring clusters (see Figure 1).

This structural shift introduces a trade-off: (i) *reduced training time and sources of error:* Training fewer NNs directly reduces total training time and the number of error-accumulating steps across the tree decomposition. (ii) *increased sampling cost:* Generating training samples for a cluster requires summing over (eliminating) multiple variables, with time cost exponential in the number eliminated.

Contributions. The starting point of this paper is a neurosymbolic message-passing algorithm, NeuroBE, shown to be highly competitive with state-of-the art algorithms at approximating the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

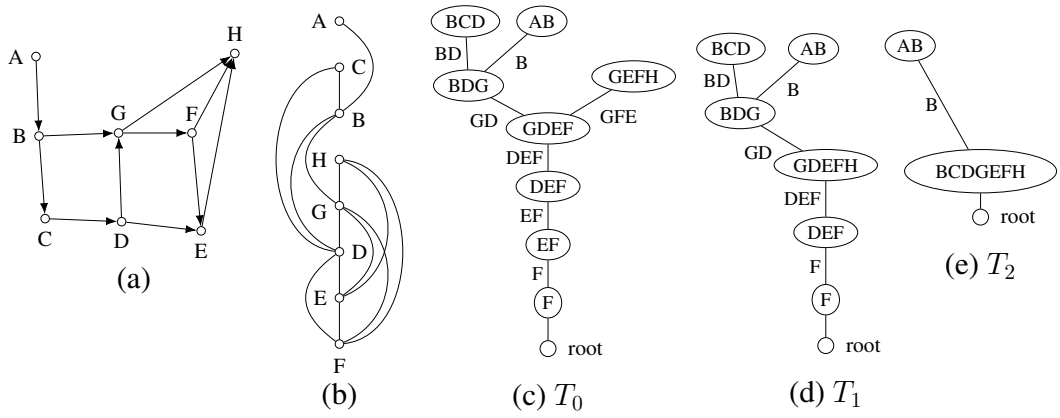

Figure 1: (a) A belief network; (b) its ordered (induced) graph, whose variables are eliminated from top to bottom (A, C, B, H, G, D, E, F), an order that induces no new edges; (c) the bucket tree T_0; (d) a cluster tree T_1; and (e) a coarser cluster tree T_2. Edges are annotated with their separators.

partition function (Agarwal et al., 2022) . We introduce the framework of *Neural Cluster-Tree-Elimination (NCTE)* which extends NeuroBE from bucket-trees to general cluster-trees. We demonstrate that NCTE is significantly more time-efficient than NeuroBE while achieving higher accuracy.

Other Related Work. **Exact and bounded elimination.** Bucket and cluster-tree elimination scale exponentially with induced width and separator size; mini-bucket and weighted mini-bucket algorithms bound message scopes via an i-bound parameter (Dechter & Rish, 2002; Liu & Ihler, 2011). **Sampling estimators.** Methods like importance sampling with search (Gogate & Dechter, 2011), dynamic importance sampling (Lou et al., 2017), annealed importance sampling (Neal, 2001), and AND/OR abstraction sampling (Kask et al., 2020) estimate partition functions directly through sampling. **Message-based approximations.** Variational bounds (Liu et al., 2015), LP relaxations (Sontag et al., 2008), and loopy belief propagation with join/region-graph extensions (Mateescu et al., 2010; Yedidia et al., 2000) reframe inference as continuous optimization.

2 Background

2.1 Graphical Models

Graphical models compactly encode structured relationships among discrete variables, representing a joint distribution as a product of local factors (Pearl, 2009; Koller & Friedman, 2009; Darwiche, 2009; Dechter, 2019); both directed Bayesian networks and undirected factor graphs fit this factor-based representation. Formally, a graphical model is a tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of discrete variables, $\mathbf{D} = \{D_i : i \in [n]\}$ specifies the finite domain for each variable X_i, and $\mathbf{F} = \{f_\alpha\}$ is a set of factors (non-negative functions) defined over subsets of variables, where α denotes the set of variable indices. Each factor $f_\alpha : D_\alpha \to \mathbb{R}_{\geq 0}$ is defined over its scope $X_\alpha \subseteq \mathbf{X}$, where D_α denotes the Cartesian product of the domains of variables in X_α. This set of factors implicitly defines a probability distribution by its product: $P(\mathbf{X}) = \frac{1}{Z} \prod_\alpha f_\alpha(X_\alpha)$ where Z, the partition function is $Z = \sum_{\mathbf{x} \in \mathbf{D}} \prod_\alpha f_\alpha(\mathbf{x}_\alpha)$

The primal graph of a graphical model associates each variable with a node. An edge connects nodes i and j if and only if some function contains both X_i and X_j in its scope. Figure 1(a) shows a Bayesian network (Pearl & Russel, 2001) with variables indexed from A to H, whose primal graph also connects the parents of each variable (e.g., B and D, the parents of G). Figure 1(b) shows its induced graph along the ordering $d = (F, E, D, G, H, B, C, A)$, listed from bottom to top, obtained by processing the nodes from last to first (top to bottom) and connecting each node's earlier neighbors (those drawn below it). The induced width of a node is its number of earlier neighbors in the induced graph, and the induced width of the ordering is the maximum induced width over all nodes. The minimum induced width over all orderings is called the **induced width** or **treewidth** of the graph. Answering a query such as the partition function $Z = \sum_{\mathbf{X}} \prod_\alpha f_\alpha(X_\alpha)$

requires summing over exponentially many joint configurations and is #P-hard (Roth, 1996), but its complexity is also bounded by the graph's induced width.

2.2 Bucket and Cluster Tree Elimination

Bucket-elimination is a variable-elimination algorithm for computing the partition function over graphical models (Dechter, 1999b). It processes each variable along an ordering $d = (X_1, \ldots, X_n)$, from last to first. For each X_i, a bucket B_i collects all unprocessed factors whose scope includes X_i but no Later variable, and marginalizes out X_i to produce a message

$$\lambda_{i\to p}(S_i) = \sum_{x_i} \prod_{f_\alpha \in B_i} f_\alpha(X_\alpha), \tag{1}$$

where $\{f_\alpha \in B_i\}$ are all factors in B_i (including messages from child buckets), the *separator* S_i is the message scope (all the variables in $B_i \setminus \{X_i\}$, and X_p is the latest variable in S_i in the ordering. The message $\lambda_{i\to p}$ is placed into bucket B_p. The buckets form a *bucket tree*, whose nodes are the buckets and an edge is directed from B_i to bucket B_p (called its parent) along the tree. In summary, BE sends messages from bucket leaves towards the bucket tree's root B_1, which then computes $Z = \sum_{x_1} \prod_{f_\alpha \in B_1} f_\alpha(X_\alpha)$. BE has time and space complexity exponential in the induced width of the elimination ordering (Dechter, 1999b).

Cluster-Tree Elimination. Cluster trees can be derived from bucket-trees by combining several consecutive buckets along the tree into larger clusters to create a coarser tree decomposition, and *Cluster Tree Elimination (CTE)* generalizes BE by sending messages between neighboring clusters along the tree. So, while BE generates messages by summing out a single variable per bucket, CTE's messages are generated by summing out several variables (all those that would be summed out by the cluster's member buckets) at once, replacing a chain of buckets that each emit a message by a single cluster emitting one message.

Example 1. *Figure 1(c) shows the bucket-tree decomposition of the graph in Figure 1(a) along the ordering of Figure 1(b), while Figure 1(d) shows a tree-decomposition that groups together buckets $GEFH$ and $GDEF$ into a single cluster $GDEFH$, and buckets DEF and EF into a single cluster DEF. Thus a single message from cluster $GDEFH$ to DEF replaces two messages in the bucket-tree, and a single message from DEF to F replaces two more.*

Formally, a tree-decomposition (Dechter & Pearl, 1989; Dechter, 2019) of $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$ is a triplet $\langle T, \chi, \psi \rangle$, where $T = (V, E)$ is a tree and $\chi : V \to 2^{\mathbf{X}}$ and $\psi : V \to 2^{\mathbf{F}}$ are labeling functions, each associating a node with its set of variables (χ) and its set of factors (ψ) subject to the standard covering and running-intersection conditions: every function is assigned to a node whose variable set contains its scope, and for each variable, the nodes whose variable set contains it induce a connected subtree (Kask et al., 2005; Gottlob et al., 2000). Cluster tree elimination sends messages over clusters along the directed rooted tree from leaves to root. Its time complexity is exponential in the maximum cluster size, while its space complexity is exponential in the maximum separator size. (Dechter & Fattah, 2001; Kask et al., 2005).

Notations The *scope* of a cluster C is its set of variables, $\chi(C)$, whose size is its *width*. The separator, denoted $S(C)$, is the intersection of a cluster C's scope and its parent cluster, $P(C)$: $S(C) = \chi(C) \cap \chi(P(C))$. The **eliminator** $E(C)$ is defined as $E(C) = \chi(C) \setminus S(C)$.

Example 2. *In Figure 1(d) we assume messages are sent along the directed tree from leaves to root. Then, the message sent from cluster C labeled $\chi(C) = \{GDEFH\}$ to its parent cluster $P(C)$ labeled $\chi(P(C)) = \{DEF\}$ is $\lambda_{C\to P(C)} = \sum_{GH} \prod_{f\in C} f$. So, the message λ's arguments are the separator variables, and it is generated by summing out its eliminator $E(C) = \{GH\}$.*

2.3 Mini-bucket and Neural Bucket elimination

When the induced-width is too high we cannot run bucket or cluster tree elimination, so we must approximate. A popular approximation is mini-bucket elimination. **Mini-bucket elimination** (MBE) partitions a high-width bucket into mini-buckets of width at most i (Dechter & Rish, 2002), whose combined messages approximate the bucket's message. Weighted mini-bucket elimination (WMBE) refines this with weighted power sums and message passing reparameterization (Liu & Ihler, 2011).

Algorithm 1 NN-TRAIN (adapted from Agarwal et al. (2022))

Require: Cluster C, separator S, eliminator E; number of layers L; constants b, η; epoch limit I
Ensure: Neural message approximation $\lambda^{\theta}_{C\to P(C)}$

1: $h \leftarrow b|S|; \quad N \leftarrow \eta(Lh)^2 \log h$
2: $\mathcal{D} \leftarrow$ GENERATE-SAMPLES(C, E, N)
3: Split $\mathcal{D}$ into training and validation sets $\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}}$
4: Initialize the neural message model $\lambda^{\theta}_{C\to P(C)}$ with L layers and hidden width h
5: Fit θ on $\mathcal{D}_{\text{tr}}$ using minibatch Adam and weighted log-space MSE (Sec. 4.1), stopping after I epochs or three consecutive epochs without improvement in the best validation loss
6: **return** $\lambda^{\theta}_{C\to P(C)}$

Neural Bucket Elimination (NeuroBE). An alternative to WMBE is learning a parameterized approximation of high-width messages using neural networks. Deep Bucket Elimination (DBE) (Razeghi et al., 2021) introduced this approach, training an NN per high-width bucket using sampled assignments and target message values. NeuroBE (Agarwal et al., 2022) refined DBE, improving accuracy and runtime. Like bucket elimination, NeuroBE processes buckets in reverse elimination order. For each bucket B_i, if its induced-width is at most $w_{\max}$, it computes the exact message $\lambda^{*}_{X_i\to P_i} = \sum_{X_i} \prod_{f_\alpha \in B_i} f_\alpha$. Otherwise, it fits an NN approximation $\lambda_{\theta_{B_i}, X_i\to P_i}$ via a procedure called NN-TRAIN and places the resulting message into the parent bucket B_{p_i}. Architecture, loss, and training details are in Agarwal et al. (2022); NeuroBE was shown to be far more *accurate* than weighted mini-bucket (wmb) even when the latter was provided with highest memory resources.

3 NEURAL CLUSTER TREE ELIMINATION

In NeuroBE (Agarwal et al., 2022), runtime is dominated by the need to train many neural networks. To reduce this training cost, our Neural Cluster Tree Elimination (NCTE) scheme merges adjacent buckets into clusters, aiming to reduce the number of messages requiring NN approximation. Though merging can reduce training time, generating each sample requires marginalizing over more variables, potentially increasing sample generation costs. NCTE must therefore balance these two costs. We first describe sample generation and present NCTE for a given cluster tree, then introduce merging strategies that balance training and sample-generation costs.

Sample generation. To fit an NN to the outgoing message $\lambda_{C\to P(C)}$, from a cluster C to its parent cluster $P(C)$, we generate training pairs $(s, \lambda^*(s))$. We draw assignments s to the separator S between C and its parent $P(C)$ uniformly at random and compute $\lambda^*(s) = \sum_{e\in D(E)} \prod_{g\in C} g(s, e)$, where E is the set of variables eliminated by C, $D(E)$ is their joint domain, and the product is over the cluster's original factors and messages received from its children. We denote this procedure by GENERATE-SAMPLES(C, E, N), where N is the number of training samples. Clearly the complexity of generating a single sample is exponential in $|E|$ since we sum over its domain.

Proposition 1 (GENERATE-SAMPLES complexity)**.** *Let r be the number of functions in cluster C, and let k bound the domain size Then Sample-generate(C, E, N) has time complexity $O\big(rN\, k^{|E|}\big)$.*

Neural Network training NN-TRAIN (Algorithm 1) extends NeuroBE's training procedure (Agarwal et al., 2022) to cluster messages by modifying sample generation. It fits an NN $\lambda^{\theta}_{C\to P(C)}$ over separator S using the generated samples $\{(s_n, \lambda^{*}_{C\to P(C)}(s_n))\}_{n=1}^{N}$.

Step 1 sets the NN's hidden width h and the number of samples accordingly, giving $N = O(|S|^2 \log |S|)$ for fixed L, b, and η. Step 2 generates the training and validation data in $O(r|S|^2 \log |S|\, k^{|E|})$ time (Proposition 1). Steps 3-5 perform the training. Let γ_{NN} denote the average fitting time per sample over the full epoch budget I, without early stopping.

Proposition 2. *The runtime of* NN-TRAIN *for a cluster with separator S and eliminator E is bounded by $O(\,(|S|^2 \cdot log|S|)[r \cdot k^{|E|} + \gamma_{\text{NN}}]\,)$.*

Algorithm 2 Neural Cluster Tree Elimination (NCTE)

Require: Graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F})$; tree decomposition $\mathcal{T} = \langle T, \chi, \psi \rangle$ of $\mathcal{M}$, cluster order $(C_1, \ldots, C_m)$ (root C_1 last), w the treewidth of $\mathcal{T}$, s the maximum separator of $\mathcal{T}$. Hyperparameters $s_{\max}$ and $w_{\max}$; number of layers L; constants b, η

Ensure: Approximate Partition function $\hat{Z}$, Cluster-tree augmented with all messages.

1: **for** $t = m$ down to 2 **do**
2: **if** $|S(C_t)| \leq s_{\max}$ **and** $w(C_t) \leq w_{\max}$ **then**
3: $\lambda_{C_t \to P(C_t)} \leftarrow \sum_{E(C_t)} \prod_{g_\alpha \in \psi(C_t)} g_\alpha$ {exact message}
4: **else**
5: $\lambda_{C_t \to P(C_t)} \leftarrow$ NN-TRAIN(C_t, S, E) {neural approximation; samples drawn from the exact message $\lambda^*_{C_t \to P(C_t)}$ }
6: place $\lambda_{C_t \to P(C_t)}$ into $P(C_t)$
7: $Z \leftarrow \prod_{g \in \psi(C_1)} g$ {the root scope is empty, so all remaining functions are scalars}
8: **return** Z and $\mathcal{T}$ augmented with generated messages.

Algorithm Neural Cluster Tree elimination (NCTE) NCTE (Algorithm 2) uses the separator bound $s_{\max}$ and cluster-width bound $w_{\max}$ to choose between tabular (exact) and NN messages. $s_{\max}$ limits table storage to $O(k^{s_{\max}})$, while $w_{\max}$ bounds exact-computation cost by $O(rk^{w_{\max}})$ per cluster. Both are needed because separator size does not bound cluster width (Dechter, 1996).

Processing clusters from leaves to root, NCTE computes and stores a tabular message when $|S(C_t)| \leq s_{\max}$ and $w(C_t) \leq w_{\max}$ (line 3). Otherwise, NN-TRAIN approximates the message using generated training samples (line 5). The resulting message is passed to the parent cluster (line 6). At the root, all functions (now just scalars) are multiplied to estimate the partition function Z (line 7). With a bucket-tree decomposition, NCTE reduces to NeuroBE.

Complexity of NCTE. With cluster-width bound $w_{\max}$, exact processing takes $O(rk^{w_{\max}})$ time per cluster, where r counts its original factors and incoming messages. We treat factor and NN evaluations as constant-time operations. By Proposition 2, an upper bound covering either branch for a cluster with separator S and eliminator E is $O\big(rk^{w_{\max}} + |S|^2 \log |S| \left[rk^{|E|} + \gamma_{\mathrm{NN}}\right]\big)$. Processing each of m clusters once, NCTE takes at most m times the largest per-cluster bound.

3.1 Selecting a good tree decomposition

To improve performance, we construct cluster trees by merging adjacent buckets, balancing fewer NN approximations against more expensive sample generation. We guide merging using the separator bound $s_{\max}$ and an eliminator budget $e_{\max}$, which limits the number of variables eliminated by a merged cluster and thus controls sample-generation cost. We consider two merging schemes. *Subsumption-only* requires the child's scope to contain the parent's scope, whereas *Reduce-NN* permits merges without this restriction. We first define the merging operation.

Merging adjacent clusters. Merging a cluster C with its parent $P(C)$ in a tree decomposition $\mathcal{T} = \langle T, \chi, \psi \rangle$ produces a cluster U with variables $\chi(U) = \chi(C) \cup \chi(P(C))$ and functions $\psi(U) = \psi(C) \cup \psi(P(C))$. The merged cluster U takes $P(C)$'s place in T and the cluster ordering, inheriting its parent and remaining children together with C's children; C is removed. Its eliminator is $E(U) = E(C) \cup E(P(C))$. The procedure MERGE$(T, C, P(C))$ is given in Appendix A.

Subsumption-only merging. Algorithm 3 copies the input tree (step 1) and makes a single leaves-to-root pass (step 2), skipping the root (steps 3–4). It calls MERGE (step 6) when the parent's variables are *subsumed* by the child's, $\chi(P(C)) \subseteq \chi(C)$, and the merged eliminator stays within $e_{\max}$ (step 5). The merged cluster replaces $P(C)$ in both T' and the processing order. The updated labeling functions are returned with T' (steps 7–8). Such merges add no variables to the child cluster, so its width stays unchanged.

Reduce-NN. Algorithm 4 permits merges without requiring subsumption. Given an initial tree decomposition and bounds $s_{\max}$ and $e_{\max}$, it copies the tree (step 1) and processes clusters leaves

Algorithm 3 SUBSUMPTION-ONLY MERGING($\langle T, \chi, \psi \rangle$, $e_{\max}$)

Require: Tree decomposition $\langle T, \chi, \psi \rangle$ (initially the bucket tree); eliminator budget $e_{\max}$
Ensure: Coarsened decomposition $\langle T', \chi', \psi' \rangle$
1: $T' \leftarrow T$
2: **for all** clusters C in T', in leaf-to-root order **do**
3: **if** C is the root **then**
4: **continue**
5: **if** $\chi(P(C)) \subseteq \chi(C)$ **and** $|E(C) \cup E(P(C))| \leq e_{\max}$ **then**
6: MERGE($T', C, P(C)$) {the merged cluster takes $P(C)$'s place and is examined when the pass reaches it}
7: Let χ' and ψ' be the updated labeling functions of T'
8: **return** $\langle T', \chi', \psi' \rangle$ with all messages

Algorithm 4 REDUCE-NN($\langle T, \chi, \psi \rangle$, $s_{\max}$, $e_{\max}$)

Require: Tree decomposition $\langle T, \chi, \psi \rangle$; separator bound $s_{\max}$; eliminator budget $e_{\max}$.
Ensure: Coarsened decomposition $\langle T', \chi', \psi' \rangle$
1: $T' \leftarrow T$
2: $Q \leftarrow (C_1, \ldots, C_t)$ in cluster order (parent before child)
3: **while** $Q \neq \emptyset$ **do**
4: $C \leftarrow \mathrm{pop}(Q)$ {remove the last cluster}
5: **while** $\big(|S(C)| > s_{\max}\big)$ {C is approximated} **and** $|E(C) \cup E(P(C))| \leq e_{\max}$ **do**
6: $C \leftarrow$ MERGE($T', C, P(C)$)
7: Let χ' and ψ' be the updated labeling functions of T'
8: **return** $\langle T', \chi', \psi' \rangle$ with all messages

to root (steps 2–4). For each cluster C, it checks whether $|S(C)| > s_{\max}$ and $|E(C) \cup E(P(C))| \leq e_{\max}$ (step 5). If both hold, it merges C with its parent and repeats the check with the merged cluster (step 6). The eliminator budget also bounds the resulting cluster width, as shown next.

Properties of Reduce-NN. Assume we apply Reduce-NN to a bucket tree T having width w with the hyper parameter $e_{\max}$, then the width w' of the output tree-decomposition T' satisfies $w' \leq e_{\max} + w - 1$. This is because whenever we merge some cluster C with its parent $P(C)$ the resulting new cluster C' can be seen to obey that $w(C') = E(C) + E(P(C)) + S(P(C))$. So, since merges happen only if $E(C) + E(P(C)) \leq e_{\max}$ and since $S(P(C)) \leq w(P(C)) - 1 \leq w - 1$ we get the result. Note also that $e(C') \leq e_{\max}$ for any $C' \in T'$.

Selecting a good $e_{\max}$. Let a cluster C eliminate $e = |E(C)|$ variables of domain size k and train its network on N samples of the separator set $S(C)$. Generating one sample enumerates the k^e assignments to eliminated variables $E(C)$ for a sampled configuration of $S(C)$. This summation requires $\gamma_{\text{gen}}\, k^e$ time, where γ_{gen} is the time to evaluate a joint configuration of C. Let γ_{NN} denote the training time per sample over the full epoch budget (without early stopping). The time to compute C's message is thus $T(C) \approx N\,(\gamma_{\text{gen}}\, k^e + \gamma_{\text{NN}})$. Merging an NN-approximated cluster C_1 into its NN-approximated parent C_2 saves one network but eliminates $e_1 + e_2$ variables at once. Under this model, assuming equal sample counts, a sufficient condition for merging to save time is $\gamma_{\text{gen}} k^{e_1+e_2} \leq \gamma_{\text{NN}}$. Solving for $e_1 + e_2$ gives $e_1 + e_2 \leq e^*_{\max} = \log_k(\gamma_{\text{NN}}/\gamma_{\text{gen}})$.

We estimate these constants empirically on our hardware. Training a 2-layer NN with batch size 256 for 500 epochs gives $\gamma_{\text{NN}} \approx 6.25 \times 10^{-3}$ seconds per sample. The constant γ_{gen} may depend on the number of factors in the cluster, and in particular the number that are themselves neural networks (which are slower to evaluate); the highest cost we encountered in our experiments was a cluster with 21 NNs, for which $\gamma_{\text{gen}} \approx 4.0 \cdot 10^{-7}$ seconds per evaluation. Using these values with binary domains ($k = 2$) suggests time benefits to our method whenever $e_1 + e_2 \leq e^*_{\max} \approx 13.9$.

4 EXPERIMENTS

4.1 METHODOLOGY

Benchmarks. We evaluate on 25 standard instances from the UAI Inference Competition repository (UAI Inference Competition, 2018), spanning three families: (1) **Grid networks**: Pairwise binary Markov networks on $N \times N$ lattices. We include 100–400 variable `grid-small` instances ($w^* = 13$–27, exact Z known) and `grid-hard` models with periodic boundary edges (`wrap`)

that double the induced width. The suffix `fk` indicates potential strength. (2) **Pedigree networks**: Bayesian networks from genetic linkage analysis (Fishelson & Geiger, 2002) with 693–922 variables, domain sizes up to 5, high zero-probability ratios, and $w^* = 28$–35 (exact Z known). (3) **Restricted Boltzmann Machines (RBMs)**: Complete bipartite networks with $2n$ binary variables ($w^* = n$) and ferromagnetic couplings (`ferro`). We evaluate $n \in \{20, 21, 22\}$ (exact Z known).

Algorithms. Each evaluated algorithm first applies a clustering strategy and then runs NCTE on the resulting cluster tree. We compare three strategies: no merging (the NeuroBE-style baseline), subsumption-only merging, and Reduce-NN, naming each algorithm by its clustering strategy. We evaluate the merging algorithms over eliminator budgets $e_{\max} \in \{2, 4, \ldots, 18\}$, with additional measurements at higher budgets up to 24 for some instances. We set $w_{\max} = \infty$.

Neural network design and training. Each NN is fully connected, with ReLU activations and two hidden layers of width $h = 3|S|$, where $|S|$ is the separator size. Training minimizes the weighted mean-squared error $\mathcal{L}(\theta) = \frac{1}{N_{\text{tr}}} \sum_{i=1}^{N_{\text{tr}}} w_i(\hat{y}_i - y_i)^2$, with $w_i = y_i/A$, where N_{tr} is the training-set size, y_i is the log-message target shifted and scaled using extrema estimated from a separate initialization sample set $\mathcal{I}$, and $\hat{y}_i$ is its NN prediction. The constant $A = \sum_{j \in \mathcal{I}} y_j$ is computed once before training. We use Adam with learning rate 10^{-3}, batch size 256, no weight decay or learning-rate scheduling, and a maximum of 500 epochs. Training stops after three consecutive epochs without improvement in the best validation loss; the median duration is 105 epochs.

Measures and reference values. We evaluate partition function estimates $\hat{Z}$ using the signed log-error $\Delta = \log_{10} \hat{Z} - \log_{10} Z_{\text{ref}}$ and the absolute log-error $|\Delta|$. We use the exact partition function Z^* as the reference when available; for `grid-hard`, we use the estimate $\tilde{Z}$ obtained from a 10-hour run of abstraction sampling (Pezeshki et al., 2024). Reported runtimes measure the total wall-clock time per instance, including sample generation, NN training, and exact message computation.

4.2 Results

We evaluate 27 configurations spanning 25 problem instances, where each configuration pairs an instance with a separator bound $s_{\max}$. Each configuration is run with three independent random seeds, varying sample generation and NN initialization; reported values are arithmetic means over these runs. All experiments use an NVIDIA TITAN RTX GPU (24 GB VRAM) and an AMD EPYC 7302 CPU. To control for implementation differences, we reimplemented NeuroBE and reran the no-merge baseline using the same codebase.

Figures 2 and 3 compare algorithm performance across elimination budgets. Figure 2 details error (top panel) and runtime in minutes (bottom panel) for two instances per benchmark family (more in Appendix B). Blue curves, red curves, and green markers denote Reduce-NN, subsumption-only merging, and the no-merge baseline, respectively. Upper-panel labels indicate neural-approximation counts, and red crosses mark subsumption saturation, beyond which performance remains constant. Reference values correspond to exact partition functions Z^*, except for `grid-hard`, which uses an abstraction-sampling estimate and caps training sets at 100k samples per network. Figure 3 aggregates results across the 19 configurations with exact references, displaying geometric-mean absolute log-error (left panel) and geometric-mean runtime (right panel). As shown in Figure 2 (and detailed further in the appendix for individual instances), both merging schemes significantly reduce runtime compared with the baseline up to a certain point of e_{max}. Specifically, Reduce-NN's runtime is U-shaped, initially decreasing as $e_{\max}$ increases before rising again, with a minimum around $e_{\max} = 12$ or 14. This pattern illustrates the fundamental tradeoff between reducing the number of neural network approximations and increasing sample-generation time. Overall, both merging strategies yield substantial runtime improvements compared to the baseline.

On instances with exact references, both merging strategies generally improve accuracy relative to the no-merge baseline (Figure 2), although improvements are not always monotonic in $e_{\max}$. Figure 3 shows that both strategies improve aggregate accuracy and runtime throughout the displayed range. Subsumption-only error levels off once its subsumption merges are exhausted. At larger budgets, Reduce-NN achieves lower aggregate error and runtime than subsumption-only merging, with its lowest plotted aggregate error at $e_{\max} = 16$.

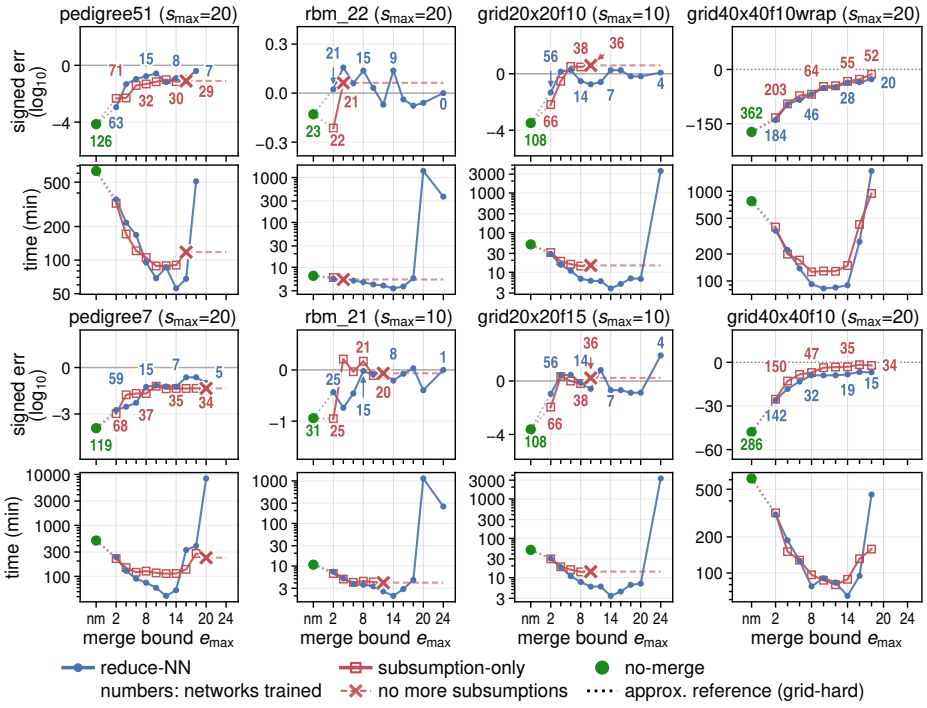

Figure 2: Signed log-error (top) and runtime in minutes (bottom) versus $e_{\max}$ for two instances from each benchmark group: pedigree, RBM, `grid-small`, and `grid-hard` (more in Appendix B). The `grid-hard` runs cap each network's training set at 100,000 samples, unlike Table 1.

Table 1 compares the three strategies at $e_{\max} = 14$, which gives the lowest aggregate runtime for Reduce-NN among the budgets plotted in Figure 3. Each main row specifies an instance and separator bound $s_{\max}$. The final two rows summarize absolute log-error and runtime on configurations with exact references using arithmetic and geometric means.

Both merging strategies reduce runtime and network count relative to no merging in all 27 configurations. Reduce-NN is fastest in 25 and uses fewer networks than subsumption-only merging in all 27. Among the 19 configurations with exact references, subsumption-only merging reduces error relative to no merging in all 19, and Reduce-NN in 18. Reduce-NN achieves the lowest error on every `grid-small` and pedigree instance, whereas subsumption-only merging does so on every RBM configuration. Thus, fewer approximations do not always yield greater accuracy. For `grid-hard`, deviations from the approximate references are mixed despite consistent runtime reductions. The arithmetic and geometric mean rows show that, on average, Reduce-NN achieves moderately lower absolute log-errors than subsumption-only merging (7% and 28% lower, respectively) while reducing time by 53% and 59%, respectively.

5 Summary and Conclusion

The starting point of this paper is algorithm NeuroBE, a neuro symbolic scheme approximating Bucket elimination, which was shown to be highly competitive against state-of-the art message-passing schemes such as weighted mini-bucket with its highest memory budget Agarwal et al. (2022). In this work we extended NeuroBE into a class of neurosymbolic algorithms (NCTE) operating over tree-decompositions. Remarkably, our new framework facilitates better performance in both time and accuracy in most of the problem instances tested. Our preliminary analysis also shows that selecting a good tree-decomposition can be obtained by analyzing the problem structure combined with empirical estimation of some performance parameters from data. Our immediate future work will aim to provide a more systematic scheme for identifying the tree-decomposition that can fit best a given problem class. More broadly NCTE messages are complementary to sampling-based estimators, so hybrid schemes that combine the two are a natural next step.

Table 1: Results at merge bound $e_{\max} = 14$: $s_{\max}$ is the separator bound, n the number of variables, w the induced width, #NN the number of networks trained, t the wall time in minutes, and $\text{err} = |\log_{10} \hat{Z} - \log_{10} Z_{\text{ref}}|$; t and err are averaged over three runs. Reference values in *italics* (grid-hard) are abstraction-sampling estimates; all others are exact. **Bold**: best t and err in each row. Table 2 (appendix) gives domain sizes and per-network sample and epoch counts. The last two rows summarize runtime and absolute log-error on configurations with exact references using arithmetic and geometric means. Cell color: t and err relative to the row's best, log scale: best $3\times$ $\geq 10\times$.

		reference			no-merge			reduce-NN			subsumption-only		
problem	$s_{\max}$	$\log_{10} Z$	n	w	#NN	t	err	#NN	t	err	#NN	t	err
grid10x10f10	10	303.09	100	13	6	2.2	.270	0	**.10**	**.0042**	1	.40	.0766
grid10x10f10wr	10	333.32	100	21	26	12.2	1.44	1	**.70**	**.436**	10	4.2	.554
grid20x20f10	10	1311.98	400	27	108	50.5	3.49	7	**3.9**	**.255**	36	14.0	.881
grid20x20f15	10	1962.98	400	27	108	50.8	3.62	7	**3.5**	**.687**	36	13.6	1.20
grid20x20f2	10	291.73	400	27	108	51.5	.416	7	**3.6**	**.101**	36	13.3	.108
grid20x20f5	10	665.12	400	27	108	51.8	2.22	7	**3.5**	**.131**	36	13.3	.533
grid40x40f10	20	*5504.45*	1600	55	286	668	40.3	19	95.8	3.54	35	**94.0**	**2.52**
grid40x40f10wr	20	*5633.53*	1600	114	362	6775	**25.9**	28	1019	39.3	55	**888**	27.4
grid40x40f15	20	*8226.87*	1600	55	286	698	48.4	19	**94.6**	**2.49**	35	99.0	4.77
grid40x40f15wr	20	*8445.74*	1600	114	362	6565	52.1	28	**996**	31.6	55	1029	**28.9**
grid40x40f2	20	*1220.27*	1600	55	286	795	7.43	19	**96.1**	**.680**	35	119	1.36
grid40x40f2wr	20	*1231.43*	1600	114	362	7223	5.74	28	**982**	1.71	55	1018	**1.66**
grid40x40f5	20	*2795.44*	1600	55	286	742	18.3	19	**97.2**	**.936**	35	100	1.65
grid40x40f5wr	20	*2831.57*	1600	114	362	7043	22.3	28	**972**	**1.52**	55	1008	2.25
pedigree13	20	-31.18	888	33	126	172	.874	7	**23.8**	**.133**	26	24.8	.245
pedigree19	20	-59.02	693	28	82	265	1.39	3	**11.7**	**.0200**	21	58.5	.436
pedigree34	20	-64.23	922	33	114	403	2.48	7	**31.2**	**.0978**	31	61.3	.717
pedigree41	20	-76.04	885	32	95	168	3.04	4	**9.9**	**.0958**	19	30.8	.484
pedigree51	20	-73.87	871	35	126	630	4.14	8	**55.7**	**.901**	30	90.2	1.15
pedigree7	20	-64.82	867	34	119	504	3.92	7	**53.6**	**1.24**	35	113	1.36
rbm_20	10	58.53	40	20	29	8.5	1.52	7	**2.1**	.342	19	3.5	**.190**
rbm_21	10	63.15	42	21	31	10.8	.945	8	**2.0**	.452	20	3.8	**.0690**
rbm_21	20	63.15	42	21	21	4.9	.446	8	**2.0**	.452	20	3.8	**.0690**
rbm_22	10	66.55	44	22	33	11.8	.414	9	**3.4**	.142	21	5.1	**.0837**
rbm_22	20	66.55	44	22	23	6.5	.198	9	**3.4**	.142	21	5.1	**.0837**
rbm_ferro_21	20	152.62	42	21	21	7.1	1.47	8	**3.2**	.945	20	6.2	**.152**
rbm_ferro_22	20	166.11	44	22	23	7.8	3.81	9	**3.5**	2.73	21	6.5	**1.66**
mean (exact)					–	127	1.90	–	**11.6**	**.490**	–	24.8	.529
geomean (exact)					–	35.9	1.32	–	**4.6**	**.223**	–	11.2	.312

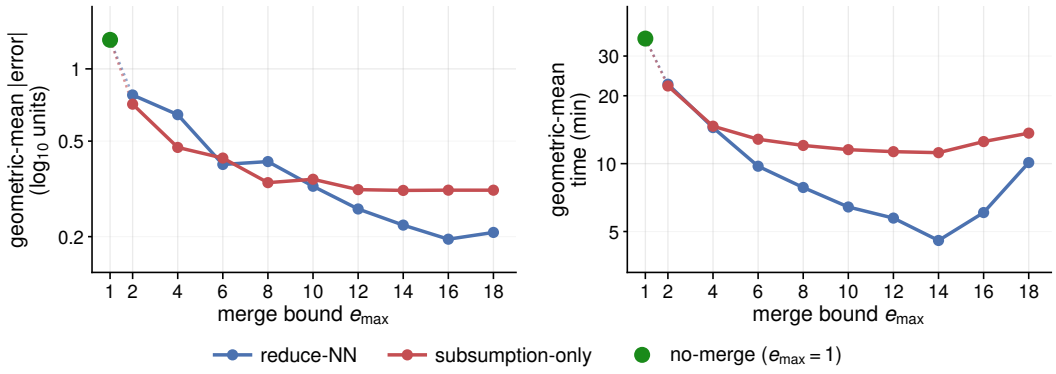

Figure 3: Geometric-mean absolute log-error (left) and runtime in minutes (right) for configurations with exact references, over $e_{\max} \in \{2, 4, \ldots, 18\}$. Each configuration's values are first averaged over three seeds. Blue and red curves denote Reduce-NN and subsumption-only merging; the green dot at $e_{\max} = 1$ denotes the no-merge baseline.

AI USE STATEMENT

We used generative AI tools to assist with developing and debugging our code, preparing and formatting data, scripting experiments, generating figures, and editing the manuscript for clarity and concision. The authors take full responsibility for the research methodology, experimental results, and final content of the paper.

REFERENCES

Sakshi Agarwal, Kalev Kask, Alexander Ihler, and Rina Dechter. Neurobe: Escalating neural network approximations of bucket elimination. In James Cussens and Kun Zhang (eds.), *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands*, volume 180 of *Proceedings of Machine Learning Research*, pp. 11–21. PMLR, 2022.

Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009. ISBN 978-0-521-88438-9. URL http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521884389.

R. Dechter. Topological parameters for time-space tradeoffs. In *Uncertainty in Artificial Intelligence (UAI'96)*, pp. 220–227, 1996.

R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113: 41–85, 1999a.

R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pp. 353–366, 1989.

R. Dechter and I Rish. Mini-buckets: A general scheme for approximating inference. *Journal of the ACM*, pp. 107–153, 2002.

Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2): 41–85, 1999b.

Rina Dechter. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms, Second Edition*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019. doi: 10.2200/S00893ED2V01Y201901AIM041. URL https://doi.org/10.2200/S00893ED2V01Y201901AIM041.

Rina Dechter and Yousri El Fattah. Topological parameters for time-space tradeoff. *Artif. Intell.*, 125(1-2):93–118, 2001. doi: 10.1016/S0004-3702(00)00050-3. URL https://doi.org/10.1016/S0004-3702(00)00050-3.

Maayan Fishelson and Dan Geiger. Exact genetic linkage computations for general pedigrees. *Bioinformatics*, 18(suppl_1):S189–S198, 2002. doi: 10.1093/bioinformatics/18.suppl_1.s189.

Vibhav Gogate and Rina Dechter. Samplesearch: Importance sampling in presence of determinism. *Artif. Intell.*, 175(2):694–729, 2011.

Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural csp decomposition methods. *Artificial Intelligence*, pp. 243–282, 2000.

K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree-decompositions for reasoning in graphical models. *Artificial Intelligence*, 166(1-2):165–193, 2005.

Kalev Kask, Bobak Pezeshki, Filjor Broka, Alexander Ihler, and Rina Dechter. Scaling up AND/OR abstraction sampling. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2020.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009. ISBN 0262013193.

Qiang Liu and Alexander T. Ihler. Bounding the partition function using holder's inequality. In *ICML*, pp. 849–856, 2011.

Qiang Liu, John W Fisher III, and Alexander Ihler. Probabilistic variational bounds for graphical models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 1432–1440. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5695-probabilistic-variational-bounds-for-graphical-models.pdf.

Qi Lou, Rina Dechter, and Alexander Ihler. Dynamic importance sampling for anytime bounds of the partition function. In *Neural Information Processing Systems (NIPS)*, 2017.

Robert Mateescu, Kalev Kask, Vibhav Gogate, and Rina Dechter. Join-graph propagation algorithms. *J. Artif. Intell. Res.*, 37:279–328, 2010.

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

J. Pearl and S. Russel. Bayesian networks. MIT Press, 2001.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009. ISBN 0-521-77362-8.

Bobak Pezeshki, Kalev Kask, Alexander Ihler, and Rina Dechter. Value-based abstraction functions for abstraction sampling. In *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence*, volume 244 of *Proceedings of Machine Learning Research*, pp. 2861–2901. PMLR, 2024.

Yasaman Razeghi, Kalev Kask, Yadong Lu, Pierre Baldi, Sakshi Agarwal, and Rina Dechter. Deep bucket elimination. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pp. 4235–4242. ijcai.org, 2021.

D. Roth. On the hardness of approximate reasoning. 82(1-2):273–302, April 1996.

David Sontag, Talya Meltzer, Amir Globerson, Tommi Jaakkola, and Yair Weiss. Tightening lp relaxations for map using message passing. In *UAI*, pp. 503–510, 2008.

UAI Inference Competition. UAI problem instances. https://www.ics.uci.edu/~ihler/uai-data/, 2018.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Generalized belief propagation. In *NIPS*, pp. 689–695, 2000.

A The Merge Operation

Algorithm 5 MERGE(T, C, P)

Require: Rooted cluster tree T with labels χ, ψ; a child cluster C and its parent P.
Ensure: Merged cluster C'; T and its labels are updated in place.

1: Create a new cluster C'.
2: $\chi(C') \leftarrow \chi(C) \cup \chi(P)$
3: $\psi(C') \leftarrow \psi(C) \cup \psi(P)$
4: $\mathrm{parent}(C') \leftarrow \mathrm{parent}(P)$
5: $\mathrm{children}(C') \leftarrow \big(\mathrm{children}(C) \cup \mathrm{children}(P)\big) \setminus \{C\}$
6: Link C' to its parent and children in T, updating their links to point to C'. If P is the root, make C' the root.
7: Remove C and P from T.
8: **return** C'

B Per-Instance Error and Runtime Curves

This appendix gives, for every benchmark instance not shown in Figure 2 and for all eight grid-hard instances, the signed error (top) and wall-clock time (bottom) as functions of the eliminator budget $e_{\max}$. Throughout, Reduce-NN is blue, subsumption-only is red, and the green marker is the no-merge baseline from which both curves start; point labels give the neural-approximation count and $\times$ marks the largest feasible bound, beyond which results are pooled. The RBM, pedigree and grid-small families are measured against an exact partition function; the grid-hard family is measured against a long-run abstraction-sampling estimate, and, unlike Table 1, its curves come from runs that cap each network's training set at 100k samples. Table 2 lists, for every run in Table 1, the domain sizes of the instance and the number of networks, mean training-sample count and mean epoch count of each merging strategy.

Table 2: Domain sizes and per-network training statistics for the runs of Table 1 (merge bound $e_{\max} = 14$): the mean and maximum variable-domain size ($\bar{k}$, $k_{\max}$) and, for each strategy, the number of trained networks (#NN, as in Table 1), the mean number of training samples per network ($\bar{m}$) and the mean number of training epochs per network before early stopping ($\bar{\epsilon}$). A dash marks a strategy that trained no network.

				no-merge			reduce-NN			subsumption-only		
problem	$s_{\max}$	$\bar{k}$	$k_{\max}$	#NN	$\bar{m}$	$\bar{\epsilon}$	#NN	$\bar{m}$	$\bar{\epsilon}$	#NN	$\bar{m}$	$\bar{\epsilon}$
grid10x10f10	10	2.0	2	6	7.3k	338	0	–	–	1	6.4k	343
grid10x10f10wr	10	2.0	2	26	15k	238	1	27k	147	10	13k	256
grid20x20f10	10	2.0	2	108	17k	226	7	21k	178	36	11k	285
grid20x20f15	10	2.0	2	108	17k	226	7	21k	162	36	11k	276
grid20x20f2	10	2.0	2	108	17k	227	7	21k	166	36	11k	281
grid20x20f5	10	2.0	2	108	17k	231	7	21k	155	36	11k	279
grid40x40f10	20	2.0	2	286	77k	169	19	99k	290	35	56k	196
grid40x40f10wr	20	2.0	2	362	262k	264	28	313k	390	55	163k	264
grid40x40f15	20	2.0	2	286	77k	167	19	99k	276	35	56k	163
grid40x40f15wr	20	2.0	2	362	262k	265	28	313k	374	55	163k	258
grid40x40f2	20	2.0	2	286	77k	191	19	99k	280	35	56k	174
grid40x40f2wr	20	2.0	2	362	262k	298	28	313k	393	55	163k	276
grid40x40f5	20	2.0	2	286	77k	172	19	99k	296	35	56k	195
grid40x40f5wr	20	2.0	2	362	262k	278	28	313k	378	55	163k	283
pedigree13	20	1.89	3	126	57k	159	7	67k	260	26	47k	155
pedigree19	20	2.23	5	82	76k	287	3	61k	397	21	65k	286
pedigree34	20	1.9	5	114	75k	212	7	79k	267	31	62k	203
pedigree41	20	2.03	5	95	58k	162	4	64k	194	19	53k	184
pedigree51	20	1.87	5	126	113k	227	8	122k	318	30	78k	240
pedigree7	20	1.9	4	119	98k	224	7	118k	254	35	79k	259
rbm_20	10	2.0	2	29	21k	123	7	24k	80	19	24k	49
rbm_21	10	2.0	2	31	23k	127	8	27k	53	20	27k	46
rbm_21	20	2.0	2	21	27k	56	8	27k	53	20	27k	46
rbm_22	10	2.0	2	33	25k	129	9	30k	80	21	30k	53
rbm_22	20	2.0	2	23	30k	63	9	30k	80	21	30k	53
rbm_ferro_21	20	2.0	2	21	27k	81	8	27k	89	20	27k	74
rbm_ferro_22	20	2.0	2	23	30k	77	9	30k	82	21	30k	69

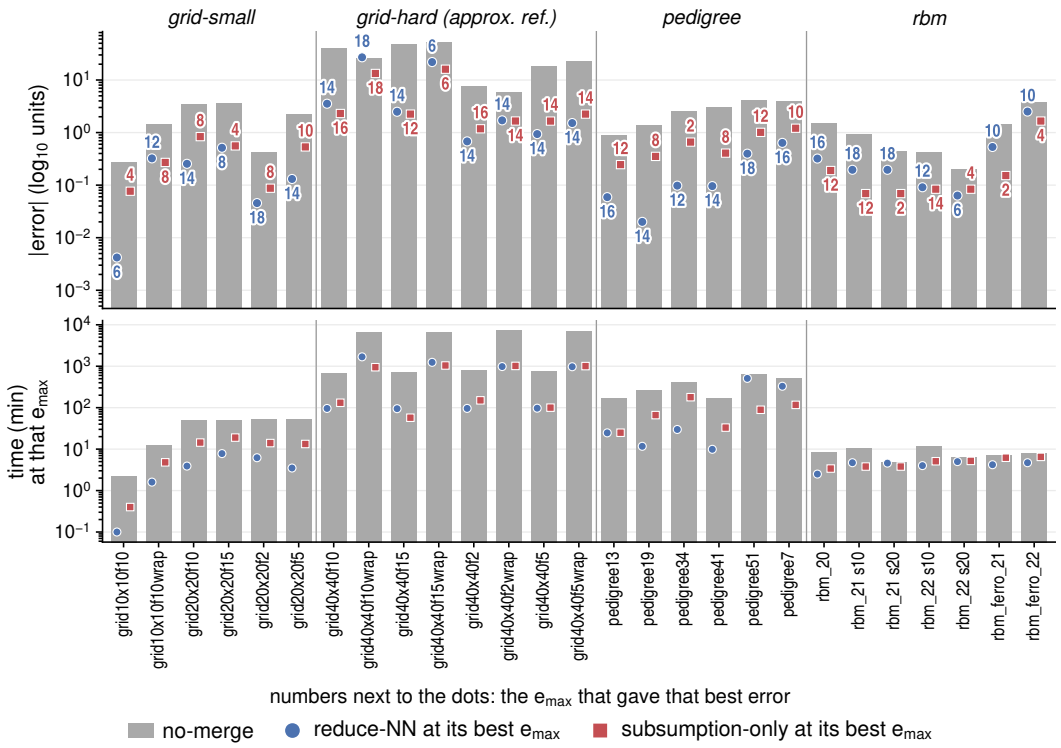

Figure 4: Best error per problem. Top: |error| in $\log_{10}$ units (log scale) for no-merge (gray bars) and for Reduce-NN (blue circles) and subsumption-only (red squares), each at the $e_{\max}$ that gave its lowest error; the number next to each marker is that $e_{\max}$. Bottom: run time in minutes for the same runs. "s10"/"s20" mark the two RBM instances run at both separator bounds $s_{\max}$. Grid-hard errors are relative to abstraction-sampling estimates (Pezeshki et al., 2024), not exact values.

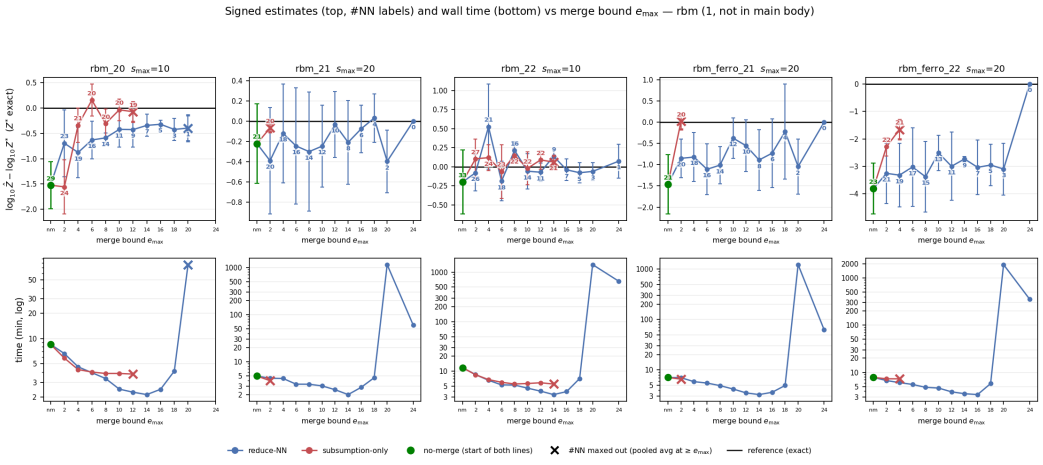

Figure 5: RBM instances: rbm_20, rbm_21 at $s_{\max} = 20$, rbm_22 at $s_{\max} = 10$, rbm_ferro_21 and rbm_ferro_22: signed error (top) and wall-clock time (bottom) versus $e_{\max}$.

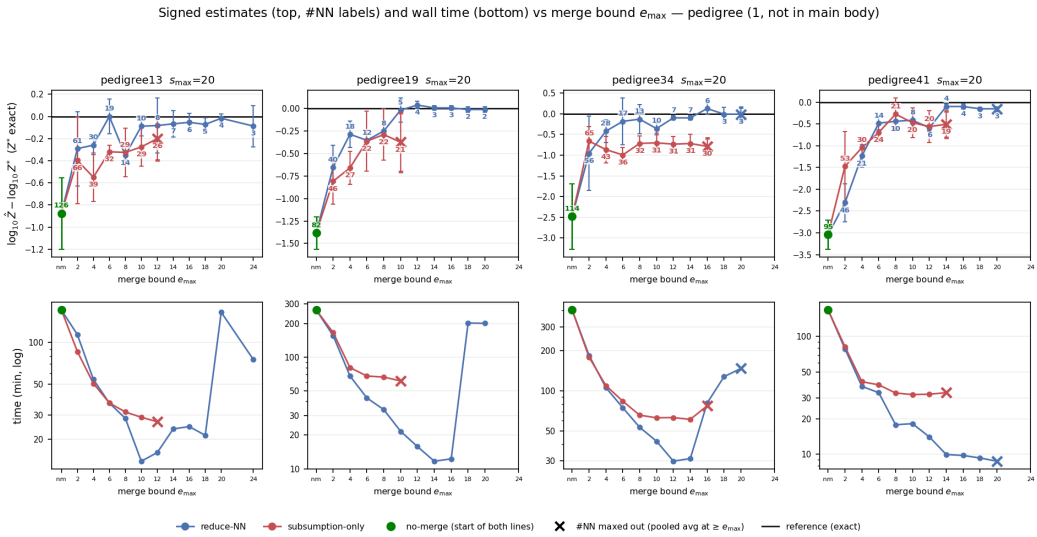

Figure 6: pedigree instances: pedigree13, pedigree19, pedigree34 and pedigree41: signed error (top) and wall-clock time (bottom) versus $e_{\max}$.

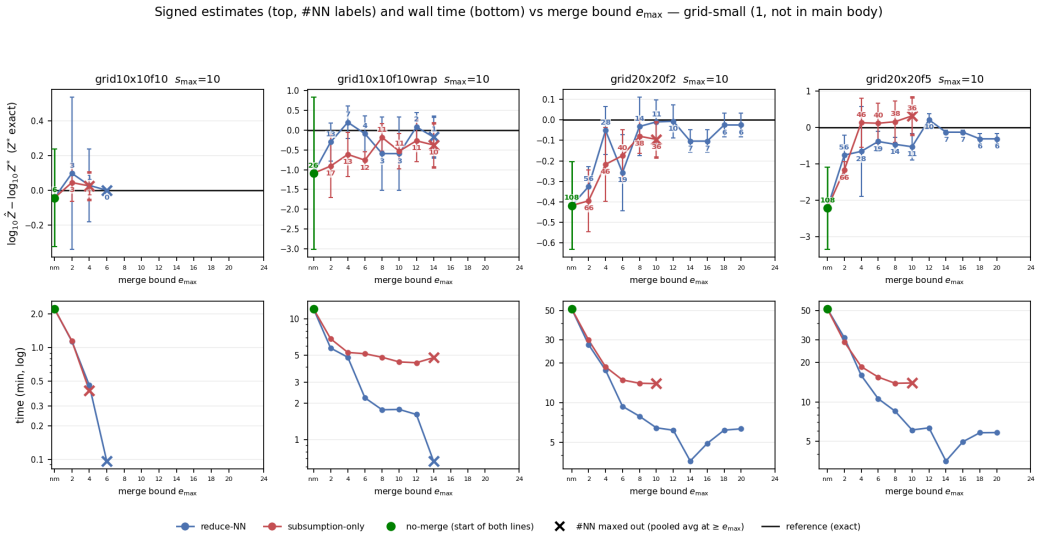

Figure 7: grid-small instances: grid10x10f10, grid10x10f10wrap, grid20x20f2 and grid20x20f5: signed error (top) and wall-clock time (bottom) versus $e_{\max}$.

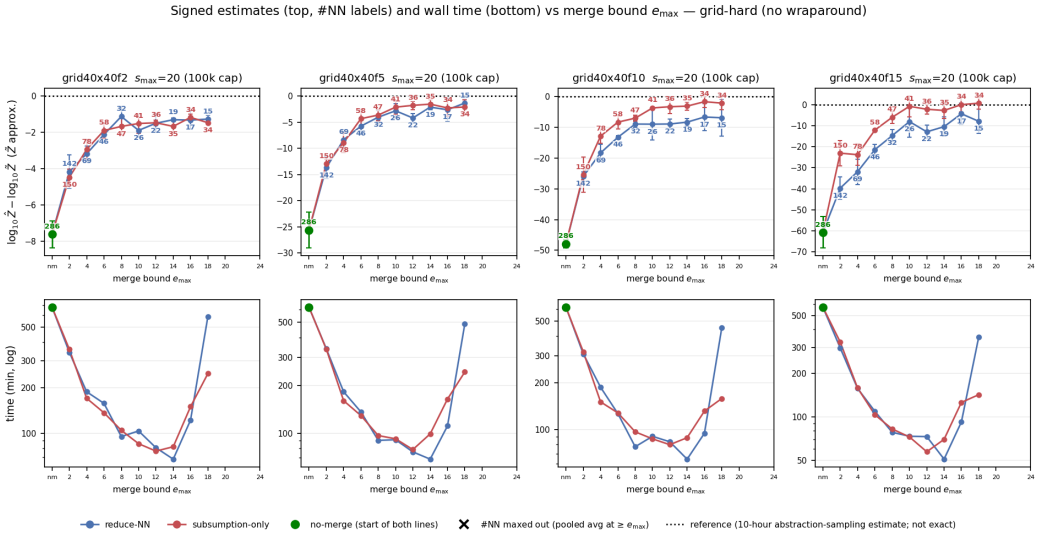

Figure 8: grid-hard instances without wraparound (1 of 2): grid40x40f2, f5, f10 and f15 (100k training-sample cap): signed error (top) and wall-clock time (bottom) versus $e_{\max}$.

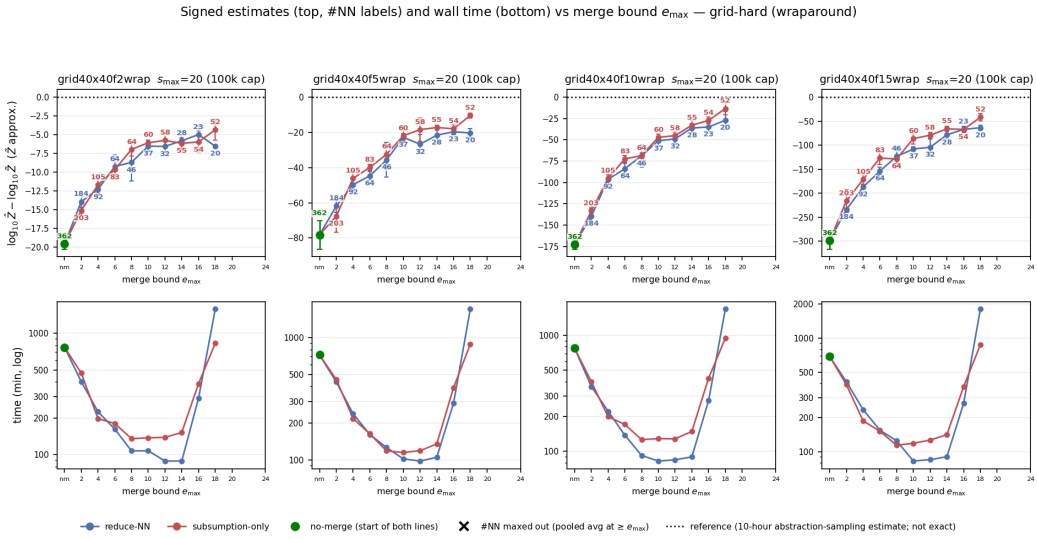

Figure 9: grid-hard instances with wraparound (2 of 2): grid40x40f2wrap, f5wrap, f10wrap and f15wrap (100k training-sample cap): signed error (top) and wall-clock time (bottom) versus $e_{\max}$.