A Appendix

A.1 Evaluation Metrics

We use different metrics to evaluate the performance of models on CO problems. We assess the quality of an individual solution $\boldsymbol{\sigma}$ by the associated value of the energy function $E_m(\boldsymbol{\sigma})$ which represents the size of the solution sets in MIS and MVC. Here m refers to the problem instance under consideration. Optimal solutions $\boldsymbol{\sigma}_{\text{opt}}$ are obtained with the Gurobi solver Gurobi Optimization, LLC [2023]. For models that generate solutions indeterministically we sample n_S different solutions $\boldsymbol{\sigma}_j$ per problem instance and calculate the test dataset average of the best relative error ϵ^*_{rel}:

$$\epsilon^*_{\text{rel}} = \frac{1}{M} \sum_{m=1}^{M} \min_j \frac{|E_m(\boldsymbol{\sigma}_{opt}) - E_m(\boldsymbol{\sigma}_j)|}{|E_m(\boldsymbol{\sigma}_{\text{opt}})|}, \tag{9}$$

where M denotes the number of problem instances in the test dataset. In case of deterministic algorithms only one sample is generated, i.e. $n_S = 1$. In several experiments it is insightful to investigate the average relative error $\hat{\epsilon}_{\text{rel}}$ for which we take the average instead of the minimum in Eq. 9. Analogously, we also define the best approximation ratio AR^* by

$$\text{AR}^* = \frac{1}{M} \sum_{m=1}^{M} \min_j \frac{|E_m(\boldsymbol{\sigma}_j)|}{|E_m(\boldsymbol{\sigma}_{\text{opt}})|}, \tag{10}$$

The average approximation ratio $\widehat{\text{AR}}$ is defined by taking the average instead of the minimum operation in Eq. 10.

A.2 Ensuring Feasible Solutions

Since there is no rigorous guarantee that the model samples only feasible solutions that satisfy the constraints, we use a fast post processing procedure to make sure that only feasible solutions are sampled. Here, we make use of our choice of the relative weighting of the energy terms A and B (see Tab. 1) in the Ising formulation, which ensures that only feasible solutions are minima in the energy landscape. Therefore, we can detect violations when $E_B > 0$ and search for the spin that causes the largest amount of violations. Subsequently, we change the spin value of the node with the highest number of violations to satisfy the constraint and repeat the process until $E_B = 0$. We observe in our experiments that this post-processing step is typically unnecessary, since only in rare cases violating solutions are sampled.

A.3 Standardization of the Energy Scale

Since the energy scale of CO problems can vary significantly, a good choice of hyperparameters like the initial temperature T_0, learning rate and relative weighting between the policy and value loss can vary between different CO problem instances. Therefore, we standardize the energy scale that makes the choice of good hyperparameters easier. For this purpose we first express the binary energy function $E(q)$ (see. Tab 1) in terms of spins, i.e. as $E(\boldsymbol{\sigma})$ by substituting q_i with $\frac{\sigma_i+1}{2}$. We then sample states for CO problem instances from the training set by using a Random Greedy Algorithm (RGA, see App. A.5.2). Since we use only a few RGA steps (see App. A.5.2), this algorithm performs only slightly better than a completely random algorithm. From these states the mean energy $\hat{E}_{\text{RGA}}$ and the standard deviation $\text{std}(E_{\text{RGA}})$ over the training dataset is computed. Subsequently, we standardize the energy scale by:

$$\hat{E}(\boldsymbol{\sigma}) = \frac{E(\boldsymbol{\sigma}) - \hat{E}_{\text{RGA}}}{\text{std}(E_{\text{RGA}})}. \tag{11}$$

A.4 Mean Field Approximation

We also compare VAG-CO with our own implementation of a Mean Field Approximation (MFA). In MFA the probability of a state σ factorizes into a product of independent Bernoulli probabilities. Therefore, the state probability is given by:

$$p_\theta(\boldsymbol{\sigma}|E) = \prod_{i=1}^{N} p_\theta(\sigma_i|E). \tag{12}$$

A.4.1 Conditional Expectation

To obtain good samples from the MFA approach in a deterministic way, we adopt the Conditional Expectation (CE) method Raghavan [1988] as described in Karalias and Loukas [2020].

To introduce randomness into the CE-based solution generation procedure we initialize the random node features (which are processed by the GNN) of every node with a random vector with six binary entries that are drawn from a uniform distribution. Unless stated otherwise, we follow the procedure in Wang and Li [2023] and report the best CE result of eight different random node feature initializations.

A.5 Greedy Algorithms

A.5.1 Degree Based Greedy Algorithm

The Degree Based Greedy (DB-Greedy) algorithm Wormald [1995] is a polynomial time algorithm for the MIS problem. The DB-Greedy algorithm works in the following way: At first all nodes are sorted according to their degrees. Then, starting from the smallest degree the node is chosen to be part of the independent set. In the next step this node, its neighbors and their corresponding edges are deleted from the graph. These steps are repeated until the graph is empty.

This algorithm can also be applied to the MVC problem by using the fact that the complement of an independent set is a vertex cover. In other words, nodes in the independent set are excluded from the vertex cover and nodes that are not part of the independent set are included into the vertex cover.

A.5.2 Random Greedy Algorithm

The Random Greedy Algorithm (RGA) is a general approach that can be utilized for solving a broad range of CO problems that can be mapped to the Ising model. Initially, the algorithm randomly samples spin values with uniform probability. Then, for a fixed number of iterations a spin is randomly selected and its value is changed if it decreases the energy value. We set the number of iterations to $N \cdot n_R$, where N is the number of nodes in the graph and n_R is the number of repetitions per node.

For the purpose of the standardization the energy scale (see. App. A.3), we employ this algorithm with n_R set to one.

A.6 Study on Sampling Methods

In order to find out, how the best samples can be obtained with VAG-CO we study in Fig. 2 three different sampling methods on the RRG-100 MIS and RB-200 MVC dataset. Here we evaluate ϵ^*_{rel} on the test dataset and plot it over the number of samples n_S that are used for each graph in the dataset. We also add MFA and DB-Greedy results, where we show for MFA how the method improves when more solutions are sampled. Since DB-Greedy is a deterministic algorithm, we draw a horizontal line that indicates the solution quality of the algorithm. To draw a relation to results that are presented in Fig. 1 (Left, Middle) we also add a vertical dotted line at $n_S = 8$ samples. For VAG-CO we denote the first method as *sampling* (S), where for each graph n_S solutions are sampled according to the corresponding probability distribution. In the second VAG-CO sampling method called *ordered sampling* (OS), we use for each graph n_O different BFS node orderings and sample n_S/n_O states per graph ordering. Finally, we sample VAG-CO solutions with a method called *ordered greedy* (OG), where we generate solutions greedily for each BFS ordering with $n_O = n_S$. Results in Fig. 2 show that the sampling strategy OG always outperforms all other sampling strategies that we proposed for VAG-CO. Additionally, we see that as we increase n_O in the OS sampling strategy, the performance improves consistently. Remarkably, DB-Greedy exhibits the best solution quality when MFA and VAG-CO are allowed only one sample ($n_S = 1$). However, DB-Greedy is outperformed by VAG-CO OG already with a modest amount of $n_S > 1$ samples.

A.7 Experimental Details

In this section we provide additional details to experiments that are presented in Sec. 6.

Parameter Checkpointing. With VAG-CO we checkpoint over the course of training the parameters that obtain the best ϵ^*_{rel} and the best $\hat{\epsilon}_{\text{rel}}$ on the validation set. For testing we always use the checkpoint with the best $\hat{\epsilon}_{\text{rel}}$, except for the results in Fig. 2 when the sampling strategies (S) and (OS) are used. Here we use the checkpoints with the best ϵ^*_{rel}.

Hyperparameter Tuning. For MFA-Anneal, the learning rate and initial temperature are tuned on the validation dataset of Enzymes MIS via a grid search. We considered learning rates (lr $\in [5\times10^{-4}, 1\times10^{-4}, 5\times10^{-5}]$) and initial temperatures ($T_0 \in [0.5, 0.25, 0.1, 0.05]$). With that, the number of GNN layers ($L \in [6, 8, 12]$) are tuned. Finally, the annealing duration (N_{anneal}) is increased until we observe that longer annealing does not lead to

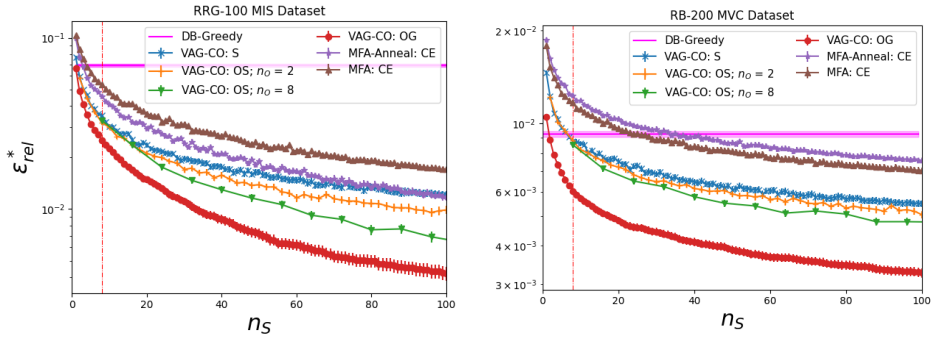

Figure 2: Ablation on different sampling strategies for VAG-CO on the RRG-100 MIS dataset (Left) and on the RB-200 MVC Dataset (Right). The averaged best relative error ϵ^*_{rel} on the test dataset is plotted over the number of solutions n_S per graph. Error bars indicate the standard error over the test dataset. TODO add explanation for OG and OS and S

improvements. For MFA, we tested on ENZYMES MIS the learning rates (lr $\in [5\times10^{-4}, 1\times10^{-4}, 5\times10^{-5}, 1\times10^{-5}]$) and with that tuned the number of GNN layers ($L \in [6, 8, 10, 12]$) and stop training when no significant improvement on the validation set is observed. In MFA and MFA-Anneal, when training on the other datasets we started with the optimal ENZYMES MIS parameters and further tested different learning rates (lr), initial temperatures (T_0) and number of GNN layers (L) individually on each dataset. For experiments on the RRG-100 MIS and RB-200 MVC we make a more extensive gridsearch. In RRG-100 MIS for MFA-Anneal we first tuned the initial temperature in the range of $T_0 \in [0.25, 0.1, 0.05, 0.035, 0.025, 0.015, 0.01]$ and in RB-200 MVC we search for the best initial temperature within the range of $T_0 \in [0.5, 0.25, 0.1, 0.05]$. For MFA and MFA-Anneal, we then iteratively search for the best learning rate in the range of lr $\in [5 \times 10^{-4}, 1 \times 10^{-4}, 5 \times 10^{-5}]$ and then for the best number of GNN layers $L \in [6, 8, 12]$. The best hyperparameters are listed in App. A.11.

On VAG-CO we iteratively tuned the learning rate (lr $\in [5 \cdot 10^{-4}, 1 \cdot 10^{-3}]$), initial temperature ($T_0 \in [0.05, 0.7, 0.1]$), number of GNN layers ($L \in [3, 4]$) and number of annealing steps ($N_{\text{anneal}} \in [3000, 6000]$) on the ENZYMES MIS validation dataset. We then initially test these hyperparameters on other datasets and adapt the number of annealing steps and the initial temperature. The choice of all hyperparameters in VAG-CO is listed in Tab. 5.

Ablation on Subgraph Tokenization. In the ablation on subgraph tokenization in Fig. 1 we keep hyperparameters the same except for the time horizon $\mathcal{T}$ and the hyperparameter λ in the PPO algorithm (see App. A.8). For the subgraph tokenization run we chose $\mathcal{T} = 20$, $\lambda = 0.95$ and $k = 5$. Therefore, the time horizon includes the generation of $\mathcal{T} \cdot k = 100$ spins per graph during the data collection phase (see App. A.8). If we would keep $\mathcal{T}$ the same, when we chose $k = 1$ for the run without subgraph tokenization only $\mathcal{T} \cdot k = 20$ spins would be generated for each graph. Therefore, we use $\mathcal{T} = 100$ when no subgraph tokenization is used. As we always set $\lambda = 1 - \frac{1}{\mathcal{T}}$ this hyperparameter is adapted accordingly.

Experiments on Random Regular Graphs. Since GNNs suffer from node ambiguity on Random Regular Graphs (RRGs) Wang and Li [2023] using random node features Abboud et al. [2021] can resolve this issue. Therefore, for the VAG-CO experiment in Fig. 1 (Left) we sample standard Gaussian random node features with the dimension of six and concatenate them to the graph representation of VAG-CO. For MFA and MFA-Anneal random node features are already used (see App. A.4.1) and had not to be added for the RRG experiments.

Averaged results on hard instances. In Fig. 1 (Left, Middle) only show results for specific generation parameters p on the RB-200 dataset and for specific values of d on the RRG-100 dataset. Here we report the corresponding averaged results on the RRG-100 MIS and RB-200 MVC dataset in Tab. 4. VAG-CO significantly outperforms all other methods for $n_S = 8$.

A.8 Proximal Policy Optimization

Proximal Policy Optimization (PPO) Schulman et al. [2017] is a popular RL algorithm that has two main components. The policy is represented by the network $p_\theta(\sigma_i|G_i)$ with parameters θ. The expected future reward is estimated by the value network $V_\phi(G_i)$ which is parameterized by ϕ.

Data Collection. In PPO the policy is rolled out through the environment in order to collect and store the states, actions, probabilities and the output of the value function into the rollout buffer. The data collection procedure is completed after going through $\mathcal{T}$ steps, where $\mathcal{T}$ is the so-called time horizon. Then, the advantages

	RB-200 MVC	RRG-100 MIS
Evaluation metric A.1	AR^*	AR^*
DB-Greedy	1.0092 ± 0.00022	0.931 ± 0.001
MFA: CE	1.0115 ± 0.00021	0.947 ± 0.001
MFA-Anneal: CE	1.0122 ± 0.00022	0.955 ± 0.001
VAG-CO (ours)	$\mathbf{1.0063 \pm 0.0002}$	$\mathbf{0.975 \pm 0.001}$

Table 4: Average approximation ratio AR^* on the test dataset of the RB-200 MVC and RRG-100 MIS dataset. Results where the AR^* is closer to one is better.

$A_i := A(\sigma_i, G_i)$ and value targets $V_i^T := V^T(G_i)$ are calculated by making use of the Generalised Advantage Estimation (GAE, Schulman et al. [2016]), where A_i is given by

$$A_i = \delta_i + (\gamma\lambda)\delta_{i+1} + ... + (\gamma\lambda)^{\mathcal{T}-t+1}\delta_{\mathcal{T}-1}, \tag{13}$$

with $\delta_i = R_i + \gamma V(G_{i+1}) - V(G_i)$ and $V_i^T = A_i + V(G_i)$. In our experiments the reward R_i is given by Eq. 6. In our experiments we chose $\gamma = 1.0$ and $\lambda = 1 - \frac{1}{\mathcal{T}}$. We set $\mathcal{T}$ to be approximately equal to the average graph size in the dataset. During the data collection phase we always collect data from $H = 30$ different CO problem instances and for each instance we sample $n_S = 30$ solutions.

Rollout Buffer. For the gradient updates in PPO, the loss is estimated with minibatches that are randomly sampled from the rollout buffer. As the rollout buffer contains data of size $H \times n_S \times \mathcal{T}$ we chose to sample data by first sampling H_{minib} graphs and then for each graph, we chose to sample N_{minib} solutions and for each solution S_{minib} time steps are sampled. We report the minibatch settings of each experiment in Tab. 5.

Training. As described before, during training minibatches of data D_{minib} is randomly sampled from the replay buffer. In PPO the overall loss that depends on the $L(\theta, \phi; D_{\text{minib}})$ is given by $L(\theta, \phi; D_{\text{minib}}) = L_1(\theta; D_{\text{minib}}) + c_V\, L_2(\phi; D_{\text{minib}})$, where the first term depends on the policy and the second term on the value function. To specify a minibatch sample from D_{minib}, we will use the index n.

The policy loss for one minibatch sample is then defined as

$$L_1(\theta)_n = -\min\left(I_n(\theta)\, A_n, clip(I_n(\theta), 1-\epsilon, 1+\epsilon)\, A_n\right), \tag{14}$$

where the importance sampling ratio $I_n(\theta) = \frac{p_\theta(\sigma_n|G_n)}{p_{\theta'}(\sigma_n|G_n)}$ is used to compute a weighted Monte Carlo estimate of A_n. Here, θ' represents the old policy parameters that were used to fill the rollout buffer during the data collection phase and θ are the new parameters that are used in gradient descend. In Eq. 14 the clipping function is used to prevent that the probability of the current policy $p_\theta(\sigma_n|G_n)$ differs by more than a factor $1 \pm \epsilon$ from the old policy $p_{\theta'}(\sigma_n|G_n)$.

Similarly, the value function loss for one sample is given by $L_2(\phi)_n = (V_\phi(G_n) - V_{\phi'}^T(G_n))^2$.

The loss $L(\theta, \phi; D_{\text{minib}})$ is then updated for each minibatch in the rollout buffer so that each sample is used at least once. This procedure is overall repeated n_{repeat} times, before we increment the epoch step N_{epoch} and new data is collected with the updated set of parameters. In our experiments we always chose $\epsilon = 0.1$, $c_V = 0.5$ and $n_{\text{repeat}} = 2$.

A.9 Model Details and GNN Architectures

Before processing the graph representation with GNNs, we encode its node features with an encoder MLP. To obtain node embeddings $\mathcal{N}_\rangle^{\,l} \in \mathbb{R}^{d(l)}$ with $l \in [1, \ldots, L]$ that depend on the graph structure, the encoded node features are processed by L layers of GNNs. Afterwards, we apply a global sum aggregation to compute a global graph embedding. The global graph embedding is then concatenated to the node embedding $\mathcal{N}_i^L$, where i is the index of the spin whose value is to be generated (see Sec. 3). We then use this embedding to calculate the policy and value function outputs (see App. A.8) with two separate MLPs.

In our experiments, we employ two distinct message passing architectures. The first one is a Message-passing Neural Network (MPNN) Battaglia et al. [2018] but with linear message functions. The second architecture also includes skip connections (MPNN-skip) Battaglia et al. [2018] which we use when more than three GNN layers are used.

Message Passing Neural Network.

Our MPNN layer is defined by the following update for node embeddings:

$$\mathcal{N}_i^{l+1} = \Psi\left(\mathcal{N}_i^l, \sum_{j \in N(i)} (W_{\mathcal{N}} \mathcal{N}_j^l + W_{\mathcal{E}}\, \mathcal{E}_{ij}^0)\right), \tag{15}$$

Where $\mathcal{N}_i^{l+1} \in \mathbb{R}^{d(l+1)}$ is the node embedding of the next layer $l+1$, $N(i)$ is the neighborhood of node i, $W_{\mathcal{N}} \in \mathbb{R}^{d_\Psi \times d(l)}$ and $W_{\mathcal{E}} \in \mathbb{R}^{d_\Psi \times 1}$ are weight matrixes and Ψ is the node MLP. $\mathcal{E}_{ij}^0$ is the initial edge feature between node i and j of the graph representation G (see Sec. 3).

Message Passing Neural Network with Skip Connections. In case of MPNN-skip we adapt the message passing layer to

$$\mathcal{N}_i^{L+1} = \ln\left(\Psi\left(\mathcal{N}_i^L, \sum_{j \in N(i)} (W_{\mathcal{N}} \mathcal{N}_j^L + W_{\mathcal{E}}\, \mathcal{E}_{ij}^0)\right) + W_{skip}\, \mathcal{N}_i^L\right), \tag{16}$$

where the skip connection is implemented by adding the term $W_{skip}\, \mathcal{N}_i^L$ after the node update, where $W_{skip} \in \mathbb{R}^{d(l+1) \times d(l)}$ is a weight matrix. Finally, a layer norm layer $\ln(\cdot)$ Ba et al. [2016] is applied as it is commonly done in Neural Network architectures when skip connections are used Liu et al. [2021].

In VAG-CO we use a policy and value MLP with three layers. Both MLPs have 120 neurons in the first two layers. The value MLP has only one neuron in the output layer and the policy MLP has 2^k output neurons. For the encoder MLP we use a two layer MLP with 40 neurons in each layer. In case of the GNN we use a two layer node MLP Ψ with 40 neurons in each layer. For the weight matrixes $W_{\mathcal{N}}$ and $W_{\mathcal{E}}$ we set $d_\Psi = 30$. Layer normalization is applied after every layer within a MLP except in the output layer of the policy and value MLP. We use Rectified Linear Unit (ReLU) activation functions Agarap [2018] except in the output of the policy MLP, where a softmax activation is used and except of the output of the value MLP where no activation is used.

In MFA with and without annealing the model uses an encoder MLP with one layer and 64 neurons. Our MFA networks always use the MPNN-skip GNN layers, where the weight matrixes $W_{\mathcal{N}}$, $W_{\mathcal{E}}$ and W_{skip} have 64 neurons. The node MLP Ψ has two layers with 64 neurons each and in the output MLP the first three layers have 64 neurons each and the output layer has 2 neurons with a softmax activation. As in VAG-CO we always use ReLU activations and layer norm within MLPs.

A.9.1 Subgraph Tokenization

For subgraph tokenization we have to make changes to the one-hot encoding as described in Sec. 3 and also to the model architecture as described in App. A.9. The one-hot encoding in the graph representation is adapted so that spins $\sigma_{i:i+k}$ that are going to be generated receive an enumeration that indicates their position in the sliced list of BFS-ordered (see Sec. 3]) indices $i : i+k$. Then, the graph G_i is processed by a GNN that provides a node embedding $GNN_\theta(x_i)$ for each node. Along with a global sum aggregation, the node embeddings of the spins that are going to be generated are then concatenated according to their BFS order (Sec. 3) and further processed by the policy MLP that calculates the probability for each of the 2^k spin configurations using a softmax output layer.

When the number of nodes N in the graph is not dividable by k, the number vertices in the CO problem instance description has to be increased without changing the inherent optimization objective. This can be realised by adding a sufficient amount of spins into the Ising model (see Sec. 2) with zero spin weight B and no connections to other spins.

A.10 Training Time and Computational Resources

All runs with VAG-CO were conducted either on A100 Nvidia GPU with 40 GB Memory or an A40 Nvidia GPUs with 48 GB Memory. In case of COLLAB MVC an A100 with 80 GB Memory is used.

The training time of our algorithm depends hyperparameters like the number of annealing steps N_{anneal} (see App. A.12), the number of edges and nodes of graphs in the dataset, on the GPUs that are used during training, on the time horizon and on the minibatch size that is used for gradient updates in PPO (see App. A.8). For example the TWITTER MVC run with $N_{\text{anneal}} = 4000, \mathcal{T} = 30, H_{\text{minib}} = N_{\text{minib}} = S_{\text{minib}} = 10$ takes one day, when trained on an A100 40 GB Nvidia GPU. The run on ENZYMES MIS trained on a A40 GPU takes ten hours, when trained with $N_{\text{anneal}} = 6000, \mathcal{T} = 20, H_{\text{minib}} = N_{\text{minib}} = 15$ and $S_{\text{minib}} = 10$.

A.11 Hyperparameters

VAG-CO Hyperparameters.

All hyperparameters that change across datasets are listed in Tab. 5, whereas hyperparameters that stay the same are specified in the corresponding section (e.g. App. A.9, A.12).

Dataset	CO Problem Instance	learning rate	N_{anneal}	T_0	L	N_{minib}	H_{minib}	S_{minib}	$\mathcal{T}$
TWITTER	MVC	1×10^{-3}	4000	0.05	3	10	10	10	30
COLLAB	MVC	1×10^{-3}	12000	0.05	3	10	10	6	35
	MIS	1×10^{-3}	4000	0.05	3	10	10	10	30
IMDB-BINARY	MVC	1×10^{-3}	2000	0.05	3	10	10	10	5
	MIS	5×10^{-4}	2000	0.05	3	10	10	10	5
ENZYMES	MIS	5×10^{-4}	6000	0.07	4	15	15	15	25
PROTEINS	MIS	5×10^{-4}	10000	0.1	3	15	15	10	35
MUTAG	MIS	1×10^{-3}	2000	0.05	3	10	10	10	7
RB-100	MVC	1×10^{-3}	2000	$5 \cdot 10^{-2}, 5 \cdot 10^{-3}, 0.0$	3	10	10	5	20
RB-200	MVC	5×10^{-4}	20000	0.05	3	15	10	10	30
RRG-100	MIS	5×10^{-4}	20000	0.1	4	10	10	10	20

Table 5: Hyperparemeters that are used in VAG-CO on different datasets.

MFA Hyperparameters.

All runs with MFA used a batch size H of 32 and we sampled $n_S = 30$ solutions. Furthermore, we used 8 GNN layers, and 6 random node features per node. For MFA-Anneal we used a learning rate of 1×10^{-4}, 2000 annealing steps N_{anneal} and a start temperature T_0 of 0.1 except for RRG-100 MIS where a T_0 of 0.015 has been used. For MFA without annealing we trained for 2000 epochs and used a learning rate of 5×10^{-5} except for ENZYMS MIS where a learning rate of 1×10^{-4} has been used.

A.12 Annealing Schedule

As described in Sec. 3, we change the temperature in the reward Eq. 6 according to a predefined annealing schedule. During the warm-up phase of N_{warmup} epochs the temperature is held constant at the initial temperature T_0. Afterwards the following temperature schedule is applied:

$$T(N_{\text{epoch}}) = T_{\text{anneal}}(N_{\text{epoch}}) \cdot \left[\cos\left(2\pi(\lambda + \frac{1}{2}) \frac{N_{\text{epoch}} - N_{\text{warmup}}}{N_{\text{anneal}}} \right) + 1 \right]. \tag{17}$$

Here, $T_{\text{anneal}}(N_{\text{epoch}})$ is the gradually decreasing amplitude of the temperature oscillations:

$$T_{\text{anneal}}(N_{\text{epoch}}) = \frac{T_0}{1 + c \cdot \frac{N_{\text{epoch}} - N_{\text{warmup}}}{N_{\text{anneal}}}}. \tag{18}$$

Here c is a scaling factor that determines the slope of $T_{\text{anneal}}(N_{\text{epoch}})$. The parameter λ determines the number temperature oscillations in the schedule and N_{anneal} is the total number of epochs that follow after the warmup phase. We use $\lambda = 3$, $N_{\text{warmup}} = 400$ and $c = 6$. The course of the annealing schedule is illustrated in Fig. 3. One reason for the usage of cosine modulation function is rather practical, namely that $T = 0$ is reached multiple times during training, which allows an assessment of the training success at an earlier stage for a given set of hyperparmeters. In the absence of cosine modulation, we found it harder to assess the final performance before the entire annealing was finished. Similar periodic schedules for learning rates have been proposed as variants of Stochastic Gradient Descent Loshchilov and Hutter [2017].

A.13 Derivations

A.13.1 Free-Energy Decomposition into Rewards

In the following we show that using the reward defined in Eq. 6 is consistent with the goal of minimizing the free-energy defined in Eq. 2.

The right-hand side of Eq. 2 contains the expectation of the energy $E(\boldsymbol{\sigma})$ and a term that is proportional to the entropy of $p_\theta(\boldsymbol{\sigma})$. For the energy we obtain the following decomposition into individual steps i of the solution generation process (Sec. 3):

$$E(\boldsymbol{\sigma}) = \sum_{i=1}^{N} \left[\sum_{j<i} J_{ij}\sigma_j\sigma_i + B_i\sigma_i \right] = \sum_{i=1}^{N} \sigma_i \left[\sum_{j<i} J_{ij}\sigma_j + B_i \right] = \sum_{i=1}^{N} \Delta E_i \tag{19}$$

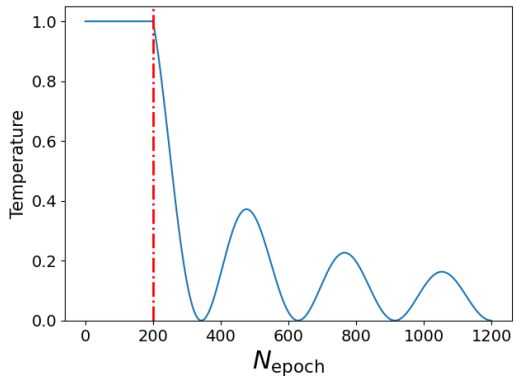

Figure 3: Illustration of the cosine modulated annealing schedule. The plot depicts the annealing schedule with $N_{\text{warmup}} = 200$ and $N_{\text{anneal}} = 1000$ steps. The vertical red line marks the end of the warmup phase.

By using an autoregressive factorization the entropy can also be decomposed in the following way:

$$\begin{aligned} S\big(p_\theta(\boldsymbol{\sigma}|E)\big) = -\sum_{\boldsymbol{\sigma}} p_\theta(\boldsymbol{\sigma}|E) \log p_\theta(\boldsymbol{\sigma}|E) &= -\sum_{\boldsymbol{\sigma}} p_\theta(\boldsymbol{\sigma}|E) \left[\sum_{i=1}^{N} \log p_\theta(\sigma_i|\sigma_{<i}, E) \right] \\ &= -\mathop{\mathbb{E}}_{\boldsymbol{\sigma}\sim p_\theta} \left[\sum_{i=1}^{N} \log p_\theta(\sigma_i|\sigma_{<i}, E) \right] \end{aligned} \tag{20}$$

Therefore, we can use this decomposition by using the reward $R_i(G_i, \beta) = -\left[\Delta E_i + \frac{1}{\beta} \log p(\sigma_i|\sigma_{<i}, E)\right]$. By the relation $F(\theta; \beta, E) = -\mathbb{E}_{\boldsymbol{\sigma}\sim p_\theta}[\sum_i R_i(G_i, \beta)]$, maximizing this reward will then be equivalent to minimizing the free-energy.

A.13.2 Details on Remark 1

We first restate Corollary 5 of Dudík et al. [2007] in our notation. In the context of this Corollary our energy function E can be regarded as a single feature and, therefore, we use their result for $n = 1$. In addition, we consider the case in which the samples $\mathcal{S}$ are drawn from the target distribution, i.e. $\pi = p(\beta^*)$. Since the terms Boltzmann distribution and Gibbs distribution can be used interchangeably we obtain:

Corollary 1 (Corollary 5 of Dudík et al. [2007])**.** *Assume that E is bounded in $[0, 1]$. Let $\hat{\beta}$ minimize $\mathcal{L}_{\hat{p}}(\beta) + \lambda|\beta|$ with $\lambda = \sqrt{\ln(2/\delta)/(2m)}$. Then with probability at least $1 - \delta$ for every Boltzmann distribution $p(\beta)$,*

$$\mathcal{L}_{p(\beta^*)}(\hat{\beta}) \leqslant \mathcal{L}_{p(\beta^*)}(\beta) + \frac{|\beta|}{\sqrt{m}} \sqrt{2\ln(2/\delta)}.$$

Now we consider the result for the case $\beta = \beta^*$ and subtract $\mathcal{L}_{p(\beta^*)}(\beta^*)$. Remark 1 follows by applying the definition of the Kullback-Leibler divergence D_{KL} to the left-hand side.
We note that an equivalent statement, however, with a consideration the Rademacher complexity of E can be derived from Theorem 2 in Cortes et al. [2015].