Conditional Tree Matching for Inference-Time Adaptation of Tree Prediction Models

Harshit Varma [1] **Abhijeet Awasthi** [1] **Sunita Sarawagi** [1]

Abstract

We present CTREEOT, a convergent, differentiable algorithm for matching two trees when each tree is conditioned on some input. Such conditional tree matching is useful for light-weight, few-shot adaptation of tree prediction models without parameter fine-tuning. CTREEOT includes an alignment algorithm that extends the popular Sinkhorn algorithm for matching tree nodes while supporting constraints on tree edges. The algorithm involves alternating between matrix rescaling and message passing updates, and can be efficiently expressed as GPU tensor operations. The second part of CTREEOT is fine-grained relevance-based reweighting of nodes that makes the match scores useful for prediction tasks. We demonstrate the usefulness of CTREEOT for cross-schema adaptation of Text-to-SQL, a popular semantic parsing task. We show that compared to state-of-the-art methods, we achieve significant increase in adaptation accuracy.

1. Introduction

Recently, memory-based methods have been established as a promising approach for inference time adaptation with a few related examples (cases) in tasks like translation (Khandelwal et al., 2021; 2020), semantic parsing (Pasupat et al., 2021; Gupta et al., 2021; Awasthi et al., 2023), and gaming (Atzeni et al., 2022). Given an input text $\mathbf{x}$, and a few related labeled cases $\mathcal{C} = \{(\tilde{\mathbf{x}}_1, \tilde{\mathbf{y}}_1), \ldots, (\tilde{\mathbf{x}}_C, \tilde{\mathbf{y}}_C)\}$, memory-based methods typically assign a match score for a proposed prediction $\mathbf{y}$ for input $\mathbf{x}$ over each case $(\tilde{\mathbf{x}}_c, \tilde{\mathbf{y}}_c) \in \mathcal{C}$. The match score after suitable pooling over all cases and the prediction model should score the correct prediction higher than incorrect candidate predictions.

In tasks like semantic parsing, where the predicted $\mathbf{y}$ is a tree, designing a sound conditional scoring function raises several challenges. The score should be structure-aware, for example, it should satisfy topological constraints imposed by the tree, or reward preserving of tree edges. For efficient training, the scoring function should be differentiable and computationally efficient for scoring several candidates for a given $\mathbf{x}$. Further, the match of a tree $\mathbf{y}$ with another $\mathbf{y}_c$ has to be carefully conditioned on corresponding input text $\mathbf{x}$ and $\mathbf{x}_c$ respectively. Otherwise, the score may not be useful for promoting the prediction of the correct $\mathbf{y}$ as against several other incorrect $\mathbf{y}'$, which may be more similar to some case $\mathbf{y}_c$.

A standard approach is to assign to each tree $\mathbf{y}$ a contextual embedding conditioned on its input $\mathbf{x}$, and use vector similarity to match them (Awasthi et al., 2023). This approach, we will show, is too coarse-grained since it collapses an entire tree into a fixed dimensional vector. Many methods have been proposed for fine-grained node-level alignment-based differentiable matching of graphs, but they are not designed for conditional matching. Also, most earlier methods co-embed the two graphs compared (Li et al., 2019; Zhang et al., 2021), and are not suitable when the cross product of number of cases and candidate trees is large.

In this paper, we propose an efficient, differentiable, structure-aware, convergent, conditional tree matching method called CTREEOT. Our method first independently assigns embeddings to each node for each tree that is contextualized on the input $\mathbf{x}$ and other nodes of the tree. We then align nodes of the two trees using a differentiable, convergent algorithm by designing an extension of the well-known set-based Sinkhorn algorithm for trees. Existing algorithms that regularize the Sinkhorn method with graph constraints (Titouan et al., 2019), do not guarantee convergence, unlike our extension. Further, we show that $\mathbf{x}$-contextual embeddings alone fail to provide precise conditional scoring. We design a node-level relevance scoring model, that when weighted with the alignment scores, provides more discriminating conditional match scores. We provide an implementation of the algorithm in terms of tensor operations that can be efficiently executed on GPUs. Thus, we provide a convergent tree matching layer that can be integrated in a deep network.

[1]Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Bombay. Correspondence to: Harshit Varma, Sunita Sarawagi <{harshitvarma, sunita}@cse.iitb.ac.in>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

Contributions (1) We provide a new algorithm CTREEOT for aligning two trees that extends the Sinkhorn algorithm for matching tree nodes while supporting edge-level constraints, e.g. penalizing topological order violations. The algorithm is guaranteed to converge, and can be efficiently implemented via tensor operations on GPUs. (2) We propose a conditional tree matching score that refines alignment weighted scores with learned node-level relevance weights. (3) We present experiments on inference time adaptation of the Text-to-SQL task and obtain significant gains in accuracy compared to existing methods.

2. Preliminaries

We use $\mathbf{x} \in \mathcal{X}$ (space of sentences) to denote the input text. Each $\mathbf{x}$ is associated with a gold tree $\mathbf{y} = \{(y_1, \ldots, y_n), E\}$ with n discrete nodes $y_1, \ldots, y_n$ and a set E of edges. We use $(y_i, y_s) \in E$ to denote that y_i is a parent of y_s. We assume that the nodes are topologically ordered so that $i < s$, $\forall (y_i, y_s) \in E$, and we denote it by $y_i \prec_E y_s$. For example, in a Text-to-SQL task, the trees could refer to the relational algebra (RA) trees corresponding to SQL queries.

We assume a base model $M_\theta(\mathbf{x})$ that generates for an input text $\mathbf{x}$ a set of candidate trees $\mathcal{Y}_x$. These could be the result of a beam search on the tree prediction model. Each candidate tree $\hat{\mathbf{y}} \in \mathcal{Y}_x$, is assigned a score $s_\theta(\hat{\mathbf{y}}|\mathbf{x})$ from the base model $M_\theta(\mathbf{x})$. We assume the model also assigns to each node y_i in a candidate tree an embedding vector contextualized on the input $\mathbf{x}$ and other nodes in the tree as $\mathbf{z}_i = \text{Embed}_\theta(y_i, \mathbf{y}, \mathbf{x})$.

We use $\mathcal{C}_x = \{(\tilde{\mathbf{x}}_1, \tilde{\mathbf{y}}_1), \ldots, (\tilde{\mathbf{x}}_C, \tilde{\mathbf{y}}_C)\}$ to denote a set of related cases available for an input text $\mathbf{x}$. Typically, a retrieval model $R_\Phi(\mathbf{x}, L)$ might be used to find related cases from a larger set of available labeled cases L. During deployment L may include new labeled examples provided by the user for few-shot adaptation. Specific instantiations of the base model M_θ and retrieval model R_Φ are discussed in the experiment section. The framework we present here is independent of the details of these components.

Our goal is to use the related cases $\mathcal{C}_x$ to assign scores $s_\psi(\hat{\mathbf{y}}|\mathbf{x}, \mathcal{C}_x)$ so that $s_\theta(\hat{\mathbf{y}}|\mathbf{x}) + s_\psi(\hat{\mathbf{y}}|\mathbf{x}, \mathcal{C}_x)$ provides higher accuracy than the base scores s_θ alone by scoring correct trees higher than incorrect trees.

3. Our Method

For computing the score $s_\psi(\hat{\mathbf{y}}|\mathbf{x}, \mathcal{C}_x)$ of a candidate tree $\hat{\mathbf{y}} \in \mathcal{Y}_x$ using the cases we first individually treat each case $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \in \mathcal{C}_x$ and compute a conditional tree matching score $\text{CTS}(\hat{\mathbf{y}}|\mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ with respect to that. We then pool together these scores and reweight them vis-a-vis the model scores.

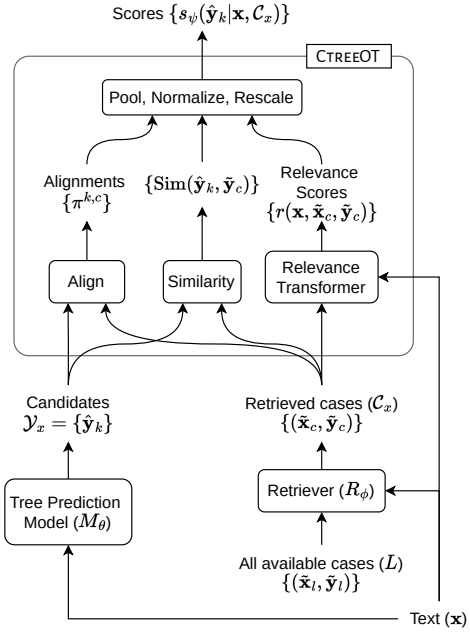


Figure 1. Overall working of CTREEOT

We describe how we pool the scores later in § 3.3. We elaborate now on how we compute $\text{CTS}(\hat{\mathbf{y}}|\mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})$. We want the score to be high when $\hat{\mathbf{y}}$ and $\tilde{\mathbf{y}}$ are similar *and* they appear in similar contexts $\mathbf{x}$ and $\tilde{\mathbf{x}}$. We achieve this by combining two types of computation. (1) **Alignment:** First, we define pairwise similarity between nodes of the two trees using their contextual embeddings. Between nodes $\hat{y}_i \in \hat{\mathbf{y}}$ and $\tilde{y}_j \in \tilde{\mathbf{y}}$ we define $\text{Sim}(\hat{y}_i, \tilde{y}_j)$ as cosine similarity between the contextualized representations $\hat{\mathbf{z}}_i$ and $\tilde{\mathbf{z}}_j$ of the respective nodes $\hat{y}_i$ and $\tilde{y}_j$. Thus, $\text{Sim}(\hat{y}_i, \tilde{y}_j) = \frac{\langle \hat{\mathbf{z}}_i, \tilde{\mathbf{z}}_j \rangle}{||\hat{\mathbf{z}}_i|| ||\tilde{\mathbf{z}}_j||}$.

We align nodes of trees $\hat{\mathbf{y}}$ with $\tilde{\mathbf{y}}$ based on similarity of node embeddings, while imposing any penalties expressible as edge costs, such as those that penalize violations of topological ordering in the alignments. We present an efficient differentiable algorithm for this alignment in § 3.1. The algorithm returns alignment $\pi_{ij} \in [0, 1]$ for all node pairs i, j in $\hat{\mathbf{y}}$ and $\tilde{\mathbf{y}}$ respectively. **(2) Relevance:** Even if two trees are structurally identical, we do not want to assign them a high score if they are associated with very different input texts $\mathbf{x}$ and $\tilde{\mathbf{x}}$. Also, a case tree $\tilde{\mathbf{y}}$ may be partially relevant to a candidate tree $\hat{\mathbf{y}}$. To allow for such fine-grained relevance-aware matching, we associate with each node of each case tree a relevance score r_j given an $\mathbf{x}$. Instead of directly summing over the similarity of aligned nodes y_j, we further weight each node in a case tree with a relevance score $r_j \in [0, 1]$ to establish its relevance to the current instance $\mathbf{x}$, as per Equation 1. In § 3.2 we present how such relevance weights are obtained. The final scoring equation for each $\hat{\mathbf{y}} \in \mathcal{Y}_x$ is:

$$\mathrm{CTS}(\hat{\mathbf{y}}|\tilde{\mathbf{y}},\mathbf{x},\tilde{\mathbf{x}}) = \sum_{i,j} \pi_{i,j}\, r_j\ \mathrm{Sim}(\hat{y}_i,\tilde{y}_j) \quad \forall\, \tilde{\mathbf{y}} \in \mathcal{C}_x \quad (1)$$

We denote the pooling of the scores over all the cases as:

$$s_\psi(\hat{\mathbf{y}}|\mathbf{x},\mathcal{C}_x) = \mathrm{Pool}(\{\mathrm{CTS}(\hat{\mathbf{y}}|\tilde{\mathbf{y}},\mathbf{x},\tilde{\mathbf{x}})\} : (\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \in \mathcal{C}_x)$$

Figure 1 shows a schematic diagram of the overall working of our method.

3.1. Alignment of Candidate and Case Trees

We next present our algorithm for aligning nodes of trees $\hat{\mathbf{y}}$ and $\tilde{\mathbf{y}}$, while penalizing edge-level constraints such as topological ordering introduced by their edges $\hat{E}, \tilde{E}$ respectively. We denote the cost of aligning a node $y_i \in \hat{\mathbf{y}}$ with a node $y_j \in \tilde{\mathbf{y}}$ with $\phi_{i,j} \in \mathbb{R}$. We define cost as negative of node similarities $\mathrm{Sim}(\hat{y}_i,\tilde{y}_j)$. We represent the alignment between the nodes of $\hat{\mathbf{y}}$ ($|\hat{\mathbf{y}}| = \hat{m}$) and $\tilde{\mathbf{y}}$ ($|\tilde{\mathbf{y}}| = \tilde{m}$) using a $\hat{m} \times \tilde{m}$ boolean matrix π, where $\pi_{i,j} = 1$ if $\hat{y}_i$ is aligned with $\tilde{y}_j$ and 0 otherwise. To allow nodes on each tree to not align, we introduce $\tilde{m}$ disconnected dummy nodes in $\hat{\mathbf{y}}$ and $\hat{m}$ disconnected dummy nodes in $\tilde{\mathbf{y}}$ with a cost of 0. Thus, the total number of nodes in each tree is $n = \hat{m} + \tilde{m}$.

Our alignment algorithm supports edge-level costs attached to alignments. We show how edge costs can be used to penalize violations of topological ordering of nodes based on their edges. For example, let nodes $(\hat{y}_i, \hat{y}_s)$ in $\hat{\mathbf{y}}$ are connected such that $i \prec_{\hat{E}} s$ and $(\tilde{y}_j, \tilde{y}_t)$ in $\tilde{\mathbf{y}}$ are connected such that $j \prec_{\tilde{E}} t$. The alignments between nodes $(\hat{y}_i, \tilde{y}_t)$ and $(\hat{y}_s, \tilde{y}_j)$, resulting in a crossover is undesirable, since the aligned nodes do not respect the topological ordering enforced by edges $(\hat{y}_i, \hat{y}_s)$ and $(\tilde{y}_j, \tilde{y}_t)$. We express this constraint by assigning a cost $\phi_{is,jt}$ when node $i \in \hat{\mathbf{y}}$ is aligned to $j \in \tilde{\mathbf{y}}$ and node $s \in \hat{\mathbf{y}}$ is aligned to $t \in \tilde{\mathbf{y}}$. The cost of violating topological order is expressed as:

$$\phi_{is,jt} \begin{cases} = \lambda \text{ if } i \prec_{\hat{E}} s \text{ and } j \succeq_{\tilde{E}} t \\ = 0 \ \text{ otherwise} \end{cases} \quad (2)$$

where $\lambda > 0$ is a fixed cost for penalizing crossovers. Further, a dummy node in $\tilde{\mathbf{y}}$ preceeds all non-dummy nodes, thus $j \prec_{\tilde{E}} t$ when j is dummy and t is non-dummy. Note although the constraint needs to hold for all $i \prec_{\hat{E}} s$ pairs in $\hat{\mathbf{y}}$, it suffices to enforce it only on the edges.

Combining the node and edge costs, the minimum cost alignments between the trees $\hat{\mathbf{y}}$ and $\tilde{\mathbf{y}}$ can be obtained as a solution to the constrained optimization problem in Equation 3.

$$\begin{aligned} \min_{\pi \in \{0,1\}^{n^2}} \quad & \sum_{i,j} \phi_{i,j}\pi_{i,j} + \sum_{(i,s)\in\hat{E}} \sum_{(j,t)} \phi_{is,jt}\pi_{i,j}\pi_{s,t} \\ \text{s.t.} \quad & \sum_{i=1}^{n} \pi_{i,j} = 1 \text{ and } \sum_{j=1}^{n} \pi_{i,j} = 1 \end{aligned} \quad (3)$$

The optimization constraints ensure that each node $y_i \in \hat{\mathbf{y}}$ is matched with exactly one node in $\tilde{y}_j \in \tilde{\mathbf{y}}$, and vice-versa. However, solving this objective is inefficient. The objective without the edge costs reduces to the well-known optimal transport problem over discrete sets. Exact optimal solutions to the OT problem have been attempted using linear programs or network flow algorithms (Ahuja et al., 1993) but these entail cubic cost and are not considered practically useful for integration in a deep network. Other methods based on solving the semi-dual using subgradients (Kitagawa et al., 2019) are also considered slow. Also, it is not differentiable and not easily integrated as a layer in a deep network. We next explore differentiable formulations.

A Differentiable Relaxed Formulation We propose two modifications to the above objective (Equation 3) to make the solution tractable and differentiable. First, following prior work (Cuturi, 2013; Peyré & Cuturi, 2019), we allow the alignment matrix π to take continuous values and we regularize the objective in Equation 3 with a negative self-entropy term over the elements of the alignment matrix π. With these changes, our objective without the edge costs, can be solved using the well-known Sinkhorn algorithm. While regular Sinkhorn converges, convergence is not guaranteed with the edge costs expressed as the quadratic terms in the objective of Equation 3. We propose an alternative formulation by introducing new edge alignment variables along with corresponding constraints to ensure consistency with the node alignment variables. For each edge $(i,s) \in \hat{E}$ we introduce variables $\pi_{is,jt}$ to denote that node $i \in \hat{\mathbf{y}}$ is aligned to $j \in \tilde{\mathbf{y}}$ and node $s \in \hat{\mathbf{y}}$ is aligned to $t \in \tilde{\mathbf{y}}$. Our final formulation with these two types of variables is:

$$\begin{aligned} \min_{\pi} \quad & \frac{1}{\epsilon}\sum_{i,j} \phi_{i,j}\pi_{i,j} + \frac{1}{\epsilon}\sum_{(i,s)\in\hat{E}} \sum_{j,t} \phi_{is,jt}\pi_{is,jt} \\ & + \sum_{ij} \pi_{i,j}(\log \pi_{i,j} - 1) \\ & + \sum_{(i,s)\in\hat{E}} \sum_{j,t} \pi_{is,jt}(\log \pi_{is,jt} - 1) \\ \text{s.t.} \quad & \sum_{i=1}^{n} \pi_{i,j} = 1\,; \sum_{j=1}^{n} \pi_{i,j} = 1 \\ & \sum_{j=1}^{n} \pi_{is,jt} = \pi_{s,t}, \quad \forall (i,s) \in \hat{E}, \forall t \in [n] \\ & \sum_{t=1}^{n} \pi_{is,jt} = \pi_{i,j}, \quad \forall (i,s) \in \hat{E}, \forall j \in [n] \end{aligned} \quad (4)$$

In the above, $\epsilon > 0$ denotes the weight of the node costs vis-a-vis the entropy regularizer, which is applied on both the node-level and edge-level alignment variables. We introduced two new constraints to make the edge alignment variables consistent with the node alignment variables, much

like in inference algorithm for graphical models. An advantage of the above formulation is that we now have a strictly convex objective with linear constraints. Strong duality holds. A feasible solution is when all variables are uniform. We solve the objective by writing its dual.

The Dual The dual of the above objective in terms of dual variables $\{u_i\}_{i=1}^n$, $\{v_j\}_{j=1}^n$, $\bigcup_{(i,s)\in\hat{E},j,t}\{m_{s\to ij}, m_{i\to st}\}$ following standard techniques (Boyd & Vandenberghe, 2004) can be written as:

$$\begin{aligned}\min_{u,v,m} &\sum_i u_i + \sum_j v_j \\ &+ \sum_{i,j} \exp\left(-\frac{\phi_{i,j}}{\epsilon} - u_i - v_j + \sum_{s\in\text{nbr}(i)} m_{s\to ij}\right) \\ &+ \sum_{(i,s)\in\hat{E}} \sum_{j,t} \exp\left(\frac{-\phi_{is,jt}}{\epsilon} - m_{i\to st} - m_{s\to ij}\right)\end{aligned} \tag{5}$$

where the primal alignments variables can be recovered from the dual solutions as:

$$\pi_{ij} = \exp\left(\frac{-\phi_{i,j}}{\epsilon} - u_i - v_j + \sum_{s\in\text{nbr}(i)} m_{s\to ij}\right) \tag{6}$$

The dual variables u_i, v_j are similar to the corresponding variables in the Sinkhorn algorithm. The $\{m_{i\to st}\}$ (and $\{m_{s\to ij}\}$) variables can be viewed as message variables that denote the message that node i sends to node s along edge (i, s) on the score of aligning s to node t of the other tree.

Dual Updates The dual variables are unconstrained, the objective is convex on the dual variables, and we solve them using block coordinate ascent. By taking the derivative with respect to each type of dual variables and equating the gradient to zero, we obtain the following updates:

$$u_i = \log\sum_j \exp\left(\sum_{s\in\text{nbr}(i)} m_{s\to ij} - \frac{\phi_{i,j}}{\epsilon} - v_j\right)$$

$$v_j = \log\sum_i \exp\left(\sum_{s\in\text{nbr}(i)} m_{s\to ij} - \frac{\phi_{i,j}}{\epsilon} - u_i\right)$$

$$\begin{aligned}m_{i\to st} = &\frac{1}{2}\log\sum_j \exp\left(-m_{s\to ij} - \frac{\phi_{is,jt}}{\epsilon}\right) \\ &+ \frac{1}{2}\left(\frac{\phi_{s,t}}{\epsilon} + u_s + v_t - \sum_{i'\in\text{nbr}(s), i'\neq i} m_{i'\to st}\right)\end{aligned}$$

Convergence We iteratively perform these updates in a loop until there is no change. The updates are guaranteed to converge to the optimal since the dual objective is lower bounded, convex and unconstrained.

Pseudocode of the Overall Algorithm These updates can be expressed as tensor operations for efficient implementation on GPUs. Following Bixler & Huang (2018), we derive the tensorized implementation of the dual updates. For $\mathbf{\Phi} \in \mathbb{R}^{n\times n}$ and $\mathbf{\Psi}_f, \mathbf{\Psi}_b \in \mathbb{R}^{n\times n\times|\hat{E}|}$ we define $\mathbf{\Phi}[i,j] = \phi_{j,i}$ for all $i, j \in [n]$, $\mathbf{\Psi}_f[j,t,(i,s)] = \phi_{is,jt}$, and $\mathbf{\Psi}_b[t,j,(s,i)] = \phi_{is,jt}$ for all $(i,s) \in \hat{E}$ and $j,t \in [n]$. We further define two auxiliary tensors $\boldsymbol{\tau}_f, \boldsymbol{\tau}_b \in \mathbb{R}^{|\hat{E}|\times n}$ as follows: $\boldsymbol{\tau}_f[(i,s),k] = 1$ if $k = s$ else 0 and $\boldsymbol{\tau}_b[(s,i),k] = 1$ if $k = i$ else 0 for all $(i,s) \in \hat{E}$ and $k \in [n]$. We store u_is in $\mathbf{u} \in \mathbb{R}^{1\times n}$, v_js in $\mathbf{v} \in \mathbb{R}^{n\times 1}$, the 'forward' (parent to child) messages in $\mathbf{m}_f \in \mathbb{R}^{n\times|\hat{E}|}$, and the 'backward' (child to parent) messages in $\mathbf{m}_b \in \mathbb{R}^{n\times|\hat{E}|}$. More specifically, $\mathbf{u}[1,i] = u_i$, $\mathbf{v}[j,1] = v_j$, $\mathbf{m}_f[t,(i,s)] = m_{i\to st}$, and $\mathbf{m}_b[j,(i,s)] = m_{s\to ij}$. Algorithm 1 shows the tensorized updates performed iteratively using these tensors. We assume broadcasting occurs implicitly between tensors of different sizes. $\text{logsumexp}_d(\cdot)$ denotes the logsumexp operation performed on the input tensor along dimension d.

Algorithm 1 Tensorized CTREEOT

Parameters: Regularization parameter: $\epsilon > 0$, maximum number of iterations: T, stopping threshold: δ
Inputs: $\mathbf{\Phi} \in \mathbb{R}^{n\times n}$, $\mathbf{\Psi}_f, \mathbf{\Psi}_b \in \mathbb{R}^{n\times n\times|\hat{E}|}$, $\boldsymbol{\tau}_f, \boldsymbol{\tau}_b \in \{0,1\}^{|\hat{E}|\times n}$
Variables: $\mathbf{u} \in \mathbb{R}^{1\times n}, \mathbf{v} \in \mathbb{R}^{n\times 1}, \mathbf{m}_f, \mathbf{m}_b \in \mathbb{R}^{n\times|\hat{E}|}$
Initialize: $\mathbf{u} \leftarrow \mathbf{0}, \mathbf{v} \leftarrow \mathbf{0}, \mathbf{m}_f \leftarrow \mathbf{0}, \mathbf{m}_b \leftarrow \mathbf{0}$
for $i = 1$ **to** T **do**
  Update u:
  $\mathbf{u}^{(i-1)} = \mathbf{u}$
  $\mathbf{u} \leftarrow \text{logsumexp}_1\left(\mathbf{m}_f\boldsymbol{\tau}_f + \mathbf{m}_b\boldsymbol{\tau}_b - \mathbf{\Phi}/\epsilon - \mathbf{v}\right)$
  Update v:
  $\mathbf{v} \leftarrow \text{logsumexp}_2\left(\mathbf{m}_f\boldsymbol{\tau}_f + \mathbf{m}_b\boldsymbol{\tau}_b - \mathbf{\Phi}/\epsilon - \mathbf{u}\right)$
  Update $\mathbf{m}_f$:
  $\mathbf{m}'_f \leftarrow (\mathbf{m}_f + (\mathbf{u}+\mathbf{v}+\mathbf{\Phi}/\epsilon - \mathbf{m}_f\boldsymbol{\tau}_f - \mathbf{m}_b\boldsymbol{\tau}_b)\boldsymbol{\tau}_f^t)/2$
  $\mathbf{m}_f \leftarrow \mathbf{m}'_f + \text{logsumexp}_1\left(-(\mathbf{m}_b + \mathbf{\Psi}_f/\epsilon)\right)/2$
  Update $\mathbf{m}_b$:
  $\mathbf{m}'_b \leftarrow (\mathbf{m}_b + (\mathbf{u}+\mathbf{v}+\mathbf{\Phi}/\epsilon - \mathbf{m}_f\boldsymbol{\tau}_f - \mathbf{m}_b\boldsymbol{\tau}_b)\boldsymbol{\tau}_b^t)/2$
  $\mathbf{m}_b \leftarrow \mathbf{m}'_b + \text{logsumexp}_1\left(-(\mathbf{m}_f + \mathbf{\Psi}_b/\epsilon)\right)/2$
  Check convergence:
  if $|\mathbf{u} - \mathbf{u}^{(i-1)}|_\infty < \delta$ **then**
    break
  end if
end for
$\mathbf{\Pi} \leftarrow \exp\left((\mathbf{m}_f\boldsymbol{\tau}_f + \mathbf{m}_b\boldsymbol{\tau}_b - \mathbf{\Phi}/\epsilon - \mathbf{u} - \mathbf{v})\right)$
Return $\mathbf{\Pi}^\top$

3.2. Relevance Weights

When designing a conditional tree matching score between $\hat{\mathbf{y}}$ and $\tilde{\mathbf{y}}$, with corresponding input texts $\mathbf{x}, \tilde{\mathbf{x}}$, it is crucial to ensure that the scores do not just reflect the similarity of $\hat{\mathbf{y}}$ and $\tilde{\mathbf{y}}$, but also the relevance of the case $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ to the current input $\mathbf{x}$. Also, a case tree may be partially relevant to a case. Instead of assigning relevance scores to whole trees based on similarity of the text strings, we design a more fine-grained notion of relevance by attaching relevance scores to subtrees rooted at each node of the case tree.

We refer to r_j as relevance scores representing the likelihood of appearance in the ground-truth $\mathbf{y}$ of the case subtree rooted in $\tilde{y}_j$. We design a module to assign relevance scores $r_j = \Pr(\tilde{y}_j \in \mathbf{y} | \mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}}, j)$. To estimate r_j, we use a transformer model $\text{TX}_\psi([\mathbf{x}; \tilde{\mathbf{x}}; \tilde{\mathbf{y}}])$ that jointly encodes the input utterance $\mathbf{x}$, the case-utterance $\tilde{\mathbf{y}}$ and the case-tree $\tilde{\mathbf{y}}$. For each node $\tilde{y}_j \in \tilde{\mathbf{y}}$ we use a sigmoid activation to output the relevance score $\Pr(\tilde{y}_j \in \mathbf{y} | \mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}}, j)$.

Training r_j: Given a training dataset D, for each example $(\mathbf{x}, \mathbf{y}) \in D$ and its case example $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$, we compute the gold membership $\delta(\tilde{y}_j \in \mathbf{y})$ for each subtree rooted at node $\tilde{y}_j$ w.r.t. the gold tree $\mathbf{y}$. This takes value 0 or 1 depending on whether $\tilde{y}_j$ is present anywhere in $\mathbf{y}$. Depending on the application, the membership test can be modified. We supervise the relevance scores by minimizing the cross-entropy between $\Pr(\cdot | \mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}}, j)$ and $\delta(\tilde{y}_j | \mathbf{y})$. The overall training objective to train the parameters ψ of the relevance transformer is:

$$\min_{\psi} \sum_{(\mathbf{x},\mathbf{y}) \in D} \sum_{(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \in \mathcal{C}_x} \text{Loss}(\delta(\tilde{y}_j \in \mathbf{y}), \Pr(\cdot | \mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}}, j)) \tag{7}$$

In Section 5.1 we present the architecture of a specific relevance transformer that we trained for an end application.

3.3. Pooling Scores

To get score of a candidate $\hat{\mathbf{y}}$ based on all the case examples $\mathcal{C}_x$, we aggregate scores over all the case examples using the soft max operator as per Equation 8 below:

$$p(\hat{\mathbf{y}} | \mathbf{x}, \mathcal{C}_x) = \log \sum_{(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \in \mathcal{C}(x)} \exp\left(\text{CTS}(\hat{\mathbf{y}} | \tilde{\mathbf{y}}, \mathbf{x}, \tilde{\mathbf{x}})\right) \tag{8}$$

Next, we normalize the scores over all possible candidates using softmax as per Equation 9 below.

$$p(\hat{\mathbf{y}} | \mathbf{x}, \mathcal{C}_x) = \frac{\exp s_\psi(\hat{\mathbf{y}} | \mathbf{x}, \mathcal{C}_x)}{\sum_{\hat{\mathbf{y}}' \in \mathcal{Y}_x} \exp s_\psi(\hat{\mathbf{y}}' | \mathbf{x}, \mathcal{C}_x)} \tag{9}$$

Finally, we assign a weight to the case scores relative to the model scores. We depend on the supervised relevance scores to assign these weights. First, for each case we determine the mean relevance $r(\mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ of a case example $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ w.r.t. an input utterance $\mathbf{x}$ is computed as: $r(\mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \frac{1}{n} \sum_j r(\tilde{y}_j | \mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})$. Note this relevance score is independent of the candidate trees. We then rescale the case scores as follows:

$$w(x) = \max_{(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \in \mathcal{C}(x)} r(\mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})$$

$$s_\psi(\hat{\mathbf{y}} | \mathbf{x}, \mathcal{C}_x) = \frac{w(x)}{(1 - w(x))} p(\hat{\mathbf{y}} | \mathbf{x}, \mathcal{C}_x)$$

Instead of directly using the the maximum over the $r(\mathbf{x}, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ values as the weight, we can also learn to reshape and translate it with a parameterized sigmoid function. However, we found the above relevance weights alone to be adequate in our experiments.

4. Related Work

Optimal Transport on Structured Data With the success of the Sinkhorn algorithm for solving the optimal transport problem of matching distributions over discrete elements (Cuturi, 2013), many follow up work attempt to extent the algorithm for matching structured objects. Titouan et al. (2019); Chen et al. (2020); Xu et al. (2019) propose an optimal transport algorithm for general graph structured data. They extend the OT objective with a quadratic term to regularize matching nodes pairs of one graph to another. However, their algorithm does not provide guarantees to converge to the global optima, in contrast to our extension with edge variables and linear constraints. We discuss this QP-based formulation in Appendix A.2. Lim et al. (2022) incorporate order constraints in optimal transport but their notion of ordering is different from ours arising out of topological ordering on matched trees.

Graph Matching There is extensive work (Li et al., 2019; Zhang et al., 2021; Roy et al., 2022) on matching graphs but this literature differs from our work in two ways. First, they do not consider the problem of matching graphs conditional on an input x. Hence their matches are not relevance aware. We show in the experiment section the key role of the relevance weightings in ranking correct trees higher even in the presence of irrelevant trees. Second, many of these either convert entire graph into a vector, or perform joint embedding of the matched graphs. We consider a middle ground where we match structural node-embeddings. Fei et al. (2022) uses matching of two trees to regularize the training of dual models. However, they independently match pairs of nodes across trees and do not maintain any structural constraints.

Inference-Time Adaptation Prior work on inference time adaptation of Text-to-Tree models largely relies on the standard sequence-to-sequence approach where the input-output

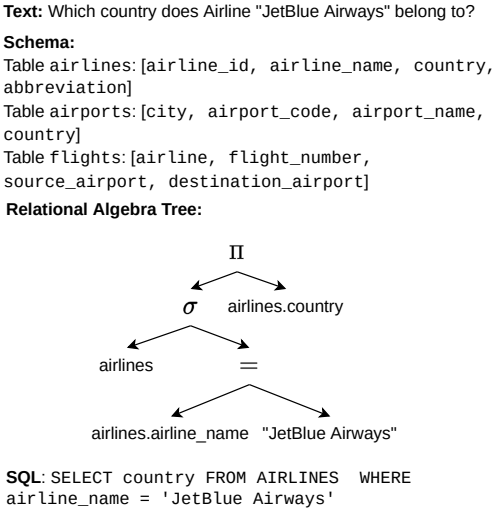


Figure 2. Example of trees arising in the Text-to-SQL task.

pairs from case examples are concatenated along with the test-input to generate the target output (Pasupat et al., 2021; Das et al., 2021; Gupta et al., 2021; Poesia et al., 2022). This approach often disregards the tree-structure of the example outputs by encoding the tree as a sequence. Awasthi et al. (2023) show that explicitly leveraging the similiarity between candidate and case subtrees leads to better decoding decisions. However, their method computes tree similarity simply as a dot product between two embedding vectors. Our method of aligning candidate and case subtrees and aggregating similarity over all the aligned nodes leads to consistently better results than Awasthi et al. (2023).

5. Experiments

We now present an empirical evaluation of CTREEOT both in terms of the quality of our proposed conditional tree matching score CTS and running time. We evaluate the quality of CTS by deploying it for inference-time adaptation of a real-life task of converting text utterances to SQL represented as an abstract relational tree. We describe the task next. The code for CTREEOT has been open-sourced[1].

5.1. Inference-Time Adaptation of Text-to-SQL Models

Text-to-SQL is a type of semantic parsing task that converts natural text question $\mathbf{x}$ on a specific database schema $\mathbf{s}$ to an SQL query $\mathbf{y}$ that can be executed on the database. An example is shown in Figure 2. The input to a Text-to-SQL model is the text concatenated with the schema $(\mathbf{x}, \mathbf{s})$ and output is a ranked list of SQL queries. Such models are trained with text-SQL pairs on a few database schema in the training data. However, they are often deployed on new schema unseen during training. Inference-time adaptation with few-shot labeled examples is particularly important for the Text-to-SQL task since several previous studies have shown the lack of cross-schema generalization (Suhr et al., 2020; Lee et al., 2021; Hazoom et al., 2021) of Text-to-SQL models. Fine-tuning the entire model is not an option because the same model is often shared by several users with different schema. This has led to much recent work on online adaptation of a trained Text-to-SQL model with a few labeled examples in a target schema. We will compare with two recent SOTA online adaptation methods, two baseline models, and CHATGPT as described next.

SMBOP (Rubin & Berant, 2021): As a base model M_θ for this task we use the SMBOP model that performs semi-autoregressive decoding to convert an input $\mathbf{x}, \mathbf{s}$ into an abstract SQL tree. The model generates the tree bottom-up layer by layer and at the last step returns a ranked list of SQL trees in its beam. We follow the authors' implementation of SMBOP[2] and initialize the text encoder with a ROBERTA-BASE model followed by 4 RAT layers (Wang et al., 2020) for encoding the schema structure.

STRUCTCBR: Awasthi et al. (2023) recently extended SMBOP for online adaptation with a few labeled examples. They rerank candidate trees at each layer based on similarity with case trees $\mathcal{C}_x$. This has been shown to lead to both increased recall at the final beam, and increased accuracy of top-1 tree. They compare similarity of two trees by collapsing the tree into a single vector. Unlike Awasthi et al. (2023), we train STRUCTCBR with 7 retrieved cases using a ROBERTA-BASE-based retriever described in Appendix A.

T5: Both the above methods directly generate trees from the model. We also compare our method with other popular models that treat the SQL as a string and generate the output as a string directly. As a base model for this approach, we utilize UnifiedSKG's (Xie et al., 2022) implementation[3] of a T5-LARGE model (Raffel et al., 2020)[4].

T5-CONCAT: A popular method of online adaptation of seq-to-seq models is by treating the cases as prompts that are concatenated with the input and jointly encoded by the encoder (Pasupat et al., 2021; Das et al., 2021; Gupta et al., 2021; Poesia et al., 2022). We use the implementation of (Awasthi et al., 2023) for this comparison.

CHATGPT: In addition to T5 and T5-CONCAT, we use CHATGPT (GPT-3.5-TURBO)[5] as another language model

[1] https://github.com/hrshtv/CTreeOT

[2] github.com/OhadRubin/SmBop

[3] github.com/HKUNLP/UnifiedSKG/blob/main/configure/Salesforce/T5_large_finetune_spider_with_cell_value.cfg

[4] github.com/google-research/text-to-text-transfer-transformer/blob/main/released_checkpoints.md#lm-adapted-t511lm100k

[5] platform.openai.com/docs/guides/chat (March 2023 version)

Table 1. EM values on five different schemas of varying difficulty and domains from the SPIDER dataset.

Schema	CHATGPT-ZS	CHATGPT-FS	T5	T5-CONCAT	SMBOP	STRUCTCBR	CTREEOT
car_1	23.9	32.8	40.6	42.2	43.3	48.9	**52.8**
cre_Doc_Template_Mgt	33.3	45.9	83.0	79.3	84.3	**89.3**	88.1
dog_kennels	40.1	49.7	59.9	**72.1**	66.7	67.4	70.1
flight_2	34.7	51.0	59.9	61.9	56.5	59.2	**61.9**
world_1	27.0	36.0	46.4	48.3	46.1	47.6	**49.1**
Micro-average	30.9	41.8	56.1	58.7	57.3	60.3	**62.2**

approach. We evaluate it in a zero-shot (CHATGPT-ZS) and few-shot setting (CHATGPT-FS). Similar to T5-CONCAT, we concatenate the few-shot case examples along with the input question in the prompt to CHATGPT. We show examples of prompts used in the zero-shot and few-shot setting in Appendix A.7.

CTREEOT: We build our method upon STRUCTCBR. We run STRUCTCBR for all steps except the final decoding step. At the final decoding step, we discard STRUCTCBR's beam scores and rescore the beam using our improved scoring mechanism discussed in § 3. We use $\epsilon = 10^{-3}$ and $\lambda = 1$. The beam is of size 30 and serves as our set of candidate trees $\mathcal{Y}_x$. This code also includes a retriever module that given an input $\mathbf{x}$ retrieves 7 related trees at test time to serve as the case set $\mathcal{C}_x$. We use the embeddings assigned by this model as the input contextual embeddings vector for each node of the tree. The only additional model we train is the Relevance transformer for assigning relevance scores r_j that we describe next.

Relevance Transformer: We train the relevance transformer independently, keeping the other modules frozen. Our relevance transformer is a relatively lightweight module consisting of approximately 3.3M parameters that is trained from scratch without any pretraining. Note that SMBOP with ROBERTA-BASE and four RAT layers has approximately 133M parameters. Our relevance transformer consists of four transformer blocks with a fully-connected layer at the end to predict the scores. A single block is a stack of self-attention (8 heads), feedforward, and layer normalization layers. We use the base model's embeddings of the input and case text and concatenate them with the uncontextualized node embeddings of the case tree given by STRUCTCBR. Our relevance transformer, takes this concatenated sequence as an input and predicts the relevance of all subtrees in the case tree. While training, we keep the batch size as a multiple of the number of cases, and design the batches such that for a candidate tree, the remaining $|\mathcal{C}| - 1$ examples are from the same schema and act as cases. We implement the matching using a hashing function that ignores the DB constants in the leaf nodes so that trees with identical structure but different DB constants result in the same hash value. We use a retriever and keep $|\mathcal{C}| = 8$. Our relevance transformer achieves an average F1 score of 77.1 on the validation split after being trained for 75 epochs. F1 score is macro-averaged over all examples and micro-averaged over all subtrees in all cases. We freeze all other modules while training the relevance transformer and supervise it via the cross-entropy loss, keeping other hyperparameters like the learning rate same as STRUCTCBR.

Datasets We adapt a Text-to-SQL model to five different target schemas from the SPIDER dataset (Yu et al., 2018) without finetuning. The target schemas were chosen have varying difficulties and domains. For training, we use SPIDER's train split containing 7000 Text-to-SQL examples from 140 schemas. For evaluation, we follow Awasthi et al. (2023) and use examples from the following five schemas from SPIDER's development set: {`world_1`, `car_1`, `cre_Doc_Template_Mgt`, `dog_kennels`, `flight_2`}. Examples from these schemas are excluded from the training and validation splits. The remaining 576 examples from the SPIDER's development set are used for validation. Like Awasthi et al. (2023), we divide a schema $\mathcal{D}$ randomly into $\mathcal{D}_{\text{test}}$ and $\mathcal{D}_{\text{cases}}$ such that $\mathcal{D}_{\text{test}} \cup \mathcal{D}_{\text{cases}} = \mathcal{D}$ and $\mathcal{D}_{\text{test}} \cap \mathcal{D}_{\text{cases}} = \emptyset$. The held out examples serve as the set of cases used for adapting models to that schema. We do this for each of the five schemas separately and evaluate on the $\mathcal{D}_{\text{test}}$ split. We report metrics averaged over three random $\mathcal{D}_{\text{test}}/\mathcal{D}_{\text{cases}}$ splits for each schema.

Evaluation Metrics Following Yu et al. (2018), we report the Exact-Set-Match Accuracy (EM). EM is 1 if all clauses in the gold and the predicted SQL match exactly (after anonymizing the DB values) and 0 otherwise.

5.2. Overall Results

Table 1 shows the EM values by different methods on five schemas of varying difficulty and domains from the SPIDER dataset. We make the following observations from this table. (1) Compared to the base SMBOP model, inference-time adaptation leads to accuracy increasing from 57.3% to 62.2%. On the most challenging schema (car_1) where the base model provides only 43.3% accuracy, we are able to boost by 9.5%. This shows the usefulness of inference-time adaptation of Text-to-SQL models to new schema just with 30 labeled examples. (2) Compared to the SOTA adaptation

Table 2. Ablation study showing the improvements obtained by different components of our proposed approach. CTREEOT–R denotes the CTREEOT algorithm without the relevance transformer while our method without the tree constraints reduces to Sinkhorn+relevance scoring. CTREEOT–A denotes our method with all alignments $\pi_{i,j}$ set to 1.

Schema	CTREEOT–A	CTREEOT–R	Sinkhorn+R	CTREEOT
car_1	46.7	50.0	52.8	52.8
cre_Doc_Templ.	78.6	84.9	88.1	88.1
dog_kennels	64.0	62.6	69.4	70.1
flight_2	55.8	60.5	61.9	61.9
world_1	40.1	43.5	48.7	49.1
Micro-average	54.7	58.0	62.0	62.2

Table 3. EM values on nine additional schemas when using the bottom 7 retrieved cases.

Schemas	SMBOP	STRUCTCBR	T5	T5-CONCAT	CTREEOT
concert_singer	71.1	80.0	86.7	35.6	80.0
employee_hire_evaluation	92.1	86.8	94.7	39.5	92.1
network_1	73.2	64.3	62.5	12.5	69.6
orchestra	85.0	82.5	90.0	42.5	82.5
pets_1	61.9	59.5	69.0	40.5	69.0
poker_player	97.5	97.5	92.5	35.0	95.0
student_transcripts_tracking	48.7	48.7	62.8	23.1	48.7
tvshow	91.9	82.3	71.0	29.0	88.7
wta_1	82.3	74.2	59.7	29.0	79.0
Micro-average	76.2	72.8	73.9	30.2	76.0

method STRUCTCBR we boost accuracy by almost 2 point in four out of five schema. This shows that matching whole trees as a single vector in inadequate, and our fine-trained tree matching helps. (3) T5-CONCAT follows a very different paradigm of online adaptation where the cases are treated as prompts that are concatenated to the input. We find that while they provide modest gains over their corresponding baseline (T5), overall their relative gains are much worse. This shows that memory-based methods that explicitly boost scores based on similarity with related cases is more effective than standard methods that encode cases with the inputs. We observe that the overall performance of CHATGPT is much worse than CTREEOT. Even though CHATGPT benefits from case examples, its overall accuracy (EM) is much lower. Previously, (Rajkumar et al., 2022) and (Poesia et al., 2022) have also observed poor Text-to-SQL performance with GPT-3 based models

5.3. Ablation Study

We next evaluate the impact of two main features of CTREEOT: the role of the tree constraints, and the role of the relevance transformer. Table 2 shows an ablation with each of the two features removed. Note, when we remove the tree constraints our method reduces to the well-known Sinkhorn algorithm (Cuturi, 2013) but with our relevance scoring. We observe that including the relevance transformer improves the EM by up to 7.5% and on average by 4.2%. We also observe that our proposed alignment algorithm that considers tree-based topological constraints provides gains over the topologically-unconstrained Sinkhorn algorithm in two schemas.

5.4. Effect of Case Relevance

The accuracy of T5-CONCAT is quite sensitive to the relevance of the input cases, and can get even worse than the baseline. To bring out this brittleness of concat-based approaches further, we conduct experiments by providing the bottom-7 cases (least relevant) as judged by the retrieval module. As can be seen in Table 3, we observe a big drop in accuracy of T5-CONCAT when presented with irrelevant cases. The evaluation setting here differs slightly from earlier experiments: for each example, we first randomly select 30 other examples from the same schema and then use the retriever on these to obtain the bottom-7 cases. A substantial drop is observed for STRUCTCBR too. In contrast, CTREEOT is more robust and does not suffer that much drop in accuracy in the presence of irrelevant cases, because of the presence of the relevance transformer.

5.5. Results on Synthetic Data

To measure the run-time on increasing tree size and study the number of topological constraints violated by CTREEOT and Sinkhorn, we performed more extensive experiments on

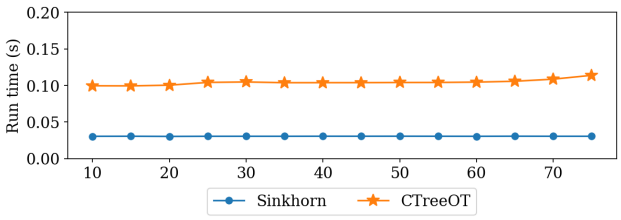


Figure 3. Variation in the run time (in seconds) of Sinkhorn and CTREEOT with respect to the number of nodes.

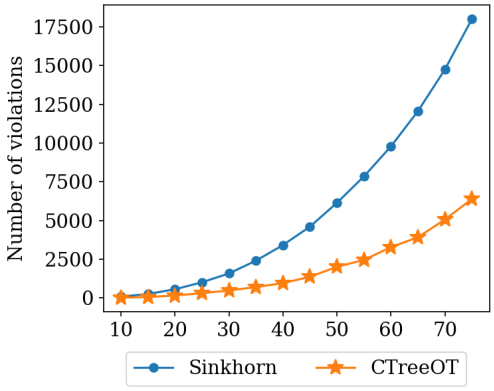


Figure 4. Variation of the number of topological violations made by Sinkhorn and CTREEOT with respect to the number of nodes.

synthetically generated trees. For each n, we generate 100 pairs of trees with n nodes each. We generate a tree level by level starting from the root node. When adding nodes at level $(l+1)$, we sample the number of nodes to be added at that level uniformly from $\{1, \ldots, 2n_l\}$, where n_l is the number of nodes added at level l. Now, for all new nodes, we uniformly sample its parent from the n_l nodes at level l. We continue to add new levels in the same way until the total number of nodes become n. To assign costs to a pair of synthetic trees, we first create a matrix $D \in [-1, 1]^{n \times n}$ with D_{ij} sampled uniformly from $[-1, 1]$. Then, we compute $C = (D + D^t)/2$ and use it as the cost. Recall that for an alignment π we want $\pi_{i,j}\pi_{s,t} = 0$ whenever $i \prec_{\hat{E}} s$ and $t \preceq_{\tilde{E}} j$. To decide whether such constraints are violated, we check whether $\pi_{i,j}\pi_{s,t} > \delta$. In our experiments, we set $\delta = 10^{-6}$ and test for all possible constraints. We vary n from 10 to 75, generate two trees and then compute the average running time and the average number of constraints violated for n when running our CTREEOT algorithm and the unconstrained Sinkhorn algorithm. These experiments were performed on a single NVIDIA RTX A6000 GPU and the algorithms were implemented in PyTorch. Figure 3 shows how the running times of CTREEOT and Sinkhorn vary with respect to the number of nodes in the input trees. Figure 4 shows the number of topological violations made

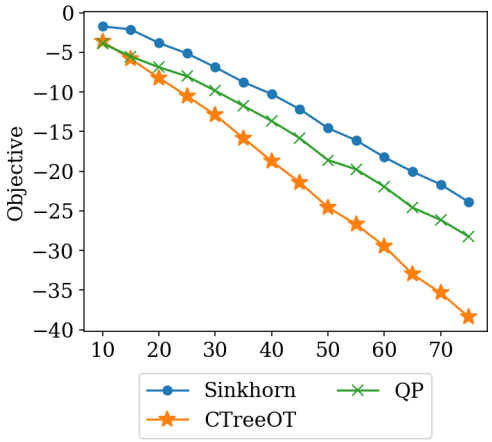


Figure 5. Variation of the objective value achieved by Sinkhorn, QP ($\lambda = 10$), and CTREEOT with respect to the number of nodes.

by Sinkhorn and CTREEOT as a function of the tree size n. We observe that while our method is slower than Sinkhorn it scales in roughly the same manner. Once all the tensors fit in GPU memory, the running time does not increase linearly. The increased running time is justified by the better quality alignments produced by CTREEOT as measured by the number of topological constraint violations. In Figure 5, we also show the objective values achieved by CTREEOT, Sinkhorn, and the QP-based formulation that is discussed in detail in Appendix A.2. We observe that in most cases the resultant CTREEOT objective is much lower than that achieved by the QP-based formulation and Sinkhorn.

6. Conclusion

We presented CTREEOT, a convergent, differentiable algorithm for matching trees conditioned on separate inputs. We applied CTREEOT for the task of cross-schema adaptation of Text-to-SQL models and showed how conditionally matching trees is useful for light-weight, few-shot adaptation of trained tree prediction models without parameter fine-tuning. We further demonstrated how the relevance-based reweighting makes the alignments useful for matching in prediction tasks, and also benefit from fine-grained partial overlap from related trees. Compared to the state-of-the-art method, CTREEOT achieves a significant increase in adaptation accuracy measured on schemas of varying difficulty and domains. In future, we plan to analyze theoretically the convergence rate of our alignment algorithm, and deploy it on other tasks.

Acknowledgements We gratefully acknowledge support from the IBM AI Horizon Networks-IIT Bombay initiative. Abhijeet thanks Google for the PhD Fellowship. We thank Soumen Chakrabarti for initial discussions.

References

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. Network flows - theory, algorithms and applications. 1993.

Atzeni, M., Dhuliawala, S. Z., Murugesan, K., and SACHAN, M. Case-based reasoning for better generalization in text-adventure games. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZDaSIkWT-AP.

Awasthi, A., Chakrabarti, S., and Sarawagi, S. Structured case-based reasoning for inference-time adaptation of text-to-sql parsers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Bixler, R. and Huang, B. Sparse-matrix belief propagation. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, March 2004.

Chen, L., Gan, Z., Cheng, Y., Li, L., Carin, L., and Liu, J. Graph optimal transport for cross-domain alignment. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 1542–1553. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20e.html.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

Das, R., Zaheer, M., Thai, D., Godbole, A., Perez, E., Lee, J. Y., Tan, L., Polymenakos, L., and McCallum, A. Case-based reasoning for natural language queries over knowledge bases. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9594–9611, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.755. URL https://aclanthology.org/2021.emnlp-main.755.

Fei, H., Wu, S., Ren, Y., and Zhang, M. Matching structure for dual learning. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 6373–6391. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/fei22a.html.

Gupta, V., Shrivastava, A., Sagar, A., Aghajanyan, A., and Savenkov, D. Retronlu: Retrieval augmented task-oriented semantic parsing. *arXiv preprint arXiv:2109.10410*, 2021.

Hazoom, M., Malik, V., and Bogin, B. Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pp. 77–87, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.nlp4prog-1.9. URL https://aclanthology.org/2021.nlp4prog-1.9.

Khandelwal, U., Levy, O., Jurafsky, D., Zettlemoyer, L., and Lewis, M. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HklBjCEKvH.

Khandelwal, U., Fan, A., Jurafsky, D., Zettlemoyer, L., and Lewis, M. Nearest neighbor machine translation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=7wCBOfJ8hJM.

Kitagawa, J., Mérigot, Q., and Thibert, B. Convergence of a newton algorithm for semi-discrete optimal transport. *Journal of the European Mathematical Society,*, 21(9):2603–2651, 2019.

Lee, C.-H., Polozov, O., and Richardson, M. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 2261–2273, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.176. URL https://aclanthology.org/2021.acl-long.176.

Li, Y., Gu, C., Dullien, T., Vinyals, O., and Kohli, P. Graph matching networks for learning the similarity of graph structured objects. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3835–3845. PMLR, 2019.

Lim, Y. C. F., Wynter, L., and Lim, S. H. Order constraints in optimal transport. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 13313–13333. PMLR, 17–23 Jul

2022. URL https://proceedings.mlr.press/v162/lim22b.html.

Pasupat, P., Zhang, Y., and Guu, K. Controllable semantic parsing via retrieval augmentation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7683–7698, 2021.

Pawlik, M. and Augsten, N. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)*, 40(1):1–40, 2015.

Pawlik, M. and Augsten, N. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173, 2016.

Peyré, G. and Cuturi, M. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019. ISSN 1935-8237. doi: 10.1561/2200000073. URL http://dx.doi.org/10.1561/2200000073.

Poesia, G., Polozov, A., Le, V., Tiwari, A., Soares, G., Meek, C., and Gulwani, S. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=KmtVD97J43e.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Rajkumar, N., Li, R., and Bahdanau, D. Evaluating the text-to-sql capabilities of large language models. *ArXiv*, abs/2204.00498, 2022.

Roy, I., Velugoti, V. S., Chakrabarti, S., and De, A. Interpretable neural subgraph matching for graph retrieval. In *AAAI Conference on Artificial Intelligence*, 2022.

Rubin, O. and Berant, J. Smbop: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 311–324, 2021.

Suhr, A., Chang, M.-W., Shaw, P., and Lee, K. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8372–8388, 2020.

Titouan, V., Courty, N., Tavenard, R., Laetitia, C., and Flamary, R. Optimal transport for structured data with application on graphs. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6275–6284. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/titouan19a.html.

Wang, B., Shin, R., Liu, X., Polozov, O., and Richardson, M. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7567–7578, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.677. URL https://aclanthology.org/2020.acl-main.677.

Xie, T., Wu, C. H., Shi, P., Zhong, R., Scholak, T., Yasunaga, M., Wu, C.-S., Zhong, M., Yin, P., Wang, S. I., et al. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*, 2022.

Xu, H., Luo, D., Zha, H., and Duke, L. C. Gromov-Wasserstein learning for graph matching and node embedding. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6932–6941. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/xu19b.html.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1425. URL https://aclanthology.org/D18-1425.

Zhang, Z., Bu, J., Ester, M., Li, Z., Yao, C., Yu, Z., and Wang, C. H2mn: Graph similarity learning with hierarchical hypergraph matching networks. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021.

A. Appendix

A.1. Case Retriever

We follow Awasthi et al. (2023) for the implementation of a RoBERTa-base-based case retriever $R_\phi(\mathbf{x}, L)$. Given an input text $\mathbf{x}$ and a set of cases $L = \{(\tilde{\mathbf{x}}_l, \tilde{\mathbf{y}}_l)\}_{l=1}^{|L|}$, the retriever assigns scores to the cases based on the cosine similarity between the embeddings of $\mathbf{x}$ and $\tilde{\mathbf{x}}$. During inference, we retrieve the top-C cases based on these scores. The retriever is trained independently of the Text-to-SQL models and is supervised using the normalized tree-edit-distance (TED) between the $\mathbf{y}$ and $\tilde{\mathbf{y}}$ that is computed using the APTED library (Pawlik & Augsten, 2015; 2016)[6]. The normalized tree-edit-distances lie in $[0, 1]$ and are independent of the leaf constants and values. Like Awasthi et al. (2023), for a candidate $(\mathbf{x}_i, \mathbf{y}_i)$, we sample 15 pairs $\{(\tilde{\mathbf{x}}_j, \tilde{\mathbf{y}}_j)\}_{j=1}^{15}$ and then compute the cosine similarities $\{\mathrm{CSim}_\phi(\mathbf{x}_i, \tilde{\mathbf{x}}_j)\}_{j=1}^{15}$ and the tree-edit-distances $\{\mathrm{TED}(\mathbf{y}_i, \tilde{\mathbf{y}}_j)\}_{j=1}^{15}$. We supervise these scores as per the below loss.

$$
\begin{aligned}
w_{i,j} &= \frac{\exp(1 - 2\,\mathrm{TED}(\mathbf{y}_i, \tilde{\mathbf{y}}_j))}{\sum_{j'} \exp(1 - 2\,\mathrm{TED}(\mathbf{y}_i, \tilde{\mathbf{y}}_{j'}))} \\
\mathcal{L} &= -\sum_{i,j} w_{i,j} \log \frac{\exp(\mathrm{CSim}_\phi(\mathbf{x}_i, \tilde{\mathbf{x}}_j))}{\sum_{j'} \exp(\mathrm{CSim}_\phi(\mathbf{x}_i, \tilde{\mathbf{x}}_{j'}))}
\end{aligned}
\tag{10}
$$

A.2. Comparing with the Quadratic Formulation

We can directly relax the formulation in Equation 3 with the entropic regularizer and obtain the below formulation:

$$
\begin{aligned}
\min_{\pi} \quad & \frac{1}{\epsilon} \sum_{i,j} \phi_{i,j} \pi_{i,j} + \frac{1}{\epsilon} \sum_{(i,s) \in \hat{E}} \sum_{j,t} \phi_{is,jt} \pi_{i,j} \pi_{s,t} + \sum_{ij} \pi_{i,j} (\log \pi_{i,j} - 1) \\
\text{s.t.} \quad & \sum_{i=1}^{n} \pi_{i,j} = 1 \,; \sum_{j=1}^{n} \pi_{i,j} = 1
\end{aligned}
\tag{11}
$$

We can solve the above QP using algorithms proposed in earlier work (Chen et al., 2020) for the Gromov-Wasserstein Distance between graphs. For each time step k, the algorithms proposed for these iteratively compute a cost matrix $\phi_{i,j}^k$ as follows.

$$
\phi_{i,j}^k = \phi_{i,j} + \sum_{(i,s) \in \hat{E}} \sum_{t} \phi_{is,jt} \pi_{s,t}^{k-1}
$$

They then invoke the well-known Sinkhorn algorithm to obtain the updated alignments π^k with the above cost matrix.

Note that this objective is not convex. We deliberately remove the quadratic term and introduce additional edge variables to get a convex objective in Equation 4. This algorithm is therefore not guaranteed to converge to a global minima unlike ours. This approach repeatedly calls the Sinkhorn routine whereas we alternate between tree message passing and Sinkhorn. We can observe the difference empirically too. In the plot in Figure 6(a), we show the value of the objective on synthetic trees of Section 5.5 ($\lambda = 1$). We observe that in most cases the resultant CTreeOT objective is much lower than that achieved by the QP-based formulation. In fact, in most cases, the QP formulation provides the same solution as the baseline Sinkhorn without any edge cost. Only for large values of edge cost ($\lambda = 10$) does it do better as can be seen in Figure 6(b).

Another difference in our work is the application. We perform conditional matching of two output trees (y, y') given two different inputs (x, x'). In contrast, Chen et al. (2020) aligns x with y. In our work, in addition to aligning, we score aligned nodes based on the similarity of their respective input contexts that are implicitly aligned by the relevance transformer. Thus, in some sense, we handle a four-way match where the (x, x', y') alignment is handled implicitly by the relevance transformer. To the best of our knowledge, all prior work on graph optimal transport has reasoned about only two-way matches.

[6] https://pypi.org/project/apted/

We further evaluate our method on Barabási-Albert (BA) graphs as done in (Xu et al., 2019). Even though we did not consider general graphs when designing CTREEOT, we observe that our formulation provides better local optimums than the QP-based formulation, as can be seen in Figure 7.

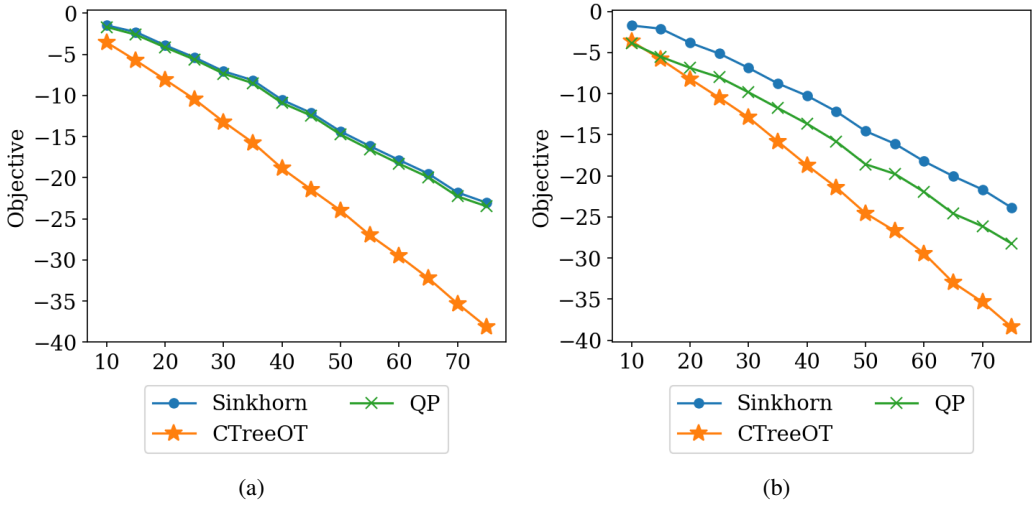


Figure 6. Variation of the objective value achieved by Sinkhorn, QP ($\lambda = 1$ and 10 respectively), and CTREEOT with respect to the number of nodes in tree.

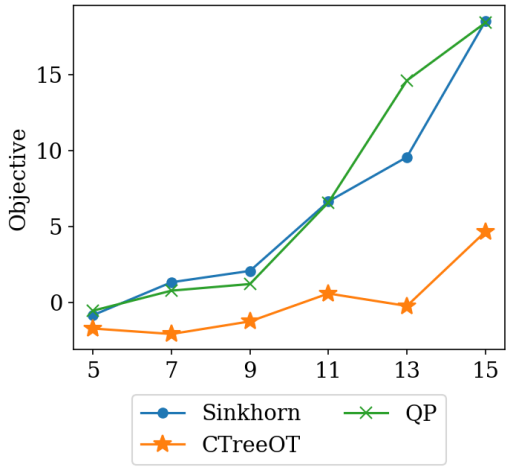
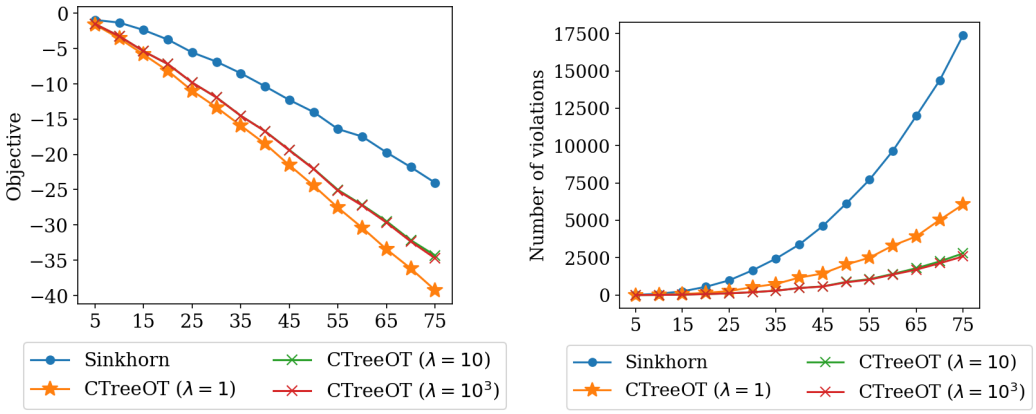


Figure 7. Variation of the objective value achieved by Sinkhorn, QP, and CTREEOT with respect to the number of nodes in the graph.

A.3. Effect of λ

Figure 8 shows how the objective value and the number of constraint violations changes with different edge costs. Note that the objective value is being computed with $\lambda = 1$ for all methods. Increasing λ leads to fewer constraint violations but a worse objective value.



(a) (b)

Figure 8. Variation of the objective value and the number of constraint violations with different edge costs.

A.4. Additional Experiments

Experiments on Additional Schemas The performance on various schemas can vary a lot depending on the difficulty of the schema and the queries. We perform experiments on nine additional schema comparing CTREEOT with the baselines.

Table 4. EM values on nine additional schemas.

Schemas	SMBOP	STRUCTCBR	T5	T5-CONCAT	CTREEOT
concert_singer	71.1	84.4	86.7	73.3	82.2
employee_hire_evaluation	92.1	92.1	94.7	86.8	97.4
network_1	73.2	75	62.5	71.4	73.2
orchestra	85	97.5	90	95	97.5
pets_1	61.9	61.9	69	52.4	73.8
poker_player	97.5	100	92.5	95	97.5
student_transcripts_tracking	48.7	51.3	62.8	57.7	52.6
tvshow	91.9	91.9	71	75.8	93.5
wta_1	82.3	80.6	59.7	72.6	82.3
Micro-average	76.2	79.3	73.9	73.7	80.8

A.5. Sensitivity of CTREEOT to hyperparameters

We report accuracy (EM) values for top-3 retrieved cases for $\epsilon \in \{1, 10^{-1}, 10^{-3}, 10^{-5}\}$ and $\lambda \in \{1, 10^6\}$ in Table 5. We observe that CTREEOT is reasonably robust to these hyperparameters for this task.

Table 5. EM values for top-3 retrieved cases for different values of ϵ and λ

$\epsilon \downarrow \lambda \rightarrow$	$\mathbf{1}$	$\mathbf{10^6}$
$\mathbf{1}$	79.7	79.0
$\mathbf{10^{-3}}$	85.8	87.8
$\mathbf{10^{-5}}$	87.8	87.8
$\mathbf{10^{-7}}$	87.5	87.5

A.6. The Sinkhorn Algorithm

The conventional optimal transport problem without any pairwise constraints can be written as per Equation 12. Solving this in the same way by computing the Dual gives us the well-known iterative Algorithm 2 (Cuturi, 2013).

$$\begin{aligned} \min_{\pi} \quad & \sum_{i,j} \phi_{i,j}\pi_{i,j} + \epsilon \sum_{i,j} \pi_{i,j} \log \pi_{i,j} \\ \text{s.t.} \quad & \sum_{i=1}^{n} \pi_{i,j} = 1 \; \forall \, j \, ; \sum_{j=1}^{n} \pi_{i,j} = 1 \; \forall \, i \, ; \pi_{i,j} \geq 0 \; \forall \, i, j \end{aligned} \tag{12}$$

Algorithm 2 Tensorized Sinkhorn

Parameters: Regularization parameter: $\epsilon > 0$,
maximum number of iterations: T, stopping threshold: δ
Inputs: $\mathbf{\Phi} \in \mathbb{R}^{n \times n}$
Variables: $\mathbf{u} \in \mathbb{R}^{n \times 1}, \mathbf{v} \in \mathbb{R}^{1 \times n}$
Initialize: $\mathbf{u} \leftarrow \mathbf{0}, \mathbf{v} \leftarrow \mathbf{0}$
for $i = 1$ **to** T **do**
  Update u:
  $\mathbf{u}^{(i-1)} = \mathbf{u}$
  $\mathbf{u} \leftarrow \epsilon \operatorname{logsumexp}_2 \left(-\mathbf{\Phi}/\epsilon - \mathbf{v}/\epsilon\right) - \epsilon$
  Update v:
  $\mathbf{v} \leftarrow \epsilon \operatorname{logsumexp}_1 \left(-\mathbf{\Phi}/\epsilon - \mathbf{u}/\epsilon\right) - \epsilon$
  Check convergence:
  if $|\mathbf{u} - \mathbf{u}^{(i-1)}|_\infty < \delta$ **then**
    break
  end if
end for
$\mathbf{\Pi} \leftarrow \exp\left(\left(-\mathbf{\Phi} - \mathbf{u} - \mathbf{v}\right)/\epsilon - 1\right)$
Return $\mathbf{\Pi}$

A.7. Examples of Prompts used for CHATGPT evaluation

Zero-Shot Prompt:

```
Suppose a database has tables: [`continents`, `countries`, `car_makers`, `model_list`, `
    car_names`, `cars_data`].
Table `continents` has columns: [`ContId`, `Continent`] with primary key `ContId`.
Table `countries` has columns: [`CountryId`, `CountryName`, `Continent`] with primary key
    `CountryId`.
Table `car_makers` has columns: [`Id`, `Maker`, `FullName`, `Country`] with primary key `
    Id`.
Table `model_list` has columns: [`ModelId`, `Maker`, `Model`] with primary key `ModelId`.
Table `car_names` has columns: [`MakeId`, `Model`, `Make`] with primary key `MakeId`.
Table `cars_data` has columns: [`Id`, `MPG`, `Cylinders`, `Edispl`, `Horsepower`, `Weight
    `, `Accelerate`, `Year`] with primary key `Id`.
Foreign key from countries.Continent to continents.ContId.
Foreign key from car_makers.Country to countries.CountryId.
Foreign key from model_list.Maker to car_makers.Id.
Foreign key from car_names.Model to model_list.Model.
Foreign key from cars_data.Id to car_names.MakeId.
Convert the following English question into SQLite queries on this database: 'How many car
     models are produced by each maker ? Only list the count and the maker full name .'.
Return the best/top 8 SQLs as a Markdown ordered/numbered list, ranked from 1 to 8. Do not
     provide any explanations or details or any additional text, just provide the list of
    SQL queries.
```

Few-Shot Prompt:

```
Suppose a database has tables: [`continents`, `countries`, `car_makers`, `model_list`, `
    car_names`, `cars_data`].
Table `continents` has columns: [`ContId`, `Continent`] with primary key `ContId`.
Table `countries` has columns: [`CountryId`, `CountryName`, `Continent`] with primary key
    `CountryId`.
Table `car_makers` has columns: [`Id`, `Maker`, `FullName`, `Country`] with primary key `
    Id`.
Table `model_list` has columns: [`ModelId`, `Maker`, `Model`] with primary key `ModelId`.
Table `car_names` has columns: [`MakeId`, `Model`, `Make`] with primary key `MakeId`.
Table `cars_data` has columns: [`Id`, `MPG`, `Cylinders`, `Edispl`, `Horsepower`, `Weight
    `, `Accelerate`, `Year`] with primary key `Id`.
Foreign key from countries.Continent to continents.ContId.
Foreign key from car_makers.Country to countries.CountryId.
Foreign key from model_list.Maker to car_makers.Id.
Foreign key from car_names.Model to model_list.Model.
Foreign key from cars_data.Id to car_names.MakeId.
Convert the following English question into SQLite queries on this database: 'How many car
     models are produced by each maker ? Only list the count and the maker full name .'.
Return the best/top 8 SQLs as a Markdown ordered/numbered list, ranked from 1 to 8. Do not
     provide any explanations or details or any additional text, just provide the list of
    SQL queries.

Examples of converting an English question into a single SQL query:

Input: What is the maximum accelerate for different number of cylinders?
Output: SELECT max(Accelerate) ,  Cylinders FROM CARS_DATA GROUP BY Cylinders;

Input: For a volvo model, how many cylinders does the version with least accelerate have?
Output: SELECT T1.cylinders FROM CARS_DATA AS T1 JOIN CAR_NAMES AS T2 ON T1.Id  =  T2.
    MakeId WHERE T2.Model  =  'volvo' ORDER BY T1.accelerate ASC LIMIT 1;

Input: What are the names and ids of all countries with at least one car maker?
Output: SELECT T1.CountryName ,  T1.CountryId FROM COUNTRIES AS T1 JOIN CAR_MAKERS AS T2
    ON T1.CountryId  =  T2.Country GROUP BY T1.CountryId HAVING count(*)  >=  1;
```