A Differentiable Relaxation of Graph Segmentation and Alignment for AMR Parsing

Chunchuan Lyu[1] **Shay B. Cohen**[1] **Ivan Titov**[1,2]
[1] ILCC, School of Informatics, University of Edinburgh
[2] ILLC, University of Amsterdam
chunchuan.lv@gmail.com scohen@inf.ed.ac.uk ititov@inf.ed.ac.uk

Abstract

Abstract Meaning Representations (AMR) are a broad-coverage semantic formalism which represents sentence meaning as a directed acyclic graph. To train most AMR parsers, one needs to segment the graph into subgraphs and align each such subgraph to a word in a sentence; this is normally done at preprocessing, relying on hand-crafted rules. In contrast, we treat both alignment and segmentation as latent variables in our model and induce them as part of end-to-end training. As marginalizing over the structured latent variables is infeasible, we use the variational autoencoding framework. To ensure end-to-end differentiable optimization, we introduce a differentiable relaxation of the segmentation and alignment problems. We observe that inducing segmentation yields substantial gains over using a 'greedy' segmentation heuristic. The performance of our method also approaches that of a model that relies on the segmentation rules of Lyu and Titov (2018), which were hand-crafted to handle individual AMR constructions.

1 Introduction

Abstract Meaning Representation (AMR; Banarescu et al. 2013) is a broad-coverage semantic formalism which represents sentence meaning as rooted labeled directed acyclic graphs. The representations have been exploited in a wide range of tasks, including text summarization (Liu et al., 2015; Dohare and Karnick, 2017; Hardy and Vlachos, 2018), machine translation (Jones et al., 2012; Song et al., 2019), paraphrase detection (Issa et al., 2018) and question answering (Mitra and Baral, 2016).

An AMR graph can be regarded as consisting of multiple concept subgraphs, which can be individually aligned to sentence tokens (Flanigan et al., 2014). In Figure 1, each dashed box represents the boundary of a single semantic subgraph. Red arrows represent the alignment between subgraphs and tokens. For example, '(o / opine-01: ARG1 (t / thing))' refers to a combination of the predicate 'opine-01' and a filler of its semantic role ARG1. Intuitively, this subgraph needs to be aligned to the token 'opinion'. Similarly, '(b / boy)' should be aligned to the token 'boy'. Given such an alignment and segmentation, it is straightforward to construct a simple parser: parsing can be framed as tagging input tokens with subgraphs (including empty subgraphs), followed by predicting relations between the subgraphs. The key obstacle to training such an AMR parser is that the segmentation and alignment between AMR subgraphs and words are latent, i.e. not annotated in the data.

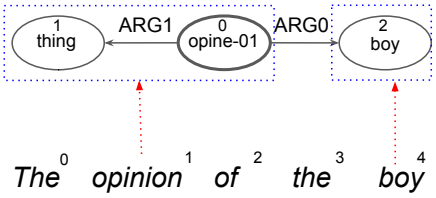

Figure 1: An example of AMR, the dashed red arrows mark latent alignment. Dashed blue boxes represent the latent segmentation.

Most previous work adopts a pipeline approach to handling this obstacle. They rely on a pre-learned aligner (e.g., (Pourdamghani et al., 2014)) to produce the alignment, and apply a rule system to segment the AMR subgraph (Flanigan et al., 2014; Werling et al., 2015; Damonte et al., 2017; Ballesteros and Al-Onaizan, 2017; Peng et al., 2015; Artzi et al., 2015; Groschwitz et al., 2018). While Lyu and Titov (2018) jointly optimize the parser and the alignment model, the rules handling specific constructions still needed to be crafted to segment the graph. The segmentation rules are relatively complex — e.g., the rules of Lyu and Titov (2018) targeted 40 different AMR subgraph types — and language-dependent. AMR has never been intended to be used as an interlingua (Banarescu et al.,

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 9075–9091
November 7–11, 2021. ©2021 Association for Computational Linguistics

2013; Damonte and Cohen, 2018) and AMR banks for individual languages substantially diverge from English AMR. For example, Spanish AMR represents pronouns and ellipsis differently from the English one (Migueles-Abraira et al., 2018). As new AMR sembanks in languages other than English are being developed (Anchiêta and Pardo, 2018; Song et al., 2020), domain-specific AMR extensions get developed (Bonn et al., 2020; Bonial et al., 2020), and extra constructions are getting introduced to AMRs (Bonial et al., 2018), eliminating the need for rules while learning graph segmentation from scratch is becoming an important problem to solve.

We propose to optimize a graph-based parser that treats the alignment and graph segmentation as latent variables. The graph-based parser consists of two parts: concept identification and relation identification. The concept identification model generates the AMR nodes, and the relation identification component decides on the labeled edges. During training, both components rely on latent alignment and segmentation, which are being induced simultaneously. Importantly, at test time, the parser simply tags the input with the subgraphs and predicts the relations, so there is no test-time overhead from using the latent-structure apparatus. An extra benefit of this approach, in contrast to encoder-decoder AMR models (Konstas et al., 2017; van Noord and Bos, 2017; Cai and Lam, 2020) is its transparency, as one can readily see which input token triggers each subgraph.[1]

To develop our parser, we frame the alignment and segmentation problems as choosing a *generation order* of concept nodes, as we explain in Section 2.2. As marginalization over the latent generation orders is infeasible, we adopt the variational auto-encoder (VAE) framework (Kingma and Welling, 2014). Intuitively, a trainable neural module (an encoder in the VAE) is used to sample a plausible generation order (i.e., a segmentation plus an alignment), which is then used to train the parser (a decoder in the VAE). As one cannot 'differentiate through' a sample of discrete variables to train the encoder, we introduce a differentiable relaxation which makes our objective end-to-end differentiable.

We experiment on the AMR 2.0 and 3.0 datasets. We compare to a *greedy segmentation* heuristic, inspired by Naseem et al. (2019), that produces a segmentation deterministically and provides a strong baseline to our segmentation induction method. We also use a version of our model with segmentation induction replaced by a hand-crafted rule-based segmentation system from previous work;[2] it can be thought of as an upper bound on how well induction can work. On AMR 2.0 (LDC2016E25), we found that our VAE system obtained a competitive Smatch score of 76.1, reducing the gap between using the segmentation heristic (75.2) and the rules exploiting the prior knowledge about AMR (76.8). On AMR 3.0 (LDC2020T02), the VAE system gets even closer to the rule-based system (75.5 vs 75.7), possibly because the rules were designed for AMR 2.0. Our main contributions are:

- we frame the alignment and segmentation problems as inducing a generation order, and provide a continuous relaxation to this discrete optimization problem;
- we empirically show that our method outperforms a strong heuristic baseline and approaches the performance of a complex hand-crafted rule system.

Our method makes very few assumptions about the nature of the graphs, so it may be effective in other tasks that can be framed as graph prediction (e.g., executable semantic parsing, Liang 2016, or scene graph prediction, Xu et al. 2017).

2 Casting Alignment and Segmentation as Choosing a Generation Order

2.1 Preliminaries

We start by introducing the basic concepts and notation. We refer to words in a sentence as $\mathbf{x} = (x_0, \ldots, x_{n-1})$, where n is the sentence length. The concepts (i.e. labeled nodes) are $\mathbf{v} = (v_0, v_1, \ldots, v_m)$, where m is the number of concepts. In particular, $v_m = \emptyset$ denotes a dummy terminal node; its purpose will be clear in Section 2.2 where we will define the generative model. We refer to all nodes, except for the terminal node ($\emptyset$), as concept nodes.

A relation between 'predicate concept' i and 'argument concept' j is denoted by $\mathbf{E}_{ij}$. It is set to $\emptyset$ if j is not an argument of i. We will use $\mathbf{E}$ to denote all edges (i.e. relations) in the graph.

[1] The code is available at https://github.com/ChunchuanLv/graph-parser.

[2] We adopt the code from Zhang et al. (2019a), which is an extension of the system introduced by Lyu and Titov (2018), and also used by Cai and Lam (2020).

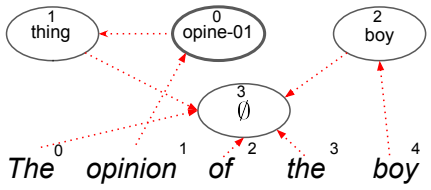

Figure 2: AMR concept identification model generates nodes following latent generation order at training time.

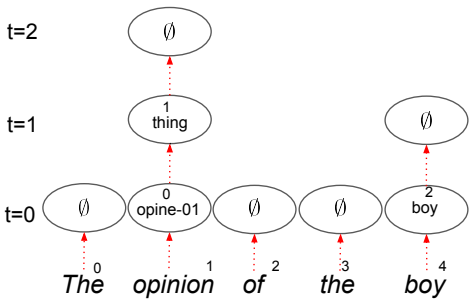

Figure 3: At test time, the AMR concept identification model generates nodes autoregressively, starting from each sentence token. Importantly, it is just an 'unrolled' form of the order shown in Figure 2.

In addition, we refer to the whole AMR graph as $\mathbf{G} = (\mathbf{v}, \mathbf{E})$.

Our goal is to associate each input token with a (potentially empty) subset of the concept nodes in the AMR graph, while making sure that we get a *partition* of the node set. In other words, each node in the original AMR graph belongs to exactly one subset. In that way, we deal with both segmentation and alignment. Each subset uniquely corresponds to a *vertex-induced subgraph* (i.e., the subset of nodes together with any edges whose both endpoints are in this subset). For this reason, we will refer to the problem as graph decomposition[3] and to each subset as a subgraph. We will explain how we deal with edges of the AMR graph in Section 3.2.

2.2 Generation Order

We choose a subset of nodes for each token by assigning an order to which the nodes are selected for such subset. In Figure 2, dashed red arrows point from every node to the subsequent node to be selected. For example, given the word 'opinion', the node 'opine-01' is chosen first, and then it is followed by another node 'thing'. After this node, we have an arrow pointing to the node $\emptyset$, signifying that we finished generating nodes aligned to the word 'opinion'. We refer to these red arrows as a *generation order*.

A generation order determines a graph decomposition. To recover it from a generation order, we assign connected nodes (excluding the terminal node) to the same subgraph. Then, a subgraph will be aligned to the token that generated those nodes. In our example, 'opine-01' and 'thing' are connected, and, thus, they are both aligned to the word 'opinion'. The alignment is encoded by arrows between tokens and concept nodes, while the segmentation is represented by arrows between concept nodes.

From a modeling perspective, the nodes will be generated with an autoregressive model, which is easy to use at test time (Figure 3). From each token, a chain of nodes is generated until the stop symbol $\emptyset$ is predicted. It is more challenging to see how to induce the order and train the autoregressive model at the same time; we will discuss this in Sections 3 and 4.

Constraints While in Figure 2 the red arrows determine a valid generation order, in general, the arrows have to obey certain constraints. Formally, we denote alignment by $\mathbf{A} \in \{0,1\}^{n \times (m+1)}$, where $\mathbf{A}_{ki} = 1$ means that for token k we start by generating node i. As the token can only point to one node, we have a constraint $\sum_i \mathbf{A}_{ki} = 1$. Similarly, for a segmentation $\mathbf{S} \in \{0,1\}^{m \times (m+1)}$ we have a constraint $\sum_j \mathbf{S}_{ij} = 1$. Setting $\mathbf{S}_{ij} = 1$ indicates that node i is followed by node j. In Figure 2, we have $\mathbf{A}_{03} = \mathbf{A}_{10} = \mathbf{A}_{23} = \mathbf{A}_{33} = \mathbf{A}_{42} = 1$ and $\mathbf{S}_{01} = \mathbf{S}_{13} = \mathbf{S}_{23} = 1$; the rest is 0. Now, we have the full generation order as their concatenation $\mathbf{O} = [\mathbf{A}; \mathbf{S}] \in \{0,1\}^{(n+m) \times (m+1)}$. As one node can only be generated once (except for $\emptyset$), we have a joint constraint: $\forall j \neq m, \sum_l \mathbf{O}_{lj} = 1$. Furthermore, the graph defined by $\mathbf{O}$ should be acyclic, as it represents the generative process. We denote the set of all valid generation orders as $\mathcal{O}$. In the following sections, we will discuss how this generation order is used in the model and how to infer it as a latent variable while enforcing the above constraints.

3 Our Model

Formally, we aim at estimating $P_\theta(\mathbf{v}, \mathbf{E}|\mathbf{x})$, the likelihood of an AMR graph given the sentence. Our graph-based parser is composed of two parts: concept identification $P_\theta(\mathbf{v}|\mathbf{x}, \mathbf{O})$ and relation identification $P_\theta(\mathbf{E}|\mathbf{x}, \mathbf{O}, \mathbf{v})$. The concept iden-

[3] We slightly abuse the terminology as, in graph theory, graph decomposition usually refers to a partition of edges – rather than nodes – of the original graph.

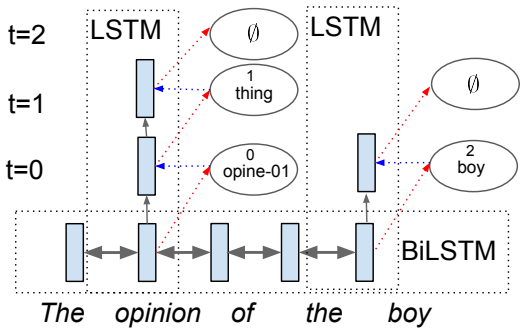

Figure 4: AMR concept identification model runs several independent LSTMs to generate nodes autoregressively at test time.

tification model generates concept nodes, and the relation identification model assigns relations between them. Both require the latent generation order at the training time, denoted by $\mathbf{O}$. Overall, we have the following objective:

$$\log P_\theta(\mathbf{v}, \mathbf{E}|\mathbf{x}) \quad (1)$$

$$= \log \sum_{\mathbf{O}} P_\theta(\mathbf{O}) P_\theta(\mathbf{v}|\mathbf{x}, \mathbf{O}) P_\theta(\mathbf{E}|\mathbf{x}, \mathbf{O}, \mathbf{v}) \ , \quad (2)$$

where $P_\theta(\mathbf{O})$ is a prior on the generation orders, discussed in Section 4.2. To efficiently optimize this objective end-to-end, as will be discussed in Section 4, we need to ensure that both concept and relation identification models admit relaxation, i.e., they should be well-defined for real-valued $\mathbf{O}$.

In the following subsections, we go through concept identification, relation identification, and their corresponding relaxations.

3.1 Concept Identification

As shown in Figure 4, our neural model first encodes the sentence with BiLSTM, producing token representations $\mathbf{h}_k^{\text{token}}$ ($k \in [0, \dots n-1]$), then generates nodes autoregressively at each token with another LSTM.

In training, we need to be able to run the models with any potential generation order and compute $P_\theta(\mathbf{v}|\mathbf{x}, \mathbf{O})$. If we take the order defined in Figure 2, the node 1 ('thing') is predicted relying on the corresponding hidden representation; we refer to this representation as $\mathbf{h}_1^{\text{node}}$ where is 1 is the node index. With the discrete generation order defined by red arrows in Figure 2, $\mathbf{h}_1^{\text{node}}$ is just the LSTM state of its parent (i.e. 'opine-01'). However, to admit relaxations, our computation should be well-defined when the generation order $\mathbf{O}$ is soft (i.e. attention-like). In that case, $\mathbf{h}_1^{\text{node}}$ will be a weighted sum of LSTM representations of other nodes and input tokens, where the weights are defined by $\mathbf{O}$. Similarly, the termination symbol $\emptyset$ for the token 'opinion' is predicted from its hidden representation; we refer to this representation as $\mathbf{h}_1^{\text{tail}}$, where 1 is the position of 'opine' in the sentence. With the hard generation order of Figure 2, $\mathbf{h}_1^{\text{tail}}$ is just the LSTM state computed after choosing the preceding node (i.e. 'thing'). In the relaxed case, it will again be a weighted sum with the weights defined by $\mathbf{O}$.

Formally, the probability of concept identification step can be decomposed into probability of generating m concepts nodes and n terminal nodes (one for each token):

$$P_\theta(\mathbf{v}|\mathbf{x},\mathbf{O}) = \prod_{i=0}^{m-1} P_\theta(\mathbf{v}_i|\mathbf{h}_i^{\text{node}}) \prod_{k=0}^{n-1} P_\theta(\emptyset|\mathbf{h}_k^{\text{tail}}) \quad (3)$$

Representation $\mathbf{h}_i^{\text{node}}$ is computed as the weighted sum of the LSTM states of preceding nodes as defined by $\mathbf{O}$ (recall that $\mathbf{O} = [\mathbf{A}; \mathbf{S}]$):

$$\mathbf{h}_i^{\text{node}} := \sum_{j=0}^{m-1} \mathbf{S}_{ji} \text{LSTM}(\mathbf{h}_j^{\text{node}}, \mathbf{v}_j) + \sum_{k=0}^{n-1} \mathbf{A}_{ki} \mathbf{h}_k^{\text{token}}. \quad (4)$$

Note that the preceding node can be either a concept node (then the output of the LSTM, consuming the preceding node, is used) or a word (then we use its contextualized encoding). The first term in Equation 4 corresponds to the former situation, and the second one to the latter.

Note that this expression is 'recursive' – each node's representation $\mathbf{h}_i^{\text{node}}$ is computed based on representations of all the nodes $\mathbf{h}_j^{\text{node}}$; $i, j \in 1, \dots m-1$. Iterating the assignment defined by Equation 4 for a valid **discrete** generation order (i.e., a DAG, like the one given in Figure 2), will converge to a stationary point. Crucially, in this discrete case, the stationary point will be equal to the result of applying the autoregressive model (as used in test time, see Figure 4). The stationary point will be reached after T steps, where T is the number of nodes in the largest subgraph.[4] This 'message passing' process is fully differentiable and, importantly, well-defined for a relaxed generation order where $\mathbf{A}_{ki}$ and $\mathbf{S}_{ji}$ are non-binary. The equivalence between the train-time message passing and the test-time autoregressive computation with discrete $\mathbf{O}$ prevents the gap between training

[4] We use $T = 4$, as we do not expect subgraphs with more than 4 nodes.

and testing, as long as the optimization converges to a near-discrete solution.

The representations $\mathbf{h}_k^{\text{tail}}$, needed for the terms $P_\theta(\emptyset|\mathbf{h}_k^{\text{tail}})$ in Equation 3, are computed as:

$$\mathbf{h}_k^{\text{tail}} = \sum_{j=0}^{m-1} \mathbf{B}_{jk} \text{LSTM}(\mathbf{h}_j^{\text{node}}, \mathbf{v}_j) + (1 - \sum_{j=0}^{m-1} \mathbf{B}_{jk}) \mathbf{h}_k^{\text{token}}, \quad (5)$$

where $\mathbf{B}_{jk} = 1$ denotes that the concept node j is the last concept node before generating $\emptyset$ for the token k, else $\mathbf{B}_{jk} = 0$. E.g. in Figure 2, we have $\mathbf{B}_{11} = \mathbf{B}_{42} = 1$, and others are 0. Again, in the discrete case, the result will be exactly equivalent to what is obtained by running the corresponding autoregressive model (as in test time, Figure 4), but the computation is also well-defined and differentiable in the relaxed cases, where B_{jk} are real-valued.

While it is clear how S_{ji} and A_{ki} in Equation 4 and B_{jk} in Equation 5 are defined with discrete $\mathbf{O} = [\mathbf{A}; \mathbf{S}]$, we show how they can defined with relaxed (non-binary) $\mathbf{O}$ in Appendix B. The MLPs used to compute $P_\theta(\mathbf{v}_i|\mathbf{h}_i^{\text{node}})$ and $P_\theta(\emptyset|\mathbf{h}_i^{\text{tail}})$ are also defined there.

3.2 Relation Identification

Similarly to Lyu and Titov (2018), we use an arc-factored model for relation identification (i.e. predicting AMR edges):

$$P_\theta(\mathbf{E}|\mathbf{x}, \mathbf{O}, \mathbf{v}) = \prod_{i,j=1}^{m} P_\theta(\mathbf{E}_{ij}|\mathbf{h}_i^{\text{edge}}, \mathbf{h}_j^{\text{edge}}) \quad (6)$$

where $P_\theta(\mathbf{E}_{ij}|\mathbf{h}_i^{\text{edge}}, \mathbf{h}_j^{\text{edge}})$ is the softmax of the biaffine function of node representations $\mathbf{h}_i^{\text{edge}}$ and $\mathbf{h}_j^{\text{edge}}$. The node representations are defined as

$$\mathbf{h}_i^{\text{edge}} = \text{NN}^{\text{edge}}(\mathbf{h}_i^{\text{node}} \circ \sum_{k=0}^{n-1} \mathbf{A}_{ki}^{\infty} \mathbf{h}_k^{\text{token}}), \quad (7)$$

where $\circ$ denotes concatenation, $\mathbf{h}_i^{\text{node}}$ is defined in section 3.1, and $\mathbf{A}_{ki}^{\infty}$ determines whether node i is in a subgraph aligned to token k or not. Note that this is different from $\mathbf{A}_{ki}$ which encodes that the node i is the first node in the subgraph (e.g., in Figure 2, $\mathbf{A}_{11} = 0$ but $\mathbf{A}_{11}^{\infty} = 1$). In the continuous case, as used during training, $\mathbf{A}_{ki}^{\infty}$ can be thought of as the alignment probability that can be computed from $\mathbf{O}$ (see Appendix C).

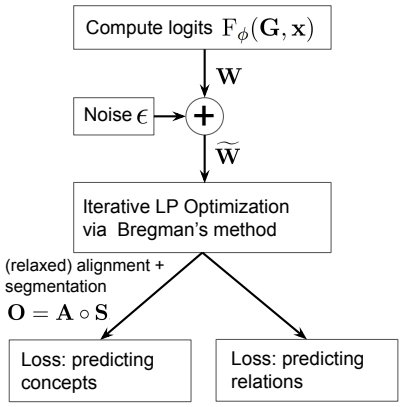

Figure 5: Overview of the computation graph.

4 Estimating Latent Generation Order

We show how to estimate the latent generation order jointly with the parser, as also illustrated in Figure 5.

4.1 Variational Inference

In Equation 2, marginalization over $\mathbf{O}$ is intractable due to the use of neural parameterization in $P_\theta(\mathbf{v}|\mathbf{x}, \mathbf{O})$ and $P_\theta(\mathbf{E}|\mathbf{x}, \mathbf{O}, \mathbf{v})$. Instead, we resort to the variational auto-encoder (VAE) framework (Kingma and Welling, 2014). VAEs optimize a lower bound on the marginal likelihood:

$$\log \sum_{\mathbf{O}} P_\theta(\mathbf{O}) P_\theta(\mathbf{v}|\mathbf{x}, \mathbf{O}) P_\theta(\mathbf{E}|\mathbf{x}, \mathbf{O}, \mathbf{v})$$
$$\geq \mathop{\mathrm{E}}_{\mathbf{O} \sim Q_\phi(\mathbf{O}|\mathbf{G}, \mathbf{x})} \log P_\theta(\mathbf{v}|\mathbf{x}, \mathbf{O}) P_\theta(\mathbf{E}|\mathbf{x}, \mathbf{O}, \mathbf{v})$$
$$- \text{KL}(Q_\phi(\mathbf{O}|\mathbf{G}, \mathbf{x}) || P_\theta(\mathbf{O})) \,, \quad (8)$$

where KL is the KL divergence, and $Q_\phi(\mathbf{O}|\mathbf{G}, \mathbf{x})$ (the encoder, aka the inference network) is a distribution parameterized with a neural network. The lower bound is maximized with respect to both the original parameters θ and the variational parameters ϕ. The distribution $Q_\phi(\mathbf{O}|\mathbf{G}, \mathbf{x})$ can be thought of as an approximation to the intractable posterior distribution $P_\theta(\mathbf{O}|\mathbf{G}, \mathbf{x})$.

4.2 Stochastic Softmax

In order to estimate the gradient with respect to the encoder parameters ϕ, we use the perturb-and-MAP framework (Papandreou and Yuille, 2011; Hazan and Jaakkola, 2012), specifically the stochastic softmax (Paulus et al., 2020), which is a generalization the Gumbel-softmax trick (Jang et al., 2016; Maddison et al., 2017) to the structured case.

With Stochastic Softmax, instead of sampling $\mathbf{O}$ directly, we independently compute logits $\mathbf{W} \in$

$\mathbb{R}^{(n+m)\times(m+1)}$ for all the potential edges in the generation order, and perturb them:

$$\mathbf{W} = \mathrm{F}_\phi(\mathbf{G}, \mathbf{x}) \tag{9}$$

$$\widetilde{\mathbf{W}} = \mathbf{W} + \epsilon, \quad \text{where} \quad \epsilon_{ij} \sim \mathcal{G}(0,1) \tag{10}$$

where F_ϕ is a neural module computing the logits (see Section 4.2.2), $\mathcal{G}(0,1)$ is the standard Gumbel distribution, and $\epsilon \in \mathbb{R}^{(n+m)\times(m+1)}$. Then, those perturbed logits $\widetilde{\mathbf{W}}$ are fed into a constrained convex optimization problem:

$$\mathbf{O}(\widetilde{\mathbf{W}}, \tau) := \arg\max_{\mathbf{O}\geq 0} \langle \widetilde{\mathbf{W}}, \mathbf{O}\rangle - \tau\langle \mathbf{O}, \log \mathbf{O}\rangle$$
$$\text{s.t.} \forall j < m \sum_{i=0}^{n+m-1} \mathbf{O}_{ij} = 1; \forall i \sum_{j=0}^{m} \mathbf{O}_{ij} = 1 \tag{11}$$

This is a linear programming (LP) relaxation of constraints discussed in Section 2.2, where we permit continuous-valued $\mathbf{O}$. Importantly, this LP relaxation is 'tight', and ensures that $\mathbf{O}(\widetilde{\mathbf{W}}, 0)$ is a valid generation order.[5]

Now, as we will show in the next section, the solution to this optimization $\mathbf{O}(\widetilde{\mathbf{W}}, \tau)$ can be obtained with a differentiable computation, thus, we write:

$$\mathbf{O}_\phi(\epsilon, \mathbf{G}, \mathbf{x}) = \mathbf{O}(\widetilde{\mathbf{W}}, \tau) \tag{12}$$

The entropy regularizor, weighted by $\tau > 0$ ('the temperature'), ensures differentiability with respect to $\mathbf{W}$ and, thus, with respect to ϕ, as needed to train the encoder.

We still need to handle the KL term in Equation 8. We define the prior probability $P_\theta(\mathbf{O})$ implicitly by having $\mathbf{W} = 0$ in the stochastic softmax framework. Even then, $\mathrm{KL}(Q_\phi(\mathbf{O}|\mathbf{G}, \mathbf{x})||P_\theta(\mathbf{O}))$ cannot be easily computed. Following Mena et al. (2018), we upper bound it by replacing it with $\mathrm{KL}(\mathcal{G}(\mathbf{W}, 1)||\mathcal{G}(0,1))$, which is available in closed form.

4.2.1 Bregman's Method

To optimize objective (11) we iterate over the following steps of optimization:

$$\mathbf{O}^{(0)} = \exp \frac{\widetilde{\mathbf{W}}}{\tau} \tag{13}$$

$$\forall j < m,\ \mathbf{O}^{(t+\frac{1}{2})}_{:,j} = \mathcal{T}(\mathbf{O}^{(t)}_{:,j}) \tag{14}$$

$$\mathbf{O}^{(t+\frac{1}{2})}_{:,m} = \mathbf{O}^{(t)}_{:,m} \tag{15}$$

$$\forall i,\ \mathbf{O}^{(t+1)}_{i,:} = \mathcal{T}(\mathbf{O}^{(t+\frac{1}{2})}_{i,:}) \tag{16}$$

where $\{i, :\}$ index ith row, $\{:, j\}$ index jth column and $\mathcal{T} = \frac{\mathbf{x}}{\sum_i \mathbf{x}_i}$ normalize the vectors. Intuitively, the alignment scores are initially computed from the logits $\widetilde{\mathbf{W}}$, without taking constraints into account, and then alternating optimization is used to 'fit' the constraints on columns and rows.

Proposition 1. $\lim_{t\to\infty} \mathbf{O}^{(t)} = \mathbf{O}(\widetilde{\mathbf{W}}, \tau)$ *where* $\mathbf{O}(\widetilde{\mathbf{W}}, \tau)$ *is defined in Equation 11.*

See Appendix I for a proof based on the proof for the Bregman method (Bregman, 1967). In practice, we take $T = 50$, and have $\mathbf{O}_\phi(\epsilon, \mathbf{G}, \mathbf{x}) = \mathbf{O}^{(T)}$. Importantly, this algorithm is highly parallelizable and amendable to batch implementation on GPU. We compute the gradients with unrolled optimization.

4.2.2 Neural Parameterization

We introduce the neural modules used for estimating logits $\mathbf{W} = \mathrm{F}_\phi(\mathbf{G}, \mathbf{x})$ and also the masking mechanism that both ensures acyclicity and enables the use of the copy mechanism. We have $\mathbf{W} = \mathbf{W}^{\text{raw}} + \mathbf{W}^{\text{mask}}$. First, we define the unmasked logits, $\mathbf{W}^{\text{raw}} = \mathbf{A}^{\text{raw}} \circ \mathbf{S}^{\text{raw}}$:

$$\mathbf{h}^{\text{g}} = \mathrm{RelGCN}(\mathbf{G}; \theta) \in \mathbb{R}^{m\times d}$$
$$\mathbf{A}^{\text{raw}} = \mathrm{BiAffine}^{\text{align}}(\mathbf{h}^{\text{token}}, \mathbf{h}^{\text{g}} \circ \mathbf{h}^{\text{end}}; \phi)$$
$$\mathbf{S}^{\text{raw}} = \mathrm{BiAffine}^{\text{segment}}(\mathbf{h}^{\text{g}}, \mathbf{h}^{\text{g}} \circ \mathbf{h}^{\text{end}}; \phi)$$

where RelGCN is a relational graph convolutional network (Schlichtkrull et al., 2018) that takes an AMR graph $\mathbf{G}$ and produces embeddings of its nodes informed by their neighbourhood in $\mathbf{G}$. $\mathbf{h}^{\text{end}} \in \mathbb{R}^{1\times d}$ is the trainable embedding of the terminal node, and $\mathbf{h}^{\text{token}} \in \mathbb{R}^{n\times d}$ is the BiLSTM encoding of a sentence from Section 3.1.

The masking also consists of two parts, the alignment mask and the segmentation mask, $\mathbf{W}^{\text{mask}} = \mathbf{A}^{\text{mask}} \circ \mathbf{S}^{\text{mask}}$. If a node is copy-able from at least one token, the alignment mask prohibits alignments from other tokens by setting the corresponding components $\mathbf{A}^{\text{mask}}_{ij}$ to $-\infty$.

Acyclicity is ensured by setting $\mathbf{S}^{\text{mask}}$ so that generation order with circles will get negative infinity in Equation 11. While there may be more general ways to encode acyclicity (Martins et al., 2009), we simply perform a depth-first search (DFS) from the root node[6] and permit an edge from node i and j only if i precedes j (not necessarily immediately) in the traversal. In other words, $\mathbf{S}^{\text{mask}}_{ij}$ is set to $-\infty$ for edges (i, j) violating this constraint. The rest of components in $\mathbf{S}^{\text{mask}}$ are set to 0. Note that this masking approach does not require changes in the optimization method.

[5] See proof in Appendix J.

[6] We use lexicographic ordering of edge labels in DFS.

5 Parsing

While we relied on the latent variable machinery to train the parser, we do not use it at test time. In fact, the encoder $Q_\phi(\mathbf{O}|\mathbf{G}, \mathbf{x})$ is discarded after training. At test time, the first step is to predict sets of concept nodes for every token using the concept identification model $P_\theta(\mathbf{v}|\mathbf{x}, \mathbf{O})$ (as shown in Figure 4). Note that the token-specific autoregressive models can be run in parallel across tokens. The second step is predicting relations between all the nodes, relying on the relation identification model $P_\theta(\mathbf{E}|\mathbf{x}, \mathbf{O}, \mathbf{v})$.

6 Experiments

We experiment on LDC2016E25 (AMR2.0) and LDC2020T02 (AMR3.0). The evaluation is based on Smatch (Cai and Knight, 2013), and the evaluation tool of Damonte et al. (2017). We compare our generation-order induction framework to pre-set segmentations, i.e., producing the segmentation on a preprocessing step. We vary the segmentation methods while keeping the rest of the model identical to our full model (i.e., the same autoregressive model and the learned alignment). We provide ablation studies for our induction framework. We further provide visualization of the induced generation order, along with extra details, in Appendix.

Rule-based Segmentation We introduce a hand-crafted rule-based segmentation method, which relies on rules designed to handle specific AMR constructions. In particular, we use the hand-crafted segmentation system of Lyu and Titov (2018), or, more specifically, its re-implementation by Zhang et al. (2019a). Arguably, this can be thought of as an upper bound for how well an induction method can do. This fixed segmentation can be incorporated into our latent-generation-order framework, so that the alignment between concept nodes and the tokens will still be induced. This is achieved by fixing $\mathbf{S}$, while still inducing $\mathbf{A}$.

Greedy Segmentation We provide a greedy strategy for segmentation that serves as a deterministic baseline. Many nodes are aligned to tokens with the copy mechanism. We could force the unaligned nodes to join its neighbors. This is very similar to the forced alignment of unaligned nodes used in the transition parser of Naseem et al. (2019). Again, the segmentation can be incorporated into our latent-generation-order framework by enforcing $\mathbf{S}$ and inducing $\mathbf{A}$. See Appendix E for extra details about the strategy.

Results In Table 1, we compare our models with recent AMR parsers (Xu et al., 2020a; Cai and Lam, 2020, 2019; Zhang et al., 2019a; Naseem et al., 2019; Lindemann et al., 2020; Lee et al., 2020), as well as (Lyu and Titov, 2018), which we build on, and (van Noord and Bos, 2017), the earliest model which does not exploit any rules. Overall, our model ('full') performs competitively, but lags behind scores reported by some of the very recent parsers.[7] However, except for a no-rule version of Cai and Lam (2020), all these models either use rules (Lee et al., 2020) (see Section 7) or specialized pretraining (Xu et al., 2020a).

Both our VAE model and the rule-based segmentation achieve high concept identification scores (Damonte et al., 2017). The relation identification component is however weaker than, e.g., (Cai and Lam, 2020). This may not be surprising, as we, following Lyu and Titov (2018), score edges independently, whereas (Cai and Lam, 2020) perform iterative refinement which is known to boost performance on relations (Lyu et al., 2019). Also, we use BiLSTM encoders, which – while cheaper to train and easier to tune – is likely weaker than Transformer encoders used by Astudillo et al.; Lee et al. While these modifications, along with using extra pre-training techniques and data augmentation, may further boost performance of our model, we believe that our model is strong enough for our purposes, i.e. demonstrating that informative segmentation can be induced without relying on any rules.

Indeed, our approach beats the greedy baseline and approaches the rule-based system. The performance gap between the rule-based system and VAE is smaller on AMR 3.0 (0.2 Smatch), possibly because the rules were developed for AMR 2.0.

Alignment Analysis We analyzed the alignment induced by our full model and the model which uses rule-based segmentation. The alignments were evaluated at the level of individual concepts: if a subgraph was aligned to a token, all its concepts were considered aligned to that token. The evaluation was done on 40 sentences. The alignment error rates were 12%, 15% and 14% for the full model, greedy methods and the rule-based method, respectively. This suggests that our

[7] Results from Lee et al. replace Roberta-large with Roberta-base in Astudillo et al.. With semi-supervised learning, Lee et al. (2020) achieved 81.3 Smatch score.

	R	Concept	SRL	Smatch
vNoord17♢	-			71.0
Lyu18	+	85.9	69.8	74.4
Zhang19	+	86	71	77.0
Naseem19	+	86	72	75.5
Cai19	-			73.2
Lindemann20	+			76.8
Lee20	+	88.1	78.2	**80.2**
Cai20:				
w/ rules	+	88.1	74.2	**80.2**
w/o rules	-	88.1	74.5	78.7
Xu20♢	-	87.4	78.9	**80.2**
greedy	-	87.5 ± 0.1	71.3 ±0.1	75.2 ± 0.1
rule	+	**88.7 ± 0.2**	73.6± 0.2	76.8 ± 0.4
full	-	88.3 ± 0.3	73.0 ± 0.2	76.1 ± 0.2

Table 1: Scores with standard deviation on the AMR 2.0 test set. digits. The columns 'R' indicate if hand-crafted rules are used for segmentation, ♢ indicates that the system used specialized pretraining or self-training. Our results are averaged over 4 runs.

Metric	Concept	SRL	Smatch
greedy	87.0	71.5	74.8
rule	**88.0**	72.6	**75.8**
full	87.8	**72.9**	75.6

Table 2: AMR 3.0 test set, averaged over 2 runs.

	Concept	SRL	Smatch
nothing learned	81.7	62.6	61.9
segmentation learned	86.0	69.1	70.5
alignment learned	87.6	71.1	74.4
full (all learned)	**88.3**	**73.0**	**76.1**

Table 3: Scores with different versions of latent segmentation on the AMR 2.0 test set, averaged over 2 runs

method is able to induce relatively accurate alignments, and joint induction of alignments with segmentation may be beneficial, or, at the very least, not detrimental to alignment quality.

Ablations To reconfirm that it is important to learn the segmentation and alignment, rather than to sample it randomly, we perform further ablations. In our parameterization, discussed in Section 4.2.2, it is possible to set $\mathbf{A}^{\text{raw}} = 0$ and/or $\mathbf{S}^{\text{raw}} = 0$, which corresponds to sampling from the prior in training (i.e. quasi-uniformly while respecting the constraints defined by masking) rather than learning them. We consider 4 potential options, from sampling everything uniformly to learning everything (as in our method). The results are summarized in Table 3. As expected, the full model performs the best, demonstrating that it is important to learn both alignments and segmentation. Interestingly, both 'segmentation learned' and 'alignment learned' obtain reasonable performance, but the 'nothing learned' model fails badly.

7 Related Work

A wide range of approaches for AMR parsing have been explored, including graph-based models (Flanigan et al., 2014; Werling et al., 2015; Lyu and Titov, 2018; Zhang et al., 2019a), transition-based models (Damonte et al., 2017; Ballesteros and Al-Onaizan, 2017), grammar-based models (Peng et al., 2015; Artzi et al., 2015; Groschwitz et al., 2018; Lindemann et al., 2020) and neural autoregressive models (Konstas et al., 2017; van Noord and Bos, 2017; Zhang et al., 2019b; Cai and Lam, 2020; Xu et al., 2020b).

The majority of strong parsers rely on explicit graph segmentation in training. Typically, the segmentation is dealt with hand-crafted rules, with rule templates developed by studying training set statistics and ensuring the necessary level of coverage. Alternatively, Artzi et al. (2015); Groschwitz et al. (2017, 2018); Lindemann et al. (2020); Peng et al. (2015) using existing grammar formalisms to segment the AMR graphs. Astudillo et al. (2020); Lee et al. (2020) - while not not relying on graph recategorization rules - use a rule system to 'pack' and 'unpack' nodes. In recent work, strong results were obtained without using any explicit segmentation and alignment, relying on sequence-sequence models (Xu et al., 2020b; Cai and Lam, 2020), still the rules appear useful even with these strong models (Cai and Lam, 2020).

More generally, outside of AMR parsing, differentiable relaxations of latent structure representations have received attention in NLP (Kim et al., 2017; Liu and Lapata, 2018), including previous applications of the perturb-and-MAP framework (Corro and Titov, 2019). From a more general goal perspective – inducing a segmentation of a linguistic structure – our work is related to tree-substitution grammar induction (Sima'an et al., 1995; Cohn et al., 2010), the DOP paradigm (Bod et al., 2003) and unsupervised semantic parsing (Poon and Domingos, 2009; Titov and Klementiev, 2011), though the methods used in that previous work are very different from ours.

8 Conclusions

To eliminate hand-crafted segmentation systems used in previous AMR parsers, we cast the alignment and segmentation as generation-order induction. We propose to treat this generation order as a latent variable in a VAE framework. Our method outperforms a simple segmentation heuristic and approaches the performance of a method using rules designed to handle specific AMR constructions. Importantly, while the latent variable modeling machinery is used in training, the parser is very simple at test time. It tags the input words with AMR concept nodes with autoregressive models and then predicts relations between the nodes independently from each other.

Vanilla sequence-to-sequence models are known to struggle with out-of-distribution generalization (Lake and Baroni, 2018; Bahdanau et al., 2019), and, in the future work, it would be interesting to see if this holds for AMR and if such more constrained and structured methods as ours can better deal with this more challenging but realistic setting.

Acknowledgments

We thank the reviewers for their useful feedback and comments. The project was supported by the European Research Council (ERC StG BroadSem 678254), the Dutch National Science Foundation (NWO VIDI 639.022.518) and Bloomberg L.P.

References

Rafael Anchiêta and Thiago Pardo. 2018. Towards AMR-BR: A SemBank for Brazilian Portuguese language. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage ccg semantic parsing with amr. In *EMNLP*.

Ramón Fernández Astudillo, Miguel Ballesteros, Tahira Naseem, A. Blodgett, and Radu Florian. 2020. Transition-based parsing with stack-transformers. *ArXiv*, abs/2010.10669.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. 2019. Systematic generalization: what is required and can it be learned? *ICLR*.

Miguel Ballesteros and Yaser Al-Onaizan. 2017. Amr parsing using stack-lstms. *ArXiv*, abs/1707.07755.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking.

Rens Bod, Remko Scha, Khalil Sima'an, et al. 2003. *Data-oriented parsing*. University of Chicago Press.

Claire Bonial, Bianca Badarau, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Tim O'Gorman, Martha Palmer, and Nathan Schneider. 2018. Abstract Meaning Representation of constructions: The more we include, the better the representation. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Claire Bonial, Lucia Donatelli, Mitchell Abrams, Stephanie M. Lukin, Stephen Tratz, Matthew Marge, Ron Artstein, David Traum, and Clare Voss. 2020. Dialogue-AMR: Abstract Meaning Representation for dialogue. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 684–695, Marseille, France. European Language Resources Association.

Julia Bonn, Martha Palmer, Zheng Cai, and Kristin Wright-Bettner. 2020. Spatial AMR: Expanded spatial annotation in the context of a grounded Minecraft corpus. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4883–4892, Marseille, France. European Language Resources Association.

L. M. Bregman. 1967. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *Ussr Computational Mathematics and Mathematical Physics*, 7:200–217.

Deng Cai and Wai Lam. 2019. Core semantic first: A top-down approach for AMR parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3799–3809, Hong Kong, China. Association for Computational Linguistics.

Deng Cai and Wai Lam. 2020. Amr parsing via graph-sequence iterative inference. In *ACL*.

Shu Cai and K. Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *ACL*.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. *The Journal of Machine Learning Research*, 11:3053–3096.

Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. 2014. *Integer Programming*. Springer Publishing Company, Incorporated.

Caio Corro and Ivan Titov. 2019. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder. *ArXiv*, abs/1807.09875.

Marco Damonte and Shay B. Cohen. 2018. Cross-lingual abstract meaning representation parsing. In *Proceedings of NAACL*.

Marco Damonte, Shay B. Cohen, and Giorgio Satta. 2017. An incremental parser for abstract meaning representation. In *EACL*.

Shibhansh Dohare and Harish Karnick. 2017. Text Summarization using Abstract Meaning Representation. *arXiv preprint arXiv:1706.01678*.

Jeffrey Flanigan, Sam Thomson, Jaime G. Carbonell, Chris Dyer, and Noah A. Smith. 2014. A discriminative graph-based parser for the abstract meaning representation. In *ACL*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, M. Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. *ArXiv*, abs/1803.07640.

Jonas Groschwitz, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2017. A constrained graph algebra for semantic parsing with AMRs. In *IWCS 2017 - 12th International Conference on Computational Semantics - Long papers*.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. Amr dependency parsing with a typed semantic algebra. In *ACL*.

Hardy and Andreas Vlachos. 2018. Guided neural language generation for abstractive summarization using abstract meaning representation. In *EMNLP*.

Tamir Hazan and Tommi Jaakkola. 2012. On the partition function and random maximum a-posteriori perturbations. *arXiv preprint arXiv:1206.6410*.

Fuad Issa, Marco Damonte, Shay B. Cohen, Xiaohui Yan, and Yi Chang. 2018. Abstract meaning representation for paraphrase detection. In *NAACL-HLT*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *ArXiv*, abs/1611.01144.

Bevan K. Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, and Kevin Knight. 2012. Semantics-based machine translation with hyperedge replacement grammars. In *COLING*.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. 2017. Structured attention networks. *arXiv preprint arXiv:1702.00887*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Diederik P. Kingma, Tim Salimans, and M. Welling. 2017. Improved variational inference with inverse autoregressive flow. *ArXiv*, abs/1606.04934.

Diederik P Kingma and Max Welling. 2014. Auto-encoding variational bayes. *International Conference on Learning Representations*.

Ioannis Konstas, Srini Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural AMR: Sequence-to-sequence models for parsing and generation. In *ACL*.

Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2873–2882. PMLR.

Youngsuk Lee, Ramón Fernández Astudillo, Tahira Naseem, Revanth Gangi Reddy, Radu Florian, and S. Roukos. 2020. Pushing the limits of amr parsing with self-learning. *ArXiv*, abs/2010.10673.

Percy Liang. 2016. Learning executable semantic parsers for natural language understanding. *Communications of the ACM*, 59(9):68–76.

Matthias Lindemann, Jonas Groschwitz, and Alexander Koller. 2020. Fast semantic parsing with well-typedness guarantees. *ArXiv*, abs/2009.07365.

Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman M. Sadeh, and Noah A. Smith. 2015. Toward Abstractive Summarization Using Semantic Representations. In *HLT-NAACL*.

Yang Liu and Mirella Lapata. 2018. Learning structured text representations. *Transactions of the Association for Computational Linguistics*, 6:63–75.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Chunchuan Lyu, Shay B. Cohen, and Ivan Titov. 2019. Semantic role labeling with iterative structure refinement. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1071–1082, Hong Kong, China. Association for Computational Linguistics.

Chunchuan Lyu and Ivan Titov. 2018. AMR parsing as graph prediction with latent alignment. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 397–407, Melbourne, Australia. Association for Computational Linguistics.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. *ArXiv*, abs/1611.00712.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

André FT Martins, Noah A Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 342–350.

Gonzalo E. Mena, David Belanger, Scott W. Linderman, and Jasper Snoek. 2018. Learning latent permutations with gumbel-sinkhorn networks. *ArXiv*, abs/1802.08665.

Noelia Migueles-Abraira, Rodrigo Agerri, and Arantza Diaz de Ilarraza. 2018. Annotating Abstract Meaning Representations for Spanish. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Arindam Mitra and Chitta Baral. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *AAAI*.

Tahira Naseem, Abhishek Shah, Hui Wan, Radu Florian, Salim Roukos, and Miguel Ballesteros. 2019. Rewarding smatch: Transition-based amr parsing with reinforcement learning. *ArXiv*, abs/1905.13370.

George Papandreou and Alan L. Yuille. 2011. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. *2011 International Conference on Computer Vision*, pages 193–200.

Max B. Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J. Maddison. 2020. Gradient estimation with stochastic softmax tricks. *ArXiv*, abs/2006.08063.

Xiaochang Peng, Linfeng Song, and Daniel Gildea. 2015. A synchronous hyperedge replacement grammar based approach for AMR parsing. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 32–41, Beijing, China. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Hoifung Poon and Pedro Domingos. 2009. Unsupervised semantic parsing. In *Proceedings of the 2009 conference on empirical methods in natural language processing*, pages 1–10.

Nima Pourdamghani, Yang Gao, Ulf Hermjakob, and Kevin Knight. 2014. Aligning English strings with Abstract Meaning Representation graphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 425–429, Doha, Qatar. Association for Computational Linguistics.

M. Schlichtkrull, Thomas Kipf, P. Bloem, R. V. Berg, Ivan Titov, and M. Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*.

Khalil Sima'an, Rens Bod, Steven Krauwer, and Remko Scha. 1995. Efficient disambiguation by means of stochastic tree substitution grammars. In *Recent Advances in NLP*, volume 136.

Li Song, Yuling Dai, Yihuan Liu, Bin Li, and Weiguang Qu. 2020. Construct a sense-frame aligned predicate lexicon for Chinese AMR corpus. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 2962–2969, Marseille, France. European Language Resources Association.

Linfeng Song, Daniel Gildea, Yue Zhang, Zhiguo Wang, and Jinsong Su. 2019. Semantic neural machine translation using amr. *Transactions of the Association for Computational Linguistics*, 7:19–31.

Ida Szubert, Marco Damonte, Shay B. Cohen, and Mark Steedman. 2020. The role of reentrancies in abstract meaning representation parsing. In *Findings of EMNLP*.

Ivan Titov and Alexandre Klementiev. 2011. A Bayesian model for unsupervised semantic parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1445–1455.

Rik van Noord and Johan Bos. 2017. Neural semantic parsing by character-based translation: Experiments with abstract meaning representations. *ArXiv*, abs/1705.09980.

Keenon Werling, Gabor Angeli, and Christopher D. Manning. 2015. Robust subgraph generation improves abstract meaning representation parsing. In *ACL*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. 2017. Scene graph generation by iterative message passing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5410–5419.

Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. 2020a. Improving AMR parsing with sequence-to-sequence pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2501–2511, Online. Association for Computational Linguistics.

Dongqin Xu, Junhui Li, Muhua Zhu, Min Zhang, and Guodong Zhou. 2020b. Improving amr parsing with sequence-to-sequence pre-training. *ArXiv*, abs/2010.01771.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019a. Amr parsing as sequence-to-graph transduction. *ArXiv*, abs/1905.08704.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019b. Broad-coverage semantic parsing as transduction. *ArXiv*, abs/1909.02607.

A Decoding

We need to model the identification of the root node of the AMR graph. We specify the root identification as:

$$P_\theta(i|\mathbf{x}, \mathbf{O}, \mathbf{v}) = \frac{\exp(\langle \mathbf{h}^{\text{root}}, \mathbf{h}_i^e \rangle)}{\sum_{j=0}^{m-1} \exp(\langle \mathbf{h}^{\text{root}}, \mathbf{h}_j^e \rangle)} \quad (17)$$

where $\mathbf{h}^{\text{root}}$ is a trainable vector. Inspired by Zhang et al. (2019a), who rely on AMR graphs being closely related to dependency trees, we first decode the AMR graph as a maximum spanning tree with log probability of most likely arc-label as edge weights. The reentrancy edges are added afterwards, if their probability is larger than 0.5. We add at most 5 reentrancy edges, based on the empirical founding of Szubert et al. (2020).

B Concept Identification Detail

Now, we specify $P_\theta(\mathbf{v}_i|\mathbf{h}_i^{\text{node}})$ and $P_\theta(\mathbf{v}_m|\mathbf{h}_i^{\text{tail}})$ with a copy mechanism. Formally, we have a small set of candidate nodes $\mathcal{V}(\mathbf{x}_i)$ for each token $\mathbf{x}_i$, and a shared set of candidate nodes $\mathcal{V}^{\text{share}}$, which contain v_{copy}. This, however, depends on the token, yet we are learning a latent alignment. During training, we consider all the union of candidate nodes from all possible tokens$\mathcal{V}(\mathbf{v}_i) = \cup_{j:\mathbf{v}_i \in \mathcal{V}(\mathbf{x}_j)} \mathcal{V}(\mathbf{x}_j)$. We abuse notation slightly, and denote the embedding of node i by $\mathbf{v}_i$. At training time, for node $\mathbf{v}_i$, we have

$$\mathbf{h}_i^c = \text{NN}^{\text{node}}(\mathbf{h}_i^{\text{node}}; \theta) \quad (18)$$

$$\begin{aligned} P_\theta(\mathbf{v}_i|\mathbf{h}_i^{\text{node}}) =& \frac{[[\mathbf{v}_i \in \mathcal{V}^{\text{share}}]] \exp(\langle \mathbf{v}_i, \mathbf{h}_i^c \rangle)}{\sum_{v \in \mathbf{v}} \exp(\langle v, \mathbf{h}_i^c \rangle)} \\ &+ \frac{[[\mathbf{v}_i \in \mathcal{V}(\mathbf{v}_i)]] \exp(\langle v_{copy}, \mathbf{h}_i^c \rangle)}{\sum_{v \in \mathbf{v}} \exp(\langle v, \mathbf{h}_i^c \rangle)} \\ &\times \frac{\exp(\text{S}(\mathbf{v}_i, \mathbf{h}_i^c))}{\sum_{v \in \mathcal{V}(\mathbf{v}_i)} \exp(\text{S}(v, \mathbf{h}_i^c)} \end{aligned} \quad (19)$$

where NN is a standard one-layer feedforward neural network, and $[[\ldots]]$ denotes the indicator function. $\text{S}(v, \mathbf{h}_i^c)$ assigns a score to candidate nodes given the hidden state. To use pre-trained word embedding (Pennington et al., 2014), the representation of v is decomposed into primitive category embedding ($\mathcal{C}(v)$[8] and surface lemma embedding. The score function is then a biaffine scoring based on the embeddings and hidden states($\mathcal{L}(v)$ $\text{S}(v, \mathbf{h}_i^c) = \text{Biaffine}(\mathcal{C}(v) \circ \mathcal{L}(v), \mathbf{h}_i^c; \theta)$. For the terminal nodes, we have:

$$\mathbf{h}_i^t = \text{NN}^{\text{node}}(\mathbf{h}_i^{\text{tail}}; \theta) \quad (20)$$

$$P_\theta(\mathbf{v}_m|\mathbf{h}_i^{\text{tail}}) = \frac{\exp(\langle \mathbf{v}_m, \mathbf{h}_i^t \rangle)}{\sum_{v \in \mathbf{v}} \exp(\langle v, \mathbf{h}_i^t \rangle)} \quad (21)$$

At testing time, we perform greedy decoding to generate nodes from each token in parallel until either terminal node or T nodes are generated.

C Computing B and $\mathbf{A}^\infty$

We obtain $\mathbf{B}$ by having:

$$\mathbf{B} = \mathbf{A}[\mathbf{S}_{:,:m} + \text{Diag}(\mathbf{S}_{:,m})]^T; \quad (22)$$

where $\mathbf{S}_{:,:m}$ takes the submatrix of $\mathbf{S}$, excluding the last column, and $\text{Diag}(\mathbf{S}_{:,m}$ is the diagonal matrix whose diagonal entries are the last column of $\mathbf{S}$. Intuitively, $[\mathbf{S}_{:,:m} + \text{Diag}(\mathbf{S}_{:,m})]$ can be thought as a Markov transition matrix that passes down the alignment along the generation order, but keeps the alignment mass if the node will generate $\emptyset$. We truncate the transition at $T = 4$, as we do not expect a subgraph containing more than 4 nodes.

To obtain $\mathbf{A}^\infty$, we observe $\mathbf{A}^\infty$ should obey the following self-consistency equation:

$$\mathbf{A}^\infty = \mathbf{A}^\infty \mathbf{S}_{:,:m} + \mathbf{A} \quad (23)$$

This means, node j is generated from token k iff node i is is generated from token k and node i generates node j or node j is directly generated from token k. This $\mathbf{A}^\infty$ can be computed by initializing $\mathbf{A}^\infty = \mathbf{A}$, and repeating Equation 23 as assignment for $T = 4$ times. Intuitively, the $\mathbf{A}^\infty$ alignment is passed down along the generation order, while keeping getting alignment mass from the first node alignment. As a result, all nodes get assigned an alignment. As an alternative motivation, the above algorithmic assignment works as a truncated power series expansion of self-consistency equation solution $\mathbf{A}^\infty = [I - \mathbf{S}_{:,:m}]^{-1}\mathbf{A}$.

D Ablation on Stochastic Softmax

Our full model uses the Straight-Through (ST) gradient estimator and the Free Bits trick with $\lambda = 10$ (Kingma et al., 2017).[9] We perform an analysis of different variations of the stochastic softmax: (1) the soft stochastic softmax is the original one with the entropic regularizer (see Section 4.2); (2) the rounded stochastic softmax, which selects the highest scored next node from each tokens and

[8] AMR nodes have primitive category, including string, number, frame, concept and special nodes (e.g. polarity).

[9] The Free Bits trick is used to prevent 'the posterior collapse' (Kingma et al., 2017). In other words, we use $\max(\lambda, \text{KL}(\mathcal{G}(\mathbf{W}, 1)||\mathcal{G}(0, 1)))$ for the KL divergence regularizer.

Metric	Concept	SRL	Smatch
no free bits	83.5	66.3	66.1
soft	84.9	68.1	70.3
rounding	87.7	71.8	74.5
straight-through	**88.3**	**73.0**	**76.1**

Table 4: Scores with different versions of latent segmentation on the AMR 2.0 test set. Scores are averaged over 2 runs

concept nodes based on the soft stochastic softmax;[10] (3) our full model with the ST estimator. All those models use Free Bits ($\lambda = 10$), while for 'no free bits' $\lambda = 0$. As we can see in Table 4, there is a substantial gap between using structured ST and the two other versions. This illustrates the need for exposing the parsing model to discrete structures in training. Also, the Free Bits trick appears crucial as it prevents the (partial) posterior collapse in our model. We inspected the logits after training and observed that, without free-bits, the learned $\mathbf{W}$ are very small, in the $[-0.01, +0.01]$ range.

E Greedy Segmentation

We present a greedy strategy for segmentation that serves as a deterministic baseline. This greedy segmentation can be used in the same way as the rule-based segmentation by setting $\mathbf{S}^{\text{mask}}$.

Many nodes are aligned to tokens with the copy mechanism. We could force the unaligned nodes to join their neighbors. This is very similar to the forced alignment of unaligned nodes used in the transition parser of Naseem et al. (2019). We traversal the AMR graph the same way as we do when we produce the masking (Section 4.2.2). During the traversal, we greedily combine subgraphs until one of the constraints is violated: (1) the combined subgraph will have more than 4 nodes; (2) the combined subgraph will have more than 2 copy-able nodes. We present the algorithm recursively (see Algorithm 1). Variable $\mathbf{z}_i$ indicates whether node i is copy-able and $T = 4$ represent the maximum subgraph size; n denotes the current subgraph size; z indicates whether the current subgraph contains a copy-able node; k is the last node in the current subgraph, which is used to generate to future nodes in a subgraph. The condition $n + n' \leq T \wedge z' + z \leq 1$ determines whether we combine the current subgraph rooted at node i and the subgraph rooted at node j. Running the algorithm on an AMR graph and the root index will get us the entire segmentation. This greedy method does not require any expert knowledge about AMR, so this should serve as a baseline.

Input: graph $\mathbf{G}$, node index i
Result: segmentation $\mathbf{S}$, n, z, k

$\mathbf{S} = \mathbf{0}$, $k = i$, $n = 1$, $z = \mathbf{z}_i$;
forall $j \in \text{Child}[i]$ **do**
  if j notvisited **then**
    $\mathbf{S}', n', z', k' = \text{Greedy}(\mathbf{G}, j)$;
    $\mathbf{S} = \mathbf{S} + \mathbf{S}'$;
    if $n + n' \leq T \wedge z' + z \leq 1$ **then**
      $\mathbf{S}_{kj} = 1$, $n = n + n'$,
      $z = z + z'$ $k = k'$;
    end
  end
end

Algorithm 1: Greedy Segmentation

F Visualizing Generation Order

In Figures 6, 7, 8, 9,[11] we present one example of the induced learned for our three variations of stochastic softmax, and one with rule-based segmentation. The nodes are represented in []. Their gold AMR is:

```
(m / make-01
    :ARG0 (t / they)
    :ARG1 (t2 / thing
        :ARG2-of (p / poster-01)
        :ARG0-of (e / express-01
            :ARG1 (t3 / thing
                :ARG1-of (o / opine-01
                    :ARG0 t)))))
```

As we can see, the standard stochastic softmax indeed produces soft latent structure that might result in large training/testing gap. Furthermore, the rounding strategy does not satisfy the constraint that every concept node can only be generated from one token or another concept node (i.e. [poster-01] is generated twice, and [thing] is never generated.). Meanwhile, the straight through stochastic softmax produce a valid generation order. In Appendix J, we will show the validity formally. It is worth to note that our learned generation order differs from the rule based one. When producing the rule-based segmentation, '(t2 / thing :ARG0-of (e / express-01

[10] Such rounding does not provide any guarantee of being a valid generation order, but serves as a baseline. In general, a threshold function (at 0.5) can be applied if the constraints have no structure.

[11] Incidentally, the greedy segementation produces the same segmentation as rule-based in this example.

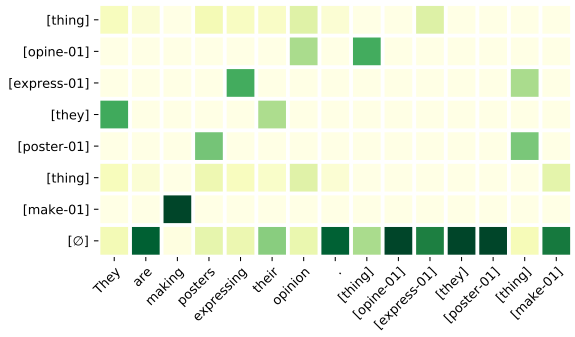

Figure 6: Example of Soft Stochastic Softmax Latent Generation Order.

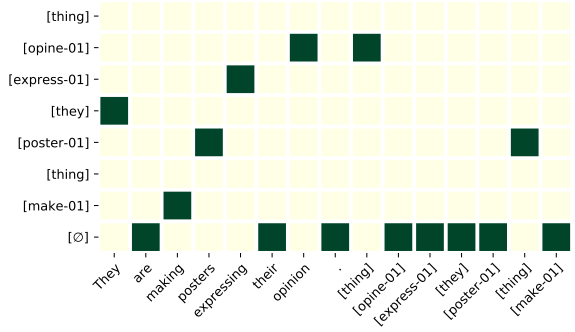

Figure 7: Example of Rounded Stochastic Softmax Latent Generation Order.

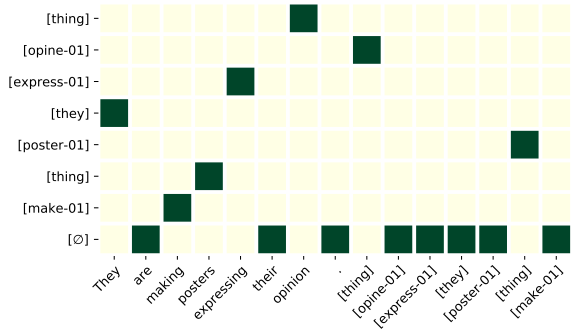

Figure 8: Example of Hard (straight-through) Stochastic Softmax Latent Generation Order.

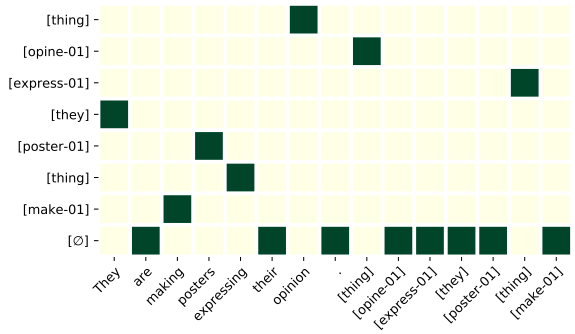

Figure 9: Example of Rule-Segmentation Stochastic Softmax Latent Generation Order.

)' took precedence over '(t2 / thing :ARG2-of (p / poster-01)' due to the order over traversal edges. The learned model, however, figured out that the poster is the thing.

G Hyper-Parameters

We use RoBERTa-large (Liu et al., 2019) from Wolf et al. (2019) for contextualised embeddings before LSTMs. BiLSTM for concept identification has 1 layer, and BiLSTM for relation identification has 2 layers. Both have hidden size 1024. Their averaged representation is used for alignment. Rel-GCN used 128 hidden units and 1 hidden layer (plus one input layer and output layer). Relation identification used 128 hidden units. The LSTM for the locally auto-regressive model is one layer with 1024 hidden units. Adam (Kingma and Ba, 2015) is used with learning rate $3e-4$ and beta=$0.9, 0.99$. Early stopping is used with maximum 60 epochs of training. Dropout is set at 0.33. Those hyper-parameters are selected manually, we basically followed the standard model size as in (Lyu and Titov, 2018; Zhang et al., 2019a). We will release the code based on the AllenNLP framework (Gardner et al., 2018).

H Pre-and-Post processing

We follow Lyu and Titov (2018) for pre-and-post processing. We use CoreNLP (Manning et al., 2014) for tokenization and lemmatization. The copy-able dictionary is built with the rules based on string matching between lemmas and concept node string as in Lyu and Titov (2018).

For post-processing, wiki tags are added after the named entity being produced in the graph via a look-up table built from the training set or provided by CoreNLP. We also collapse nodes that represent the same pronouns as heuristics for co-reference resolution.

I Proof of Proposition 1

We prove Proposition 1 based on the Bregman method (Bregman, 1967). The Bregman's method solves convex optimization with a set of linear equalities, the setting is as follows:

$$\min_{x \in \bar{\Omega}} \; F(x) \qquad \text{s.t. } Ax = b, \qquad (24)$$

where F is strongly convex and continuously differentiable. Note that A is not our alignment, but denotes a matrix that represents constraints. Two important ingredients are Bregman's diver-

gence $D_F(x,y) = F(x) - F(y) - \langle \nabla F(y), x - y \rangle$, and Bregman's projection: $P_{\omega,F}(y) = \arg\min_{x \in \omega} D_F(x,y)$, where ω represents constraint. Now, the Bregman's method works as: Intuitively, Bregman's method iteratively performs

pick $y^0 \in \{y \in \Omega | \nabla F(y) = uA, u \in \mathbb{R}^m\}$;
for $t \leftarrow 1$ **to** ∞ **do**
  $y_0^t \leftarrow y^{t-1}$;
  for $i \leftarrow 1$ **to** m **do**
    $y_i^t \leftarrow P_{A_i x = b_i, F}(y_{i-1}^t)$;
  end
  $y^t \leftarrow y_m^t$;
end

Algorithm 2: Bregman's method for solving convex optimization over linear constraints

alternating projections w.r.t. each constraint. After each projection, the score F is lowered by the construction of Bregman's projection. Such alternating projections eventually converge, and with careful initialization solve the optimization problem.

Theorem 1 (Bregman 1967)**.** $\lim_{t\to\infty} y^t$ *solves the optimization problem 24.*

Proof of Proposition 1. We show Proposition 1 by showing the Algorithm defined by equations 13, 14, 15 and 16 implements Bregman's method. Then, Proposition 1 follows from Theorem 1.

Now, we build Bregman's method for our optimization problem 11. For simplicity, we focus on the linear algebraic structure, but do not strictly follow the standard matrix notation. We have $\mathbf{O}$ as variable, and $F(\mathbf{O}) = -\langle \widetilde{\mathbf{W}}, \mathbf{O} \rangle + \tau \langle \mathbf{O}, \log \mathbf{O} - 1 \rangle$[12]. For initialization, we have $\nabla F(\mathbf{O}) = -\widetilde{\mathbf{W}} + \tau \log \mathbf{O}$. Take $u = 0$, we have $\mathbf{O}^{(0)} = \exp(\frac{\widetilde{\mathbf{W}}}{\tau}) \iff \log \mathbf{O}^{(0)} = \frac{\widetilde{\mathbf{W}}}{\tau}$. This corresponds to the initialization step as in our Equation 13. Then, we iterate through constraints to perform Bregman's projection. First, the column normalization constraints $\forall j < m, \sum_{i=0}^{n+m-1} \mathbf{O}_{ij} = 1$. Take a $j < m$, we need to compute $P_{\sum_{i=0}^{n+m-1} \mathbf{O}_{ij} = 1, F}(\mathbf{O}(t))$. A very important property is that our $F(\mathbf{O}) = \sum_{ij} f_{ij}(\mathbf{O}_{ij})$, where $f_{ij}(\mathbf{O}_{ij}) = -\widetilde{\mathbf{W}}_{ij}\mathbf{O}_{ij} + \tau \mathbf{O}_{ij}(\log \mathbf{O}_{ij} - 1)$. Moreover, $D_F(x,y) = 0 \iff x = y$. Therefore, for variables that are not involved in the constraints, they are kept the same. To simplify notation, we extend the domain of F to parts of the variable. e.g., $F(\mathbf{O}_{:,j}) = \sum_i f_{ij}(\mathbf{O}_{ij})$. Now, let us focus on column j, we have:

$$\arg\min_{x:\sum_i x_i = 1} F(x) - F(\mathbf{O}_{:,j}) - \langle \nabla F(\mathbf{O}_{:,j}), x - \mathbf{O}_{:,j} \rangle \tag{25}$$

$$= \arg\min_{x:\sum_i x_i = 1} -\langle \widetilde{\mathbf{W}}_{:,j}, x \rangle + \tau \langle x, \log x - 1 \rangle - \langle \nabla F(\mathbf{O}_{:,j}), x \rangle \tag{26}$$

$$= \arg\min_{x:\sum_i x_i = 1} -\langle \widetilde{\mathbf{W}}_{:,j}, x \rangle + \tau \langle x, \log x - 1 \rangle - \langle -\widetilde{\mathbf{W}}_{:,j} + \tau \log \mathbf{O}_{:,j}, x \rangle \tag{27}$$

$$= \arg\min_{x:\sum_i x_i = 1} \tau \langle x, \log x - 1 \rangle + \langle \tau \log \mathbf{O}_{:,j}, x \rangle \tag{28}$$

$$= \arg\min_{x:\sum_i x_i = 1} \langle x, \log x - 1 \rangle + \langle \log \mathbf{O}_{:,j}, x \rangle \tag{29}$$

$$= \mathrm{Softmax}(\log \mathbf{O}_{:,j}) \tag{30}$$

since when iterating over these mutually non-overlapping constraints, the non-focused variables are always kept the same. It is hence equivalent to computing them in parallel, which is expressed in our column normalization step 14. Similarly, we can derive row normalization step 16. Therefore, our algorithm is an implementation of Bregman's method, and Proposition 1 follows from Theorem 1. □

J Generation Order is Discrete by LP

If $\mathbf{O}(\widetilde{\mathbf{W}}, 0)$ is integral valued, it belongs to $\mathcal{O}$ by definition. In most cases, there is no guarantee that the linear programming in the relaxed space yields a solution that is also an integer. However, in our cases, we have the following result:

Proposition 2. *With probability 1, a unique* $\mathbf{O}(\widetilde{\mathbf{W}}, 0) \in \{0,1\}^{(n+m)\times(m+1)}$*, where* $\mathbf{O}(\widetilde{\mathbf{W}}, 0)$ *is defined in Equation 11.*

Intuitively, this is a generalization of a classical result about perfect matching on bipartite graph (Conforti et al., 2014). To prove this, we need the following theorems from integer linear programming.

Theorem 2 (Conforti et al. 2014, page 130,133)**.** *l Let A be an $q \times p$ integral matrix. For all integral vectors d, l, u and $c \in \mathbb{R}^p$, $\max\{\langle c, x \rangle : Ax = d, l \le x \le u\}$ is attained by an integral vector x if and only if A is totally unimodular.*[13]

[12] This regularizer differs from the original one in 11 by a constant $m + n$, due to the constraints. So, the optimization problem is equivalent.

[13] A is totally unimodular if every square submatrix has determinant $0, \pm 1$. We combined a few theorems and definitions from Conforti et al. (2014) into this theorem.

Note that this theorem does not claim at all the solution is integer, nor that it is unique. However, one should understand this limitation as some degenerate case of c. However, a total unimodular matrix does characterize the convex hull of its integral points. To prove this, we need an additional lemma.

Lemma 1 (Conforti et al. 2014, page 21)**.** *l Let $S \in \mathbb{R}^n$ and $c \in \mathbb{R}^n$. Then $\sup\{\langle c, s\rangle : s \in S\} = \sup\{\langle c, s\rangle : s \in \mathrm{Conv}(S)\}$. Furthermore, the supremum of $\langle c, s\rangle$ is attained over S if and only if it is attained over $\mathrm{Conv}(S)$.*

where $\mathrm{Conv}(S)$ is the convex hull of S. Now we have the following proposition:

Proposition 3. *Let A be an $q \times p$ integral matrix. For all integral vectors d, l, u ,and $c \in \mathbb{R}^p$ such that $\{x \in \{0,1\}^p | Ax = d, l \le x \le u\}$ is a finite set, $\{x \in \mathbb{R}^p | Ax = d, l \le x \le u\} = \mathrm{Conv}(\{x \in \{0,1\}^p | Ax = d, l \le x \le u\})$ if and only if A is totally unimodular.*

In other words, we know the LP relaxation is the convex hull.

Proof. By Theorem 2, A is totally unimodular is equivalent to maximum is attained by an integer solution. Clearly, the LP relaxation contains the convex hull. So, we only need to show that the LP relaxation does not contain any more points. Now suppose the LP relaxation contains another point x' that's not in the convex hull. Since, we restrict our discussion on finite set of integer, both the $\{x'\}$ and the convex hull is closed set. Then by the separation theorem, we have a vector c s.t. $\langle c, x'\rangle > \langle c, x\rangle \forall x \in \mathrm{Conv}(\{x \in \{0,1\}^p | Ax = d, l \le x \le u\})$, which contradicts Lemma 1. □

Theorem 3 (Conforti et al. 2014, page 133,134)**.** *A $0, \pm 1$ matrix A with at most two nonzero elements in each column is totally unimodular if and only if rows of A can be partitioned into two sets, red and blue, such that the sum of the red rows minus the sum of the blue rows is a vector whose entries are $0, \pm 1$ (admits row-bicoloring).*

Our $\mathbf{O}$ should be the column vector x, and constraints should be represented by a matrix A. In particular, we view $\mathbf{O}$ as a column vector, but still access the item by $\mathbf{O}_{ij}$.[14] The matrix $A \in \{0, \pm 1\}^{(m+(m+n))\times((n+m)(m+1))}$. $A_{:,ij}$ denotes the constraints involving $\mathbf{O}_{ij}$. The first m rows of A correspond to $\forall j < m, \sum_{i=0}^{n+m-1} \mathbf{O}_{ij} = 1$, and the remaining $m + n$ rows correspond to $\forall i, \sum_{j=0}^{m} \mathbf{O}_{ij} = 1$. Therefore, we have $\forall k < m, j < m, i, A_{k,ij} = \delta_{j,k}$ and $\forall k \ge m, j, i A_{k,ij} = \delta_{i,k-m}$, else $A_{k,ij} = 0$, where $\delta_{j,k} = [[j == k]]$. We have the linear constraints in standard form as $A\mathbf{O} = \mathbf{1}$.

Lemma 2. *The A defined above is totally unimodular.*

Proof. First, we show A admits row-bicoloring. We color the first m rows red, and remaining $n + m$ rows blue. The sum of red rows is: $R_{ij} = \sum_{k=0}^{m-1} A_{k,ij} = \sum_{k=0}^{m-1} \delta_{j,k} = [[j < m]]$ and the sum of blues is $B_{ij} = \sum_{k=m}^{2m+n-1} A_{k,ij} = \sum_{k=m}^{2m+n-1} \delta_{i,k-m} = 1$. Therefore, $R_{ij} - B_{ij} = [[j == m]] \in \{0, \pm 1\}$, and A admits a row-bicoloring. Since A has only $0, \pm 1$ value, and one variable in $\mathbf{O}$ at most participates in two constraints (incoming and outgoing), by Theorem 3, A is totally unimodular. □

Now, we prove Proposition 2.

Proof. We have A being totally unimodular. We have $c = \widetilde{W}$, $l = 0, u = 1$, by Theorem 2, the LP solutions contain an integer vector. Since the Gumbel distribution has a positive and differentiable density, by (Paulus et al., 2020, Proposition 3), $\arg\max_{\mathbf{O} \in \mathcal{O}} \langle \widetilde{\mathbf{W}}, \mathbf{O}\rangle$ yields a unique solution with probability 1. Clearly, this solution is the only integer solution in our LP solutions. Now, suppose another non-integer solution exists. We know the linear programming domain is the convex hull by Proposition 3. Clearly, another integer solution exists, which contradicts the uniqueness of the integer solution. Hence, the $\mathbf{O}(\widetilde{\mathbf{W}}, 0)$ yields a unique integer solution with probability 1. □

[14] Alternatively, one could have a vector x and $x_{i(m+1)+j} = \mathbf{O}_{ij}$. However, this will gets clumsy.