# Serving MPE Queries on Tensor Networks by Computing Derivatives

**Maurice Wenig** MAURICE.WENIG@UNI-JENA.DE
**Hanno Barschel** HANNO.BARSCHEL@UNI-JENA.DE
**Joachim Giesen** JOACHIM.GIESEN@UNI-JENA.DE
**Andreas Goral** ANDREAS.GORAL@UNI-JENA.DE
**Mark Blacher** MARK.BLACHER@UNI-JENA.DE
*Friedrich-Schiller-Universität Jena, Germany*

**Editors:** J.H.P. Kwisthout & S. Renooij

## Abstract

Recently, tensor networks have been proposed as a data structure for weighted model counting. Computing a weighted model count is thus reduced to contracting a factorized tensor expression. Inference queries on graphical models, especially PoE (probability of evidence) queries, can be expressed directly as weighted model counting problems. Maximization problems can also be addressed on the same data structure, only the standard sum-product semiring has to be replaced by either the tropical (max-sum) or the Viterbi (max-product) semiring in the computations, that is, the tensor contractions. However, tensor contractions only provide maximal values, but MPE (most probable explanation) queries on graphical models do not ask for the maximal value, but for a state, or even the states, at which the maximal value is attained. In the special case of tropical tensor networks for ground states of spin glasses, it has been observed that the ground state can be obtained by computing a derivative of the tensor network over the tropical semiring. Here, we generalize this observation, provide a generic algorithm for computing the derivatives, and prove its correctness.

**Keywords:** Graphical Models; Tensor Networks; Most Probable Explanation (MPE); Maximum Probability States; Tensor Derivatives.

## 1. Introduction

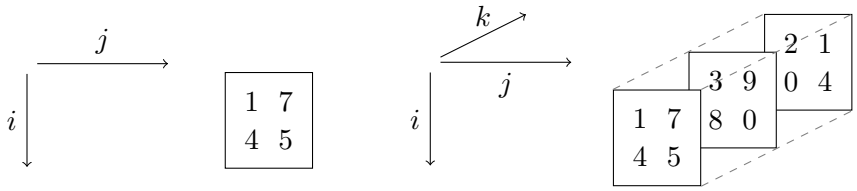
Standard inference queries on graphical models, that is, Bayesian networks or more general Markov random fields, include *probability of evidence* (PoE) and *most probable explanation* (MPE) queries (Darwiche, 2003). PoE queries can be formulated as weighted model counting problems (Chavira and Darwiche, 2008). Many state-of-the-art weighted model counting solvers, such as D4 by Lagniez and Marquis (2017) and MiniC2D by Oztok and Darwiche (2018), follow the knowledge compilation approach (Darwiche and Marquis, 2002) that translates any given problem instance, for example, a Bayesian network, into an arithmetic circuit on which inference queries can be served. Recently, tensor-network-based algorithms for weighted model counting have been shown to contribute to the state of the art by improving the virtual best solver on benchmark data sets (Dudek et al., 2019; Dudek and Vardi, 2020). However, weighted model counting only directly works for PoE queries. MPE queries can be served on arithmetic circuits by replacing sum nodes with max nodes, but require an additional downward pass on the circuit for collecting the maximizing states.

The two-pass approach for serving MPE queries has also been used by Darwiche (2003) for computing partial derivatives of maximizer circuits with respect to parameters and variables, which already shows a connection between serving MPE queries and computing derivatives. In tensor networks, PoE queries can be served with two contractions (one for each marginalized probability), but there is no known way of serving MPE queries, because replacing sums with maximizations, as it is done in arithmetic circuits, only yields the maximum probability, and not a maximizing state. However, Liu et al. (2021) have observed, in the special case of tropical tensor networks for ground states of spin glasses, that a maximizing state can be obtained by just differentiating the tensor expression which maximizes the probability. In this paper, we generalize this observation and make it rigorous by providing a proof. We also provide a proof-of-concept implementation of our approach at https://github.com/ti2-group/tensor_mpe.

## 2. Tensor Networks

So far, there is no established standard notation for tensors and tensor expressions. Therefore, we briefly introduce the notation that we are going to use in the following. Tensors are a generalization of vectors and matrices and can essentially be treated as multi-dimensional arrays. Let $[d]$ denote the discrete set $\{1, \dots, d\}$. Then, formally, a tensor with $n$ axes is a function $T : \bigtimes_{i=1}^{n} [d_i] \to R$ on a discrete domain to a set of entries, where $d_i \in \mathbb{N}$ are arbitrary axis sizes. We use $T_{ijk}$ to indicate the entry $T(i, j, k)$ and call $i$, $j$, and $k$ indices instead of function arguments. Here, we do not require $R$ to be a field, but only that it is a commutative semiring. This allows us to use operations such as the maximum, which can not be an operation of a field because it has no inverse elements. In the following, we only consider tensors with real entries. Therefore, we will only write $\mathbb{R}$ instead of $R$ from now on.

For example, a tensor with two axes $[m]$ and $[n]$ is a matrix $M \in \mathbb{R}^{m \times n}$, because it can also be seen as a function $M : [m] \times [n] \to \mathbb{R}$. We can also have tensors $T : [m] \times [n] \times [o] \to \mathbb{R}$ with more than two axes. The number of axes is called the *order* of the tensor. With this, a vector is a tensor of order one, a matrix is a tensor of order two, and a batch of matrices is a tensor of order three. A visualization of this is given in Figure 1.




(a) Matrix, with $M_{12} = 7$. (b) Tensor, with $T_{123} = 1$.

Figure 1: Axes of a matrix and axes of a third-order tensor.

We can combine multiple tensors with arithmetic operations, similar to how two matrices can be multiplied, in a *tensor network*.

**Definition 1** *A tensor network with axis sizes $d_1, \ldots, d_n \in \mathbb{N}$ is a set of tuples*

$$\mathbb{T} := \{(T^{(i)}, S^{(i)}) \mid i \in [m]\}$$

*with $m \in \mathbb{N}$, where each tuple consists of a tensor and an associated set of axes, i.e. $S^{(i)} \subseteq [n]$ and $T^{(i)} : \bigtimes_{j \in S^{(i)}} [d_j] \to \mathbb{R}$.*

Tensor networks are a very natural representation of graphical models. Actually, Robeva and Seigal (2017) have shown a duality between graphical models and tensor networks. The difference is that instead of functions over shared variables, tensor networks use tensors over shared axes, and instead of different values of a variable, tensor networks use indices along an axis. Next, we define how to operate on a tensor network. For this, we first need some notation.

**Definition 2 (Multi-Index)** *A multi-index of length $n$ is a tuple of $n$ indices.*

Multi-indices are a useful notion because it is often easier to treat the multiple indices, which are used to access tensor entries, as a single object. An example of that can be found in the following definition.

**Definition 3 (Projections)** *Let $x$ be some multi-index of length $n$, and let $S := \{s_1 < \ldots < s_m\} \subseteq [n]$ for some $m \leq n$. Then the projection of $x$ onto $S$ is a multi-index of length $m$, denoted as $x : S$ with*

$$(x : S)_i := x_{s_i}.$$

**Definition 4 (Domain of a Tensor Network)** *Let $\mathbb{T}$ be a tensor network with axis sizes $d_1, \ldots, d_n$. Then the domain of $\mathbb{T}$ is the set of all possible indices $\mathrm{dom}(\mathbb{T}) := \bigtimes_{i=1}^{n} [d_i]$. We also define the domain for a subset of axes $I \subseteq [n]$ as $\mathrm{dom}(\mathbb{T})|_I := \bigtimes_{i \in I} [d_i]$, and the domain where we fix a subset of axes $I \subseteq [n]$ to indices $x_I$ as $\mathrm{dom}(\mathbb{T})|_{I=x_I} := \{x \in \mathrm{dom}(\mathbb{T}) \mid x : I = x_I\}$.*

Note that $\mathrm{dom}(\mathbb{T})|_I$ contains all combinations of indices for the axes in $I$ and consists of multi-indices of length $|I| \leq n$, whereas $\mathrm{dom}(\mathbb{T})|_{I=x_I}$ contains all combinations of indices for the axes *not* in $I$ and consists of multi-indices of full length $n$ because they still include the fixed indices $x_I$.

The operation naturally associated with tensor networks is called *contraction*. It consists of two elementary operations, namely, an aggregation $\oplus$ and a combination $\odot$. These operations only need to define a commutative semiring on $R$. In the standard sum-product semiring $\mathbb{R}$, which is actually more than just a commutative semiring, namely, a field, we have $\oplus = +$ and $\odot = \cdot$.

**Definition 5 (Tensor Contraction)** *We define the complete contraction of a tensor network $\mathbb{T}$ as*

$$\kappa(\mathbb{T}) := \bigoplus_{x \in \mathrm{dom}(\mathbb{T})} \bigodot_{(T,S) \in \mathbb{T}} T_{x:S}.$$

Note that we aggregate over all axes for the complete contraction, which results in a scalar. There is also a more general contraction that does not always aggregate over all axes, but since we are not using it in the following, we leave it out for simplicity. Conceptually, it is helpful to also consider the combination of the tensors without the aggregation.

**Definition 6 (Tensor Combination)** *We define the combination of a tensor network $\mathbb{T}$ as an $n$-th order tensor $\pi(\mathbb{T})$ with*

$$\pi(\mathbb{T})_x := \bigodot_{(T,S)\in\mathbb{T}} T_{x:S}$$

*for $x \in \mathrm{dom}(\mathbb{T})$.*

We can use the commutative semiring operations to split up the aggregation in the complete contraction of a tensor network into multiple smaller contractions with tensors as intermediate results. For example, in a matrix chain multiplication $M_{il} = A_{ij} \cdot B_{jk} \cdot C_{kl}$, which is a special case of tensor network, we avoid computing the tensor combination $X_{ijkl}$ before contracting over $j$ and $k$, by first computing $Y_{ik} = A_{ij} \cdot B_{jk}$ as an intermediate result and then $M_{il} = Y_{ik} \cdot C_{kl}$. Alternatively, we could have first computed $Z_{jl} = B_{jk} \cdot C_{kl}$ as an intermediate results and then $M_{il} = A_{ij} \cdot Z_{jl}$. We call the choice of arranging the intermediate computations a *contraction path*. The choice of contraction path can make an exponential difference in performance (Gray and Kourtis, 2021). Finding an optimal contraction path for an arbitrarily connected tensor network is, however, an NP-hard problem (Lam et al., 1997). Still, in practice, with a good choice of contraction path, we can often efficiently marginalize and maximize in tensor networks, by using addition and maximization respectively as an aggregation operation.

For conditioning, we need another operation, namely, slicing. A slice of a tensor network considers only a subset of the full domain, where some axes have been set to fixed indices.

**Definition 7 (Slicing)** *Let $\mathbb{T}$ be a tensor network with axis sizes $d_1, \ldots, d_n$. Let $I \subseteq [n]$ and $x_I \in \mathrm{dom}(\mathbb{T})_I$, then we denote the slice of $\mathbb{T}$ where the axes $I$ have been set to the indices $x_I$ as $\mathbb{T}|_{I=x_I}$ with*

$$\mathrm{dom}(\mathbb{T}|_{I=x_I}) := \mathrm{dom}(\mathbb{T})|_{I=x_I}.$$

Because of the different domain, the operations on the sliced tensor network are different from the same operations on the unsliced tensor network. The contraction $\kappa(\mathbb{T}|_{I=x_I})$ no longer aggregates over the axes in $I$, and $\pi(\mathbb{T}|_{I=x_I})$ is an $(n - |I|)$-th order tensor with

$$\pi(\mathbb{T}|_{I=x_I})_{x_{-I}} = \pi(\mathbb{T})_{x_I x_{-I}}$$

for $x_I \in \mathrm{dom}(\mathbb{T})|_I$ and $x_{-I} \in \mathrm{dom}(\mathbb{T})|_{[n]\setminus I}$. We use $x_I x_{-I}$ to denote a complete multi-index of length $n$ with the indices from $x_I$ and $x_{-I}$ at the appropriate positions in the multi-index.

In practice, the contraction of a sliced tensor network is implemented as a contraction over slices of the contained tensors. These tensor slices can be interpreted as lower-order tensors themselves, and therefore the contraction over these slices is faster than over the full-order tensors.

## 3. Graphical Models as Tensor Networks

Tensor networks provide a natural representation of graphical models (Markov random fields). The operations of tensor contraction and tensor slicing naturally facilitate inference queries. We consider a graphical model with $n$ variables. Let $[d_1], \ldots, [d_n]$ be the domains of

the variables, let $\mathcal{C} \subseteq 2^{[n]}$ be the set of the factors of the graphical model, let $\mathcal{X}_C := \bigtimes_{c \in C}[d_c]$ be the domain of the variables in $C \in \mathcal{C}$, and let $\phi_C : \mathcal{X}_C \to \mathbb{R}_{\geq 0}$ be the interaction parameters for the factor $C \in \mathcal{C}$ such that

$$p(x) = \exp\left(\sum_{C \in \mathcal{C}} \phi_C(x) - Z\right)$$

for $x \in \mathcal{X}_{[n]}$, where $Z$ is the log-normalization constant. To put them into the context of tensor networks, we view the interaction parameters as $|C|$-th order tensors $T^{(C)}$ such that

$$T^{(C)}_x := \phi_C(x)$$

for $x \in \mathcal{X}_C$. Then, the graphical model is represented by the tensor network $\mathbb{T} := \{(T^{(C)}, C) \mid C \in \mathcal{C}\}$, and we can compute the (unnormalized) probability of the maximum probability state $\hat{x}$ with a contraction of this tensor network over the tropical semiring with the semiring operations $\oplus = \max$ and $\odot = +$ as

$$\log p(\hat{x}) = \kappa(\mathbb{T}).$$

We can even condition on variables with the use of slicing,

$$\log p(\hat{x} | I = x_I) = \kappa\left(\mathbb{T}|_{I=x_I}\right).$$

Our goal, however, is to compute the index $\hat{x}$, that is, a probability maximizing state. In the following, we show how this can be accomplished by computing the derivative of this tensor contraction over the tropical semiring.

## 4. Tensor Derivatives

In this section, we use $I$, $J$, and $K$ instead of $x_I$, $x_J$, and $x_K$ to denote multi-indices, to avoid confusion between tensor indices and function arguments, and to be more aligned with the notation of multivariable calculus.

For derivatives of operations on tensor networks, we use the standard notion of the Jacobian matrix, extended to the domain of tensors. It is helpful to view a tensor as a high-dimensional container for scalars, similar to a vector. Then the concepts already known from multivariable calculus can easily be transferred to tensors and serve as a useful tool to deal with functions that transform tensors into other tensors, such as the tensor combination in Definition 6.

**Definition 8 (Jacobian Tensor)** *Let $f$ be a function that maps an $n$-th order tensor $V$ with axis sizes $v_1, \ldots, v_n$ to an $m$-th order tensor $U = f(V)$ with axis sizes $u_1, \ldots, u_m$. Then the Jacobian tensor $\frac{dU}{dV}$ is an $(m+n)$-th order tensor with axis sizes $u_1, \ldots, u_m, v_1, \ldots, v_n$ with*

$$\left(\frac{dU}{dV}\right)_{IJ} := \frac{dU_I}{dV_J} = \frac{df(V)_I}{dV_J}$$

*for $I \in \mathrm{dom}(U)$ and $J \in \mathrm{dom}(V)$. We use $IJ := (i_1, \ldots, i_m, j_1, \ldots, j_n)$ to denote the concatenation of the multi-indices $I = (i_1, \ldots, i_m)$ and $J = (j_1, \ldots, j_n)$.*

Because we can represent the contraction as a composite function of combination and aggregation, we can compute the derivative of the contraction by the chain rule.

**Observation 1 (Chain Rule for Tensors)** *Let $g$ be a function that maps a tensor $W$ to a tensor $V = g(W)$ and let $f$ be a function that maps $V$ to a tensor $U = f(V) = f(g(W))$. Then, for $I \in \mathrm{dom}(U)$ and $K \in \mathrm{dom}(W)$,*

$$\left(\frac{d\,U}{d\,W}\right)_{IK} = \sum_{J \in \mathrm{dom}(V)} \left(\frac{d\,U}{d\,V}\right)_{IJ} \left(\frac{d\,V}{d\,W}\right)_{JK} = \sum_{J \in \mathrm{dom}(g(W))} \left(\frac{d\,f(g(W))}{d\,g(W)}\right)_{IJ} \left(\frac{d\,g(W)}{d\,W}\right)_{JK}.$$

Let us start by defining the derivative of the aggregation operation ($\oplus = \max$) over the tropical semiring. The max operator is not differentiable at the standard definition of differentiability in every point. It is non-differentiable when both inputs are equal. Therefore, we extend the definition of a derivative for the maximum operation.

**Definition 9 (Extended Derivative of the Bivariate Maximum)** *Let $a, b \in \mathbb{R}$. Then we define the extended derivative of the maximum as*

$$\frac{d\,\max(a,b)}{d\,a} := \mathbb{1}\left[a \geq b\right], \qquad \frac{d\,\max(a,b)}{d\,b} := \mathbb{1}\left[b \geq a\right].$$

This sets the extended derivative at the point of non-differentiability, $a = b$, to $1 \in \mathbb{R}$ for both the derivative with respect to $a$ and with respect to $b$. The reasoning behind this choice is, that we want to use the derivative with respect to an input as a measure of how much this input contributes to the output, and in the case where $a = b$, both $a$ and $b$ contribute equally to the output.

The extended derivative of the bivariate maximum function can be generalized to an extended derivative of a maximum over any number of inputs. In the following lemma, we maximize over outputs of a function $f(x)$ instead of elements of a set $x \in M$, to clarify that there can be multiple inputs $x_1 \neq x_2$ with the same value $f(x_1) = f(x_2)$.

**Lemma 10 (Extended Derivative of the Maximum)** *Let $f : M \to \mathbb{R}$ and let $\hat{y} := \max_{x \in M} f(x)$. Then for $x \in M$ the extended derivative of the maximum at $f(x)$ is given as*

$$\frac{d\,\hat{y}}{d\,f(x)} = \mathbb{1}\left[f(x) = \hat{y}\right].$$

**Proof** Let $M := \{x_1, \ldots, x_m\}$, let $y_i := f(x_i)$ and let $z_i := \max(y_i, \max(y_{i+1}, \max(\ldots)))$ for $i \in [m]$. Then $\hat{y} = z_1$ and with the chain rule we get the following factorization,

$$\frac{d\,\hat{y}}{d\,y_i} = \frac{d\,z_1}{d\,z_2} \cdots \frac{d\,z_{i-1}}{d\,z_i} \frac{d\,z_i}{d\,y_i} = \mathbb{1}\left[y_1 \leq z_2\right] \cdot \ldots \cdot \mathbb{1}\left[y_{i-1} \leq z_i\right] \cdot \mathbb{1}\left[y_i \geq z_{i+1}\right].$$

At least one factor in this factorization is zero if, and only if $y_i$ is not the maximum value. If $\hat{y} = y_i$, then $z_1 = z_2 = \ldots = z_i = \hat{y}$ and $y_i \geq z_{i+1}$, therefore $\frac{d\,\hat{y}}{d\,y_i} = 1$. Otherwise, if $\hat{y} \neq y_i$, then let $k$ be the greatest index such that $y_k > y_i$. If

(a) $k > i$, then $y_i < z_{i+1}$, and thus $\mathbb{1}\left[y_i \geq z_{i+1}\right]$ evaluates to zero,

(b) $k < i$, then $y_k > z_{k+1}$ by the definition of $k$, and thus $\mathbb{1}[y_k \leq z_{k+1}]$ evaluates to zero.

Therefore, if $\hat{y} \neq y_i$, then $\frac{d\,\hat{y}}{d\,y_i} = 0$. ■

With the derivative extension of the maximum, we can work out the derivative of the aggregation operation over the tropical semiring that is used in tensor contractions.

**Lemma 11 (Jacobian Tensor of Aggregations)** *Let $C$ be an $n$-th order tensor with a unique maximum entry. Let $A := \max_{I \in \mathrm{dom}(C)} C_I$, then $\frac{d\,A}{d\,C}$ is a tensor with only one non-zero entry,*

$$\left(\frac{d\,A}{d\,C}\right)_I = \mathbb{1}\left[I = \operatorname*{argmax}_{J \in \mathrm{dom}(C)} C_J\right]$$

*for $I \in \mathrm{dom}(C)$.*

**Proof** Because the maximum entry of $C$ is unique, this follows immediately from the definition of the Jacobian tensor (Definition 8) and the extended derivative of the maximum (Lemma 10). ■

In practice, we can not assume that the maximum is unique, but we can adapt the definition of the extended derivative to also cover this case. We discuss this in detail in Section 5.3.

With the derivative of the aggregation done, let us now consider the derivative of the tensor combination over the tropical semiring with combination operation $\odot = +$.

**Lemma 12 (Jacobian Tensor of Combinations)** *Let $\mathbb{T}$ be a tensor network with $n$ axes, and let $(T, S) \in \mathbb{T}$ be a tensor in the tensor network, with its corresponding set of axes, that has order $m := |S|$. Then, over the tropical semiring, the derivative of the combination $\pi(\mathbb{T})$ with respect to a tensor $T$ is an $(n+m)$-order tensor that has only zero-entries, except for one non-zero entry $\left(\frac{d\,\pi(\mathbb{T})}{d\,T}\right)_{IJ} = 1$ where $T_J$ contributes to the sum that computes the scalar value $\pi(\mathbb{T})_I$:*

$$\left(\frac{d\,\pi(\mathbb{T})}{d\,T}\right)_{IJ} = \mathbb{1}[I : S = J]$$

*for $I \in \mathrm{dom}(\mathbb{T})$ and $J \in \mathrm{dom}(T)$.*

**Proof** The combination $\pi(\mathbb{T})_I$ over the tropical semiring is a sum over the tensor entries at the corresponding indices in $I$. Because $T_J$ has no influence on the summands from the other tensors, their derivative with respect to $T_J$ is zero, that is,

$$\left(\frac{d\,\pi(\mathbb{T})}{d\,T}\right)_{IJ} = \frac{d}{d\,T_J} \sum_{(T', S') \in \mathbb{T}} T'_{I:S'} = \frac{d}{d\,T_J} T_{I:S} = \mathbb{1}[I : S = J]\,.$$

■

Now, we can combine the derivative of the aggregation operation and the derivative of the combination operation into the derivative of the complete contraction by using the chain rule and additional vectors in the original tensor network.

**Theorem 13 (Differentiation with Respect to an Axis)** *Let $\mathbb{T}$ be a tensor network with axis sizes $d_1, \ldots, d_n$ such that the combination $\pi(\mathbb{T})$ over the tropical semiring has a unique non-zero maximal entry. For $i \in [n]$, let $\tilde{T}^{(i)} := \mathbf{0}_{d_i}$ be a zero-vector of size $d_i$. Let $\tilde{\mathbb{T}} := \mathbb{T} \cup \{(\tilde{T}^{(i)}, \{i\}) \mid i \in [n]\}$ be the original tensor network augmented by the vectors $\tilde{T}^{(i)}$. Then, we have over the tropical semiring that*

1. *$\kappa(\tilde{\mathbb{T}}) = \kappa(\mathbb{T})$, and*
2. *for all $i \in [n]$, the Jacobian tensor $\frac{d\,\kappa(\tilde{\mathbb{T}})}{d\,\tilde{T}^{(i)}}$ is a vector of size $d_i$ with only one non-zero entry at the index $\hat{x}_i$, where $\hat{x} := \operatorname*{argmax}_{I \in \mathrm{dom}(\mathbb{T})} \pi(\mathbb{T})_I$.*

**Proof** 1. We assume that all axes of the tensor network are used by at least one of its tensors. If this is not the case, axes that are not used can be removed. With this, the addition of $\{(\tilde{T}^{(i)}, \{i\}) \mid i \in [n]\}$ does not introduce any new axes. Therefore $\pi(\tilde{\mathbb{T}})$ has the same shape as $\pi(\mathbb{T})$. Furthermore, because $0 \in \mathbb{R}$ is the neutral element of the addition in $\mathbb{R}$, which is the combination operation $\odot = +$ in the tropical semiring, we have $\pi(\tilde{\mathbb{T}}) = \pi(\mathbb{T})$ and therefore $\kappa(\tilde{\mathbb{T}}) = \kappa(\mathbb{T})$.

2. It follows that, for all $i \in [n]$ and $k \in [d_i]$,

$$\begin{aligned}
\left(\frac{d\,\kappa(\tilde{\mathbb{T}})}{d\,\tilde{T}^{(i)}}\right)_k &= \sum_{J \in \mathrm{dom}(\tilde{\mathbb{T}})} \left(\frac{d\,\kappa(\tilde{\mathbb{T}})}{d\,\pi(\tilde{\mathbb{T}})}\right)_J \left(\frac{d\,\pi(\tilde{\mathbb{T}})}{d\,\tilde{T}^{(i)}}\right)_{Jk} && \text{chain rule} \\
&= \sum_{J \in \mathrm{dom}(\mathbb{T})} \left(\frac{d\,\kappa(\mathbb{T})}{d\,\pi(\mathbb{T})}\right)_J \left(\frac{d\,\pi(\tilde{\mathbb{T}})}{d\,\tilde{T}^{(i)}}\right)_{Jk} && 1. \\
&= \sum_{J \in \mathrm{dom}(\mathbb{T})} \left(\frac{d\,\kappa(\mathbb{T})}{d\,\pi(\mathbb{T})}\right)_J \mathbb{1}\left[J : \{i\} = k\right]. && \text{combination}
\end{aligned}$$

Because of Lemma 11, the derivative of the aggregation by the combination $\frac{d\,\kappa(\mathbb{T})}{d\,\pi(\mathbb{T})_J}$ evaluates to one if $J$ is the argmax of $\pi(\mathbb{T})$ and zero otherwise. Since $\hat{x} = \operatorname*{argmax}_{I \in \mathrm{dom}(\mathbb{T})} \pi(\mathbb{T})_I$ by definition, it follows that, for all $k \in [d_i]$,

$$\begin{aligned}
\left(\frac{d\,\kappa(\tilde{\mathbb{T}})}{d\,\tilde{T}^{(i)}}\right)_k &= \sum_{J \in \mathrm{dom}(\mathbb{T})} \left(\frac{d\,\kappa(\mathbb{T})}{d\,\pi(\mathbb{T})}\right)_J \mathbb{1}\left[J : \{i\} = k\right] \\
&= \sum_{J \in \mathrm{dom}(\mathbb{T})} \mathbb{1}\left[J = \hat{x}\right] \mathbb{1}\left[J : \{i\} = k\right] \\
&= \mathbb{1}\left[\hat{x}_i = k\right],
\end{aligned}$$

which means that $\frac{d\,\kappa(\tilde{\mathbb{T}})}{d\,\tilde{T}^{(i)}}$ has only one non-zero entry at the index $\hat{x}_i$. ■

## 5. The Algorithm

Theorem 13 suggests an algorithm for serving MPE queries on graphical models that is represented by a tensor network over the tropical semiring. First, construct the augmented tensor network $\tilde{\mathbb{T}}$ by adding the zero-vectors $\tilde{T}^{(i)}$ to the tensor network $\mathbb{T}$ that represents the graphical model. Then, compute the derivative of the $\kappa(\tilde{\mathbb{T}})$, which computes the maximum log probability value, with respect to the $i$-th axis, that is, compute $d\,\kappa(\tilde{\mathbb{T}})/d\,\tilde{T}^{(i)}$. The single non-zero entry in the derivative gives the $i$-th index in the multi-index that represents the argmax. In practice, the derivatives can be computed by automatic differentiation (Griewank and Walther, 2008) of the contraction $\kappa(\tilde{\mathbb{T}})$.

### 5.1. Link to the Log Partition Function

With a contraction over the standard sum-product semiring, we can compute the log partition function. That is, by switching the semiring from the sum-product semiring to the tropical semiring, we switch from computing the log normalizing constant to computing the maximum probability value of any state. The contraction $\kappa(\tilde{\mathbb{T}})$ can thus be computed by replacing the sum operations in the log partition function with max operations, that is,

$$\underbrace{\log \sum_{x \in \mathcal{X}} \prod_{C \in \mathcal{C}} \exp\left(\phi_C(x)\right)}_{Z} \quad \overset{\text{switch semiring}}{\longrightarrow} \quad \log \max_{x \in \mathcal{X}} \prod_{C \in \mathcal{C}} \exp\left(\phi_C(x)\right)$$

$$= \underbrace{\max_{x \in \mathcal{X}} \sum_{C \in \mathcal{C}} \phi_C(x)}_{\kappa(\mathbb{T})} .$$

A well-known observation is that computing the derivative of the log partition function over the standard semiring gives the expected value of the sufficient statistics $(\phi_c)_{C \in \mathcal{C}}$. That is, we have a correspondence between the derivatives of the log partition function over the standard sum-product and over the tropical semiring, which is illustrated in Figure 2.

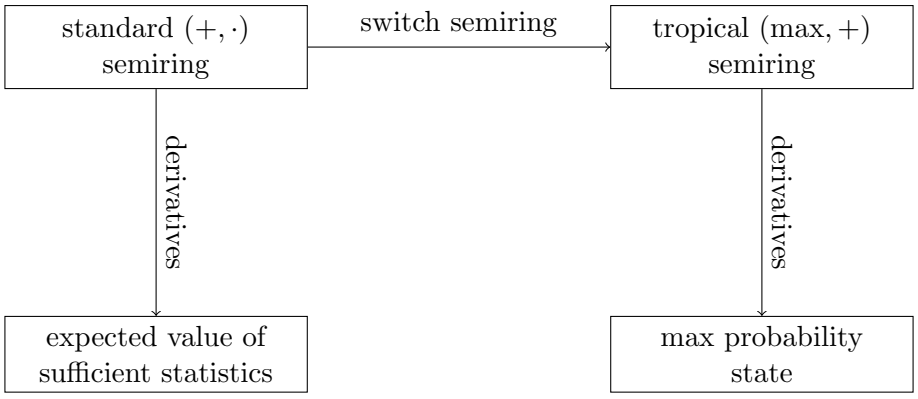



Figure 2: What the derivatives of the log partition function yield in different semirings.

### 5.2. Computational Complexity

Computing the log partition function over either semiring needs the same number of semiring operations, that is, the complexity is asymptotically the same if we assume that all semiring operations can be computed in constant time. Computing the derivatives of the log partition function over either semiring can be accomplished by reverse-mode automatic differentiation within a constant factor in the time needed to compute the contraction $\kappa(\tilde{\mathbb{T}})$ itself (Griewank and Walther, 2008, Sections 4.5).

For the memory overhead of computing the derivative of the contraction over the tropical semiring, we have to consider that the complete contraction, performed along a contraction path, is transformed into a series of tropical batch-matrix-multiplications. The memory overhead for the derivatives of the contraction path is then composed of the memory overhead for the derivatives of these batch-matrix-multiplications. Consider the following batch-matrix-multiplication:

$$C_{sij} = \max_k \left(A_{sik} + B_{skj}\right).$$

To see how much memory we need to compute the derivative of this batch-matrix-multiplication, let $\hat{k}_{sij}$ be the index at which $C_{sij} = A_{si\hat{k}_{sij}} + B_{sj\hat{k}_{sij}}$ is maximized, then the derivative is given as the following sixth-order tensor

$$\left(\frac{d\,C}{d\,A}\right)_{sijs'i'k} = \frac{d\,C_{sij}}{d\,A_{s'i'k}} = \mathbb{1}\left[s = s' \wedge i = i' \wedge k = \hat{k}_{sij}\right],$$

because the entry $A_{s'i'k}$ can only contribute to $C_{sik}$, if it is used in the maximization which defines $C_{sik}$, which means that $s' = s$ and $i' = i$. Furthermore, the specific entry $A_{sik}$ is only used if $k$ is the maximizing index. This means that we only have to store $\hat{k}_{sij}$ for every $s$, $i$, and $j$. This requires the same memory as is needed for storing $C$ itself. Therefore, the extra memory needed for computing the derivative is almost the same as the memory needed for computing the maximum value itself. The key differences between the computation of the maximal value itself and the computation of the derivative, that affect the memory requirements, are the stored data types, namely floating point numbers for computing the maximum value and integers for computing the maximum state, and that when computing the maximal value we can deallocate intermediate tensors that are no longer needed. The latter should not amount to large differences, because good contraction paths usually have only a small number of large intermediate tensors. However, if memory is a limiting constraint, then we can trade the memory for runtime by using checkpointing in automatic differentiation (Griewank and Walther, 2000).

### 5.3. Multiple Maxima

In practice, we can not assume that there is only one maximum probability state. This leads to the problem that the derivative of the aggregation might have more than one non-zero entry. Therefore the derivative of the contraction might have multiple non-zero entries as well. This can however easily be dealt with, by defining an alternative extended derivative of the maximum function, which guarantees the existence of exactly one non-zero entry in the Jacobian tensor of an aggregation, even if the maximum is not unique. One possible

strategy to achieve this is to always choose the smallest maximizing index as the unique non-zero entry and to set the extended derivative at all the other maximizing indices to zero. Note that this strategy does not necessarily compute the same maximizing state for different contraction paths, because the computed maximizing state now depends on the order in which the axes have been aggregated. For example, consider a graphical model over two variables with maximum probability states $a = (1, 2)$ and $b = (2, 1)$. In this case, with forward mode automatic differentiation, if the first axis is aggregated first, then $a$ will be computed, otherwise $b$ will be computed.

### 5.4. Implementation

We have implemented the algorithm for serving MPE queries on tensor networks as a proof of concept using PyTorch (Ansel et al., 2024) and its autograd features. For this, we implemented our own autograd function for a batch-matrix-multiplication over the tropical semiring. To test the practical validity of the approach, we used our implementation on randomly generated graphical models in the form of tensor trains (see Figure 3). The corresponding runtimes of the implementation on these tensor trains are shown in Figure 4.

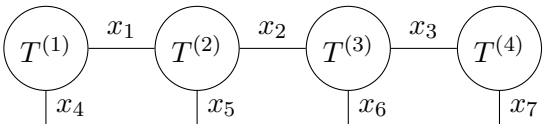



Figure 3: A tensor train graphical model with four factors.

Our proof of concept implementation is available in our repository at https://github.com/ti2-group/tensor_mpe. The implementations can be improved by using better contraction paths, harnessing the parallel processing power of graphical processing units (GPUs) for batch-matrix-multiplications over the tropical semiring, or by avoiding contracting over all elements by using a conditional variant of slicing that mimics branch-and-bound algorithms.

## 6. Conclusion

For tensor network representations of graphical models, we have explored the connection between derivatives of tensor network contractions over the tropical semiring and MPE (most probable explanation) states, that is, states of maximum probability. The connection, which is in the spirit of Darwiche's differential approach to inference in Bayesian networks, is reminiscent of the connection of derivatives of a graphical model's log partition function over the standard sum-product semiring and the expected value of its sufficient statistics. Over the tropical semiring, the connection directly leads to an algorithm for serving MPE queries on graphical models that are represented by tensor networks. The algorithm uses reverse-mode automatic differentiation to compute the derivatives of a complete tensor network contraction over the tropical semiring. By standard results from automatic differentiation, the time complexity of the algorithm is asymptotically the same as the corresponding com-

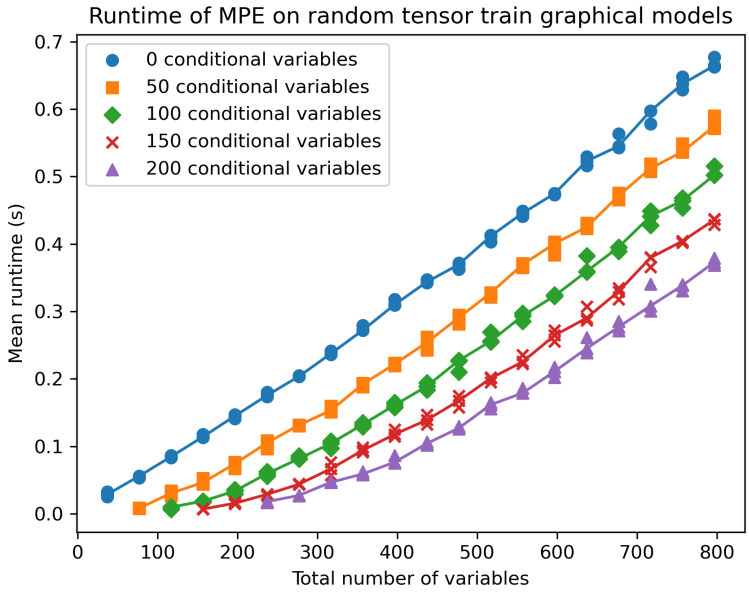



Figure 4: Runtimes of the implementation of our approach on tensor trains with 64 possible values for each variable, varying total number of variables, and varying number of conditional variables. The model was evaluated on three different queries for each configuration of total and conditional variables, and the median times are connected by lines. Each query was run ten times.

plexity for computing the log partition function with a tensor network contraction. The space complexity, however, depends on the size of the largest intermediate tensor and thus can be larger than the corresponding complexity for computing the log partition function.

## Acknowledgments

This work was supported by the Carl Zeiss Stiftung within the project "Interactive Inference".

## References

J. Ansel, E. Z. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. K. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, S. Zhang, M. Suo, P. Tillet, X. Zhao, E. Wang, K. Zhou, R. Zou, X. Wang, A. Mathews, W. Wen, G. Chanan, P. Wu, and S. Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the ACM International Confer-*

*ence on Architectural Support for Programming Languages and Operating Systems*, pages 929–947, 2024.

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

A. Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM*, 50(3):280–305, May 2003. ISSN 1557-735X. doi: 10.1145/765568.765570.

A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, Sept. 2002. ISSN 1076-9757.

J. M. Dudek and M. Y. Vardi. Parallel weighted model counting with tensor networks. *CoRR*, abs/2006.15512, 2020. URL https://arxiv.org/abs/2006.15512.

J. M. Dudek, L. Dueñas-Osorio, and M. Y. Vardi. Efficient contraction of large tensor networks for weighted model counting through graph decompositions. *CoRR*, abs/1908.04381, 2019. URL http://arxiv.org/abs/1908.04381.

J. Gray and S. Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, 2021.

A. Griewank and A. Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45, Mar. 2000. ISSN 1557-7295. doi: 10.1145/347837.347846.

A. Griewank and A. Walther. *Evaluating derivatives*. SIAM, Philadelphia, 2. ed. edition, 2008. ISBN 9780898716597. Includes bibliographical references and index.

J. Lagniez and P. Marquis. An Improved Decision-DNNF Compiler. In *Proceedings of the Joint Conference on Artificial Intelligence (IJCAI)*, pages 667–673, 2017.

C. Lam, P. Sadayappan, and R. Wenger. On Optimizing a Class of Multi-Dimensional Loops with Reductions for Parallel Execution. *Parallel Processing Letters*, 7(2):157–168, 1997.

J.-G. Liu, L. Wang, and P. Zhang. Tropical tensor network for ground states of spin glasses. *Physical Review Letters*, 126(9):090506, Mar. 2021. ISSN 1079-7114. doi: 10.1103/physrevlett.126.090506.

U. Oztok and A. Darwiche. An Exhaustive DPLL Algorithm for Model Counting. *Journal of Artificial Intelligence Research*, 62:1–32, 2018.

E. Robeva and A. Seigal. Duality of graphical models and tensor networks. *CoRR*, abs/1710.01437, 2017. URL http://arxiv.org/abs/1710.01437.