## Fast Weighted Model Sampling for $\mathrm{UFO}^2$ with Tree and Cardinality Constraints

Though proved domain-lifted, Algorithm 2 would needs too many WFOMC calls and thus is not practical. In this section, we present a more efficient WMS for $\mathbf{UFO}^2$ with tree and cardinality constraints.

The inefficiency of Algorithm 2 mainly roots in the following problems.

- When sampling a U-types assignment, one needs to compute the count distribution of a large number of predicates $\Psi = \{\xi_1, \ldots, \xi_{2^m}\}$, which needs to be evaluated on a set of size of the order $O(n^{2^m})$.
- When sampling each $\rho_{i,j}$, Algorithm 2 always computes WFOMC from scratch, while these WFOMCs may share some terms that can be cached.
- Computing WFOMC's needs a huge number of WMCs, which may be redundant.

Next, we will show how to address these problems respectively.

### UnaryTypes

We provide a fast algorithm to realise $\mathsf{UnaryTypes}$. The intuition is that in the case of $\mathbf{UFO}^2$ with tree and cardinality constraints, computing the WFOMC of $\Gamma \wedge \Sigma_{T_u} \wedge \mathcal{C}$ for any U-types assignment $T_u$ is domain-lifted from Section **??**. More specifically, when the processed index pairs $\Omega' = \varnothing$, $\mathsf{WFOMC}(\Gamma \wedge \sigma \wedge \mathcal{C}, n, w, \bar{w})$ is exactly $\mathsf{WFOMC}(\Gamma \wedge \Sigma_{T_u} \wedge \mathcal{C}, n, w, \bar{w})$.

Fix a cardinality vector of U-types $\mathbf{n}$, there are $\binom{n}{n_1, n_2, \ldots, n_{2^m}}$ possible U-types assignments, and as discussed in Section **??**, all these assignments share the same probability. Thus, the probability of sampling $\mathbf{n}$ is proportional to

$$\binom{n}{n_1, \ldots, n_{2^m}} \cdot \mathsf{WFOMC}(\Gamma \wedge \Sigma_{T_u} \wedge \mathcal{C}, n, w, \bar{w}) \quad (1)$$

where $T_u$ can be any U-types assignment that admits the cardinality vector $\mathbf{n}$ of U-types. Here we provide a simple assignment that follows the order of domain: for all $i \in [n]$, set the U-type of element $c_i$ to $\tau^{(j)}$, where $j$ is the minimal index such that $\sum_{k=1}^{j} n_k \geqslant i$.

### Caching for WFOMC

Suppose that $\mathcal{P}_\Gamma$ contains $P_1, \ldots, P_m$. From the proof of Theorem 1, the WFOMC of $\Gamma \wedge \sigma$ with constraint $\mathcal{C}$ derived from combining a tree constraint $\mathsf{T}_R$ with a cardinality constraint $\mathcal{C}_f$ can be written as

$$\begin{aligned}\mathsf{WFOMC}(\Gamma \wedge \sigma \wedge \mathcal{C}, n, w, \bar{w}) &= \mathsf{WFOMC}(\Gamma \wedge \sigma \wedge \mathsf{T}_R, n, w, \bar{w}) \cdot \\ &\sum_{\mathbf{n} \in \mathcal{D}} f(\mathbf{n}) \cdot q_{\Gamma \wedge \sigma \wedge \mathsf{T}_R, \mathcal{P}_\Gamma}(\mathbf{n})\end{aligned} \quad (2)$$

where $\mathcal{D} = \times_{t=1}^{m} \{1, 2, \ldots, n^{arity(P_t)}\}$. Let $\mathbf{M} = [n^{arity(P_1)} + 1, \ldots, n^{arity(P_m)} + 1]$. Applying DFT and inverse DFT on $q_{\Gamma \wedge \sigma \wedge \mathsf{T}_R, \mathcal{P}_\Gamma}(\mathbf{n})$, we have

$$q_{\Gamma \wedge \sigma \wedge \mathsf{T}_R, \mathcal{P}_\Gamma}(\mathbf{n}) = \frac{\sum_{\mathbf{k} \in \mathcal{D}} \mathsf{WFOMC}(\Gamma \wedge \sigma \wedge \mathsf{T}_R, n, w_{\mathbf{k}}, \bar{w}_{\mathbf{k}}) \cdot h(\mathbf{n}, \mathbf{k})}{\mathsf{WFOMC}(\Gamma \wedge \sigma \wedge \mathsf{T}_R, n, w, \bar{w}) \cdot \prod_{t=1}^{m} \mathbf{M}_t}, \quad (3)$$

where for all $t \in [m]$, $w_{\mathbf{k}}(P_t) = w(P_t) \cdot e^{-i2\pi \mathbf{k}_t / \mathbf{M}_t}$ and $\bar{w}_{\mathbf{k}}(P_t) = \bar{w}(P_t)$, and $h(\mathbf{n}, \mathbf{k}) = \exp(i2\pi \langle \mathbf{n}, \mathbf{k}/\mathbf{M} \rangle)$. Note here the notation $i$ denotes the imaginary number. Plugging (3) into (2), we obtain the full decomposition of $\mathsf{WFOMC}(\Gamma \wedge \sigma \wedge \mathcal{C}, n, w, \bar{w})$:

$$\begin{aligned}&\mathsf{WFOMC}(\Gamma \wedge \sigma \wedge \mathcal{C}, n, w, \bar{w}) = \\ &\frac{1}{\prod_{t \in [m]} \mathbf{M}_t} \cdot \sum_{\mathbf{n} \in \mathcal{D}} \sum_{\mathbf{k} \in \mathcal{D}} f(\mathbf{n}) \cdot h(\mathbf{n}, \mathbf{k}) \cdot \prod_{i \in [n]} w_i(\mathbf{k}) \cdot \\ &\prod_{\{i,j\} \in \Omega'} \hat{r}_{i,j}(\mathbf{k}) \cdot \prod_{\{i,j\} \in \mathcal{F}} r^+_{i,j}(\mathbf{k}) \cdot \prod_{\{i,j\} \in \Omega \setminus \Omega' \setminus \mathcal{F}} r^-_{i,j}(\mathbf{k}) \cdot \\ &\mathsf{TS}(A(\mathbf{k}), \mathcal{F}_\sigma \cup \mathcal{F}),\end{aligned} \quad (4)$$

where

$$\begin{aligned}w_i(\mathbf{k}) &= \mathsf{WMC}(\tau_i, w_{\mathbf{k}}, \bar{w}_{\mathbf{k}}), \\ \hat{r}_{i,j}(\mathbf{k}) &= \mathsf{WMC}(\psi'(c_i, c_j) \wedge \rho_{i,j}, w_{\mathbf{k}}, \bar{w}_{\mathbf{k}}), \\ r^+_{i,j}(\mathbf{k}) &= \mathsf{WMC}(\psi'(c_i, c_j) \wedge R(c_i, c_j), w_{\mathbf{k}}, \bar{w}_{\mathbf{k}}), \\ r^-_{i,j}(\mathbf{k}) &= \mathsf{WMC}(\psi'(c_i, c_j) \wedge \neg R(c_i, c_j), w_{\mathbf{k}}, \bar{w}_{\mathbf{k}}),\end{aligned}$$

and $A(\mathbf{k})$ is defined based on $r^+_{i,j}(\mathbf{k})$ and $r^-_{i,j}(\mathbf{k})$ accordingly.

Denote $\mathbf{k}_i$ the $i$th item of $\mathcal{D}$. We first compute and cache all terms of $f(\mathbf{n}) \cdot h(\mathbf{n}, \mathbf{k})$ as a $|\mathcal{D}| \times |\mathcal{D}|$-dimensional matrix $F$, in which the item at position $(i, j)$ is $F[i, j] = f(\mathbf{k}_i) \cdot h(\mathbf{k}_i, \mathbf{k}_j)$. When Algorithm 2 goes over all index pairs in $\Omega$, some terms in (4) can also be cached to avoid recomputing.

- We precompute and cache a vector $CW$ with the $i$th item $CW[i] = \prod_{j \in [n]} w_j(\mathbf{k}_i)$.
- We maintain a variable vector $Q$ with the $i$th item $Q[i] = \prod_{\{s,t\} \in \Omega'} \hat{r}_{s,t}(\mathbf{k}_i)$, and update it once a new index pair is added into $\Omega'$;
- For every $\Omega'$, we precompute and cache a vector $J_{\Omega'}$ with the $i$th item $J_{\Omega'}[i] = \prod_{\{s,t\} \in \mathcal{F}} r^+_{s,t}(\mathbf{k}_i) \cdot \prod_{\{s,t\} \in \Omega \setminus \Omega' \setminus \mathcal{F}} r^-_{s,t}(\mathbf{k}_i)$. Note that we can compute all $J_{\Omega'}$'s in an iterative manner:

$$J_{\Omega'}[i] = \begin{cases} J_{\Omega' \setminus \{s,t\}}[i] \cdot r^+_{s,t}(\mathbf{k}_i) & \text{if } \{s,t\} \in \mathcal{F} \\ J_{\Omega' \setminus \{s,t\}}[i] \cdot r^-_{s,t}(\mathbf{k}_i) & \text{if } \{s,t\} \notin \mathcal{F} \end{cases} \quad (5)$$

Equipped with the cached terms, we can write (4) as

$$\mathsf{WFOMC}(\Gamma \wedge \sigma \wedge \mathcal{C}, n, w, \bar{w}) = \frac{F \otimes (CW \odot Q \odot J_{\Omega'} \odot T_{\Omega'})}{\prod_{t \in [m]} \mathbf{M}_t},$$

where $T_{\Omega'}$ denotes the vector

$$[\mathsf{TS}(A(\mathbf{k}_1), \mathcal{F}_\sigma \cup \mathcal{F}), \ldots, \mathsf{TS}(A(\mathbf{k}_{|\mathcal{D}|}), \mathcal{F}_\sigma \cup \mathcal{F})], \quad (6)$$

$\otimes$ is the dot product and $\odot$ is the element-wise multiplication. Note all operations in the equation above can be efficiently implemented by scientific computing packages such as numpy[1].

[1] https://numpy.org/

## 0.1 Caching for WMCs

In this subsection, we will show how to reduce the redundancy of the WMCs of $w_i(\mathbf{k}), \hat{r}_{i,j}(\mathbf{k}), r^+_{i,j}(\mathbf{k})$ and $r^-_{i,j}(\mathbf{k})$.

Recall that $\tau^{(s)}$ is the $s$th U-type in $\mathcal{U}$. For every $s \in [2^m]$ and $\mathbf{k} \in \mathcal{D}$, we denote $w^{(s)}(\mathbf{k}) = \mathsf{WMC}(\tau^{(s)}(a), w, \bar{w})$. Then the value of $w_i(\mathbf{k})$ is exactly $w^{(s)}(\mathbf{k})$, where $s$ is the index of $\tau_i$ in $\mathcal{U}$. Thus we can precompute and cache $w^{(s)}(\mathbf{k})$ for every $s \in [2^m]$ and $\mathbf{k} \in \mathcal{D}$.

The value $r^+_{i,j}(\mathbf{k})$ and $r^-_{i,j}(\mathbf{k})$ can be cached in a similar way. Take $r^+_{i,j}(\mathbf{k})$ for an example. For every pair $\{s,t\} \in [2^m] \times [2^m]$ and $\mathbf{k} \in \mathcal{D}$, we precompute and cache

$$r^{(s,t)+}(\mathbf{k}) = \mathsf{WMC}(\psi_{s,t}(a,b) \wedge R(a,b), w_{\mathbf{k}}, \bar{w}_{\mathbf{k}}) \quad (7)$$

where $\psi_{s,t}(a,b)$ is the formula $\psi(a,b) \wedge \psi(b,a)$, in which all determined atoms have been replaced by true or false, according to $\tau^{(s)}(a)$ and $\tau^{(t)}(b)$. Then the value of $r^+_{i,j}(\mathbf{k})$ can be obtained from the cached $r^{(s,t)+}(\mathbf{k})$, where $s$ and $t$ is the index of U-types $\tau_i$ and $\tau_j$ in $\mathcal{U}$ respectively.

The value of all $\hat{r}_{i,j}(\mathbf{k})$'s can be also cached by computing and store $\hat{r}^{(s,t,u)}(\mathbf{k}) = \mathsf{WMC}(\psi_{s,t}(a,b) \wedge \rho^{(u)}(a,b), w_{\mathbf{k}}, \bar{w}_{\mathbf{k}})$ where $\rho^{(u)}$ is the $u$th B-type in $\mathcal{B}$.

We note that the number of all computed and cached items above , which is of the order $O(|\mathcal{D}|)$, could be much smaller than the number of $w_i(\mathbf{k}), \hat{r}_{i,j}(\mathbf{k}), r^+_{i,j}(\mathbf{k})$ and $r^-_{i,j}(\mathbf{k})$ for the large domain.

## Algorithm

We present the final algorithm in Algorithm 1. In lines 1-10, we compute and cache all terms, including the value of WMCs, the count distribution and the matrix $F$, that is independent of the latter U-type assignment. When sampling multiple models for a given sentence with constraints, we execute these lines only once, and repeatedly invoke the following code to generate the models. The function $\mathsf{RandomAssign}(\mathbf{n}, \Delta)$ randomly assigns U-types for all elements in $\Delta$ according to the sampled U-type cardinality $\mathbf{n}$. $\mathsf{SampleDist}(q)$ samples a vector from the distribution $q$. $\mathsf{Reverse}(\Omega)$ returns the reverse list of $\Omega$.

---

Algorithm 1: Fast Weighted Model Sampler for $\mathbf{UFO}^2$ with constraint

---

**Input**: A $\mathbf{UFO}^2$ sentence $\Gamma$, a constraint $\mathcal{C}$, a domain $\Delta$ of size $n$ and a weighting $(w, \bar{w})$
**Output**: A model $\mu$ from $\mathrm{models}_n(\Gamma \wedge \mathcal{C})$ with the probability (**??**)

1: Compute and cache all $w^{(s)}(\mathbf{k})$, $\hat{r}^{(s,t,u)}(\mathbf{k})$, $r^{(s,t)+}(\mathbf{k})$ and $r^{(s,t)-}(\mathbf{k})$ according to Section 0.1
2: $\mathbf{N} \leftarrow \{(n_1, \ldots, n_{2^m}) \mid \sum_{i=1}^m n_i = n\}$
3: // Compute count distribution
4: **for** $\mathbf{n} \in \mathbf{N}$ **do**
5:     $(\tau_1, \ldots, \tau_n) \leftarrow \mathsf{RandomAssign}(\mathbf{n}, \Delta)$
6:     Fetch all $w_i(\mathbf{k})$, $r^+_{i,j}(\mathbf{k})$ and $r^-_{i,j}(\mathbf{k})$ from cache according to $(\tau_1, \ldots, \tau_n)$
7:     Compute $W = \mathsf{WFOMC}(\Gamma \wedge \bigwedge_{i=1}^n \tau_i \wedge \mathcal{C}, n, w, \bar{w})$ by (4)
8:     $q(\mathbf{n}) \leftarrow \binom{n}{n_1, \ldots, n_{2^m}} \cdot W$
9: **end for**
10: Compute $F$ according to Section
11: // Sample a model of $\Gamma \wedge \mathcal{C}$
12: $\mathbf{n} \leftarrow \mathsf{SampleDist}(q)$
13: $T = (\tau_1, \ldots, \tau_n) \leftarrow \mathsf{RandomAssign}(\mathbf{n}, \Delta)$
14: Fetch all $w_i(\mathbf{k})$, $r^+_{i,j}(\mathbf{k})$ and $r^-_{i,j}(\mathbf{k})$ from cache according to $(\tau_1, \ldots, \tau_n)$
15: Compute $CW$ according to Section
16: $\Omega \leftarrow \{\{1,2\}, \ldots, \{n-1, n\}\}$
17: // Compute all $J_{\Omega'}$'s
18: $\Omega' \leftarrow \Omega$
19: $J_{\Omega'} = 1$
20: **for** $\{s,t\} \in \mathsf{Reverse}(\Omega)$ **do**
21:     $\Omega' \leftarrow \Omega' \cup \{\{s,t\}\}$
22:     Compute $J_{\Omega'}$ by (5)
23: **end for**
24: $Q \leftarrow 1$
25: $W \leftarrow q(\mathbf{n}) / \binom{n}{n_1, \ldots, n_{2^m}}$
26: $\Omega' \leftarrow \varnothing$
27: // Sample U-types
28: **for** $\{i,j\} \in \Omega$ **do**
29:     $\Omega' \leftarrow \Omega' \cup \{\{i,j\}\}$
30:     **for** $\rho \in \mathcal{B}$ **do**
31:         Fetch all $\hat{r}_{i,j}(\mathbf{k})$ from cache according to $\tau_i, \tau_j$ and $\rho$
32:         $Q' \leftarrow Q \odot [\hat{r}_{i,j}(\mathbf{k}_1), \ldots, \hat{r}_{i,j}(\mathbf{k}_{|\mathcal{D}|})]$
33:         $T_{\Omega'} \leftarrow (6)$
34:         $W_\rho \leftarrow \frac{F \otimes (CW \odot Q' \odot J_{\Omega'} \odot T_{\Omega'})}{\prod_{t \in [m]} \mathbf{M}_t}$
35:         **if** $\mathsf{Uniform}(0,1) < \frac{W_\rho}{W}$ **then**
36:             $Q \leftarrow Q'$
37:             $W \leftarrow W_\rho$
38:             append $\rho$ to $T$
39:             **break**
40:         **else**
41:             $W \leftarrow W - W_\rho$
42:         **end if**
43:     **end for**
44: **end for**
45: **return** the unique possible world characterised by $T$

---