Generating Random Trees Without Cross Products

Tree queries only!

- query graph: $G = (V, E)$, $|V| = n$, G must be a tree.
- level: root has level 0, children thereof 1, etc.
- $\mathcal{T}_G$: join trees for G

[7]

Partitioning $\mathcal{T}_G$

$\mathcal{T}_G^{v(k)} \subseteq \mathcal{T}_G$: subset of join trees where the leaf node (i.e. relation) v occurs at level k.

Observations:

- $n = 1$: $|\mathcal{T}_G| = |\mathcal{T}_G^{v(0)}| = 1$
- $n > 1$: $|\mathcal{T}_G^{v(0)}| = 0$ (top is a join and no relation)
- The maximum level that can occur in any join tree is $n - 1$.
  Hence: $|\mathcal{T}_G^{v(k)}| = 0$ if $k \geq n$.
- $\mathcal{T}_G = \cup_{k=0}^{n} \mathcal{T}_G^{v(k)}$
- $\mathcal{T}_G^{v(i)} \cap \mathcal{T}_G^{v(j)} = \emptyset$ for $i \neq j$
- Thus: $|\mathcal{T}_G| = \sum_{k=0}^{n} |\mathcal{T}_G^{v(k)}|$

The Specification

- The algorithm will generate an unordered tree with n leaf nodes.
- If we wish to have a random ordered tree, we have to pick one of the 2^{n-1} possibilities to order the $(n-1)$ joins within the tree.

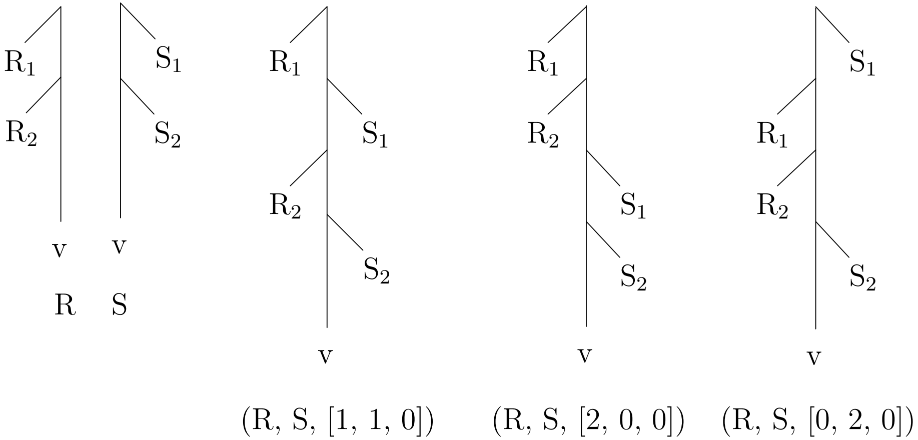
The Procedure

1. List merges (notation, specification, counting, unranking)
2. Join tree construction: leaf-insertion and tree-merging
3. Standard Decomposition Graph (SDG): describes all valid join trees
4. Counting
5. Unranking algorithm

List Merge

- Lists: Prolog-Notation: $< a|t >$
- Property P on elements
- A list l' is the *projection* of a list L on P, if L' contains all elements of L satisfying the property P.
  Thereby, the order is retained.
- A list L is a *merge* of two disjoint lists L_1 and L_2, if L contains all elements from L_1 and L_2 and both are projections of L.

Example



List Merge: Specification

A merge of a list L_1 with a list L_2 whose respective lengths are l_1 and l_2 can be described by an array $\alpha = [\alpha_0, \ldots, \alpha_{l_2}]$ of non-negative integers whose sum is equal to l_1, i.e. $\sum_{i=0}^{l_2} \alpha_i = |l_1|$.

- We obtain the merged list L by first taking α_0 elements from L_1.
- Then, an element from L_2 follows. Then follow α_1 elements from L_1 and the next element of L_2 and so on.
- Finally follow the last α_{l_2} elements of L_1.

List Merge: Counting

Non-negative integer decomposition:

- What is the number of decompositions of a non-negative integer n into k non-negative integers α_i with $\sum_{i=1}^{k} \alpha_k = n$.

Answer: $\binom{n+k-1}{k-1}$

List Merge: Counting (2)

Since we have to decompose l_1 into $l_2 + 1$ non-negative integers, the number of possible merges is $M(l_1, l_2) = \binom{l_1+l_2}{l_2}$.
The observation $M(l_1, l_2) = M(l_1 - 1, l_2) + M(l_1, l_2 - 1)$ allows us to construct an array of size $n * n$ in $O(n^2)$ that materializes the values for M.
This array will allow us to rank list merges in $O(l_1 + l_2)$.

List Merge: Unranking: General Idea

The idea for establishing a bijection between $[1, M(l_1, l_2)]$ and the possible αs is a general one and used for all subsequent algorithms of this section. Assume we want to rank the elements of some set S and $S = \cup_{i=0}^{n} S_i$ is partitioned into disjoint S_i.

1. If we want to rank $x \in S_k$, we first find the *local rank* of $x \in S_k$.
2. The rank of x is then $\sum_{i=0}^{k-1} |S_i| + \texttt{local-rank}(x, S_k)$.
3. To unrank some number $r \in [1, N]$, we first find k such that $k = \min_j r \leq \sum_{i=0}^{j} |S_i|$.
4. We proceed by unranking with the new local rank $r' = r - \sum_{i=0}^{k-1} |S_i|$ within S_k.

List Merge: Unranking

We partition the set of all possible merges into subsets.

- Each subset is determined by α_0.
  For example, the set of possible merges of two lists L_1 and L_2 with length $l_1 = l_2 = 4$ is partitioned into subsets with $\alpha_0 = j$ for $0 \leq j \leq 4$.
- In each partition, we have $M(j, l_2 - 1)$ elements.
- To unrank a number $r \in [1, M(l_1, l_2)]$ we first determine the partition by computing $k = \min_j r \leq \sum_{i=0}^{j} M(j, l_2 - 1)$.
  Then, $\alpha_0 = l_1 - k$.
- With the new rank $r' = r - \sum_{i=0}^{k} M(j, l_2 - 1)$, we start iterating all over.

Example

k	α_0	$(k, l_2 - 1)$	$M(k, l_2 - 1)$	rank intervals
0	4	$(0, 3)$	1	$[1, 1]$
1	3	$(1, 3)$	4	$[2, 5]$
2	2	$(2, 3)$	10	$[6, 15]$
3	1	$(3, 3)$	20	$[16, 35]$
4	0	$(4, 3)$	35	$[36, 70]$

Decomposition

UnrankDecomposition(r, l_1, l_2)
Input: a rank r, two list sizes l_1 and l_2
Output: encoding of the inner leafes of a tree
alpha $= <>$, $k = 0$
while $l_1 > 0 \wedge l_2 > 0$ {
  $m = M(k, l_2 - 1)$
  if $r \leq m$ {
    alpha=alpha$\circ < l_1 - k >$
    $l_1 = k, k = 0, l_2 = l_2 - 1$
  } **else** {
    $r = r - m, k = k + 1$
  }
}
return alpha$\circ < l_1 > \circ \bigcirc_{|alpha|+1 \leq i < l_2} < 0 >$

Anchored List Representation of Join Trees

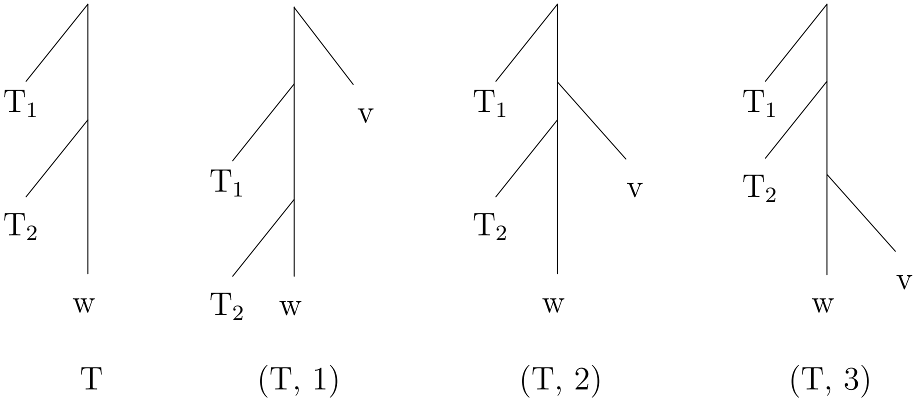
Definition Let T be a join tree and v be a leaf of T. The *anchored list representation* L of T is constructed as follows:

- If T consists of the single leaf node v, then $L = <>$.
- If $T = (T_l \bowtie T_2)$ and without loss of generality v occurs in T_2, then $L = < T_1 | L_2 >$ where L_2 is the anchored list representation of T_2.

We then write $T = (L, v)$.

Observation If $T = (L, v) \in \mathcal{T}_G$ then $T \in \mathcal{T}_G^{v(k)} \prec\succ |L| = k$

Leaf-Insertion: Example



Leaf-Insertion

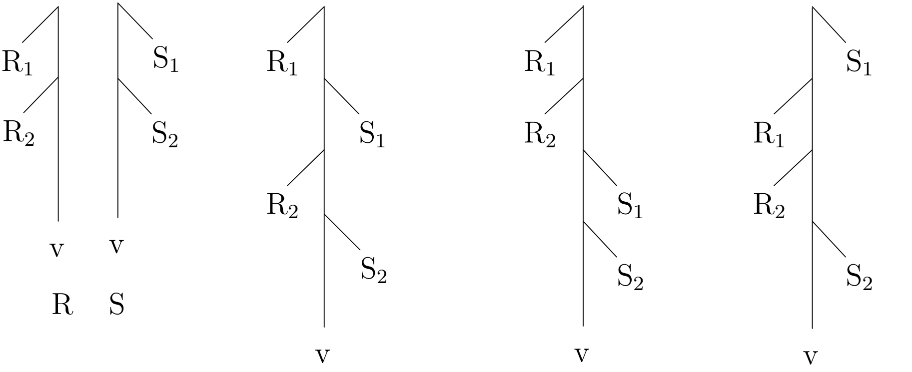
Definition Let $G = (V, E)$ be a query graph, T a join tree of G. $v \in V$ be such that $G' = G|_{V \setminus \{v\}}$ is connected, $(v, w) \in E$, $1 \leq k < n$, and

$$\begin{aligned} T &= (< T_1, \ldots, T_{k-1}, v, T_{k+1}, \ldots, T_n >, w) \\ T' &= (< T_1, \ldots, T_{k-1}, T_{k+1}, \ldots, T_n >, w). \end{aligned}$$

Then we call (T', k) an *insertion pair* on v and say that T is *decomposed into* (or *constructed from*) the pair (T', k) on v.

Observation: Leaf-insertion defines a bijective mapping between $\mathcal{T}_G^{v(k)}$ and insertion pairs (T', k) on v, where T' is an element of the disjoint union $\cup_{i=k-1}^{n-2} \mathcal{T}_{G'}^{w(i)}$.

Tree-Merging: Example



(R, S, [1, 1, 0]) (R, S, [2, 0, 0]) (R, S, [0, 2, 0])

Tree-Merging

Two trees $R = (L_R, w)$ and $S = (L_S, w)$ on a common leaf w are merged by merging their anchored list representations.

Definition. Let $G = (V, E)$ be a query graph, $w \in V$, $T = (L, w)$ a join tree of G, $V_1, V_2 \subseteq V$ such that $G_1 = G|_{V_1}$ and $G_2 = G|_{V_2}$ are connected, $V_1 \cup V_2 = V$, and $V_1 \cap V_2 = \{w\}$. For $i = 1, 2$:

- Define the property P_i to be "every leaf of the subtree is in V_i",
- Let L_i be the projection of L on P_i.
- $T_i = (L_i, w)$.

Let α be the integer decomposition such that L is the result of merging L_1 and L_2 on α. Then, we call (T_1, T_2, α) a *merge triplet*. We say that T is *decomposed into* (*constructed from*) (T_1, T_2, α) on V_1 and V_2.

Observation

Tree-Merging defines a bijective mapping between $\mathcal{T}_G^{w(k)}$ and merge triplets (T_1, T_2, α), where $T_1 \in \mathcal{T}_{G_1}^{w(i)}$, $T_2 \in \mathcal{T}_{G_2}^{w(k-i)}$, and α specifies a merge of two lists of sizes i and $k-i$. Further, the number of these merges (i.e. the number of possibilities for α) is $\binom{i+(k-i)}{k-i} = \binom{k}{i}$.

Standard Decomposition Graph (SDG)

A *standard decomposition graph* of a query graph describes the possible constructions of join trees.
It is not unique (for $n > 1$) but anyone can be used to construct all possible unordered join trees.
For each of our two operations it has one kind of inner nodes:

- A unary node labeled $+_v$ stands for leaf-insertion of v.
- A binary node labeled $*_w$ stands for tree-merging its subtrees whose only common leaf is w.

Constructing a Standard Decomposition Graph

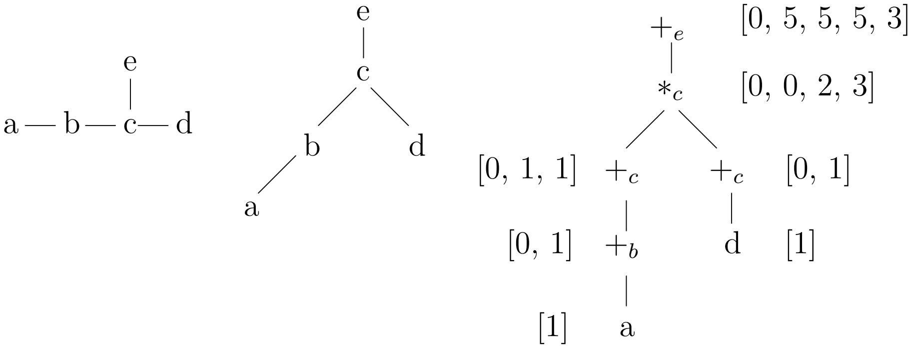
The standard decomposition graph of a query graph $G = (V, E)$ is constructed in three steps:

1. pick an arbitrary node $r \in V$ as its root node
2. transform G into a tree G' by directing all edges away from r;
3. call QG2SDG(G', r)

Constructing a Standard Decomposition Graph (2)

QG2SDG(G', r)
Input: a query tree $G' = (V, E)$ and its root r
Output: a standard query decomposition tree of G'
Let $\{w_1, \ldots, w_n\}$ be the children of v
switch n {
  case 0: label v with "v"
  case 1:
    label v as "$+_v$"
    QG2SDG(G', w_1)
  otherwise:
    label v as "$*_v$"
    create new nodes l, r with label $+_v$
    $E = E \setminus \{(v, w_i) | 1 \leq i \leq n\}$
    $E = E \cup \{(v, l), (v, r), (l, w_1)\} \cup \{(r, w_i) | 2 \leq i \leq n\}$
    QG2SDG(G', l), QG2SDG(G', r)
}
return G'

Constructing a Standard Decomposition Graph (3)



Counting

For efficient access to the number of join trees in some partition $\mathcal{T}_G^{v(k)}$ in the unranking algorithm, we materialize these numbers.
This is done in the `count` array.
The semantics of a `count` array $[c_0, c_1, \ldots, c_n]$ of a node u with label $\circ_v$ ($\circ \in \{+, *\}$) of the SDG is that

- u can construct c_i different trees in which leaf v is at level i.

Then, the total number of trees for a query can be computed by summing up all the c_i in the `count` array of the root node of the decomposition tree.

Counting (2)

To compute the count and an additional `summand` adornment of a node labeled $+_v$, we use the following lemma:

Lemma. Let $G = (V, E)$ be a query graph with n nodes, $v \in V$ such that $G' = G|_{V \setminus v}$ is connected, $(v, w) \in E$, and $1 \leq k < n$. Then

$$|\mathcal{T}_G^{v(k)}| = \sum_{i \geq k-1} |\mathcal{T}_{G'}^{w(i)}|$$

Counting (3)

The sets $\mathcal{T}_{G'}^{w(i)}$ used in the summands of the former Lemma directly correspond to subsets $\mathcal{T}_{G}^{v(k),i}$ $(k-1 \leq i \leq n-2)$ defined such that $T \in \mathcal{T}_{G}^{v(k),i}$ if

1. $T \in \mathcal{T}_{G}^{v(k)}$,
2. the insertion pair on v of T is (T', k), and
3. $T' \in \mathcal{T}_{G'}^{w(i)}$.

Further, $|\mathcal{T}_{G}^{v(k),i}| = |\mathcal{T}_{G'}^{w(i)}|$. For efficiency, we materialize the summands in an array of arrays `summands`.

Counting (4)

To compute the `count` and `summand` adornment of a node labeled $*_v$, we use the following lemma.

Lemma. Let $G = (V, E)$ be a query graph, $w \in V$, $T = (L, w)$ a join tree of G, $V_1, V_2 \subseteq V$ such that $G_1 = G|_{V_1}$ and $G_2 = G|_{V_2}$ are connected, $V_1 \cup V_2 = V$, and $V_1 \cap V_2 = \{v\}$. Then

$$|\mathcal{T}_G^{v(k)}| = \sum_i \binom{k}{i} \; |\mathcal{T}_{G_1}^{v(i)}| \; |\mathcal{T}_{G_2}^{v(k-i)}|$$

Counting (5)

The sets $\mathcal{T}_{G'}^{w(i)}$ used in the summands of the previous Lemma directly correspond to subsets $\mathcal{T}_G^{v(k),i}$ $(0 \leq i \leq k)$ defined such that $T \in \mathcal{T}_G^{v(k),i}$ if

1. $T \in \mathcal{T}_G^{v(k)}$,
2. the merge triplet on V_1 and V_2 of T is (T_1, T_2, α), and
3. $T_1 \in \mathcal{T}_{G_1}^{v(i)}$.

Further, $|\mathcal{T}_G^{v(k),i}| = \binom{k}{i} \; |\mathcal{T}_{G_1}^{v(i)}| \; |\mathcal{T}_{G_2}^{v(k-i)}|$.

Counting (6)

Observation: Assume a node v whose `count` array is $[c_1, \ldots, c_m]$ and whose `summands` is $s = [s^0, \ldots, s^n]$ with $s_i = [s^i_0, \ldots, s^i_m]$, then

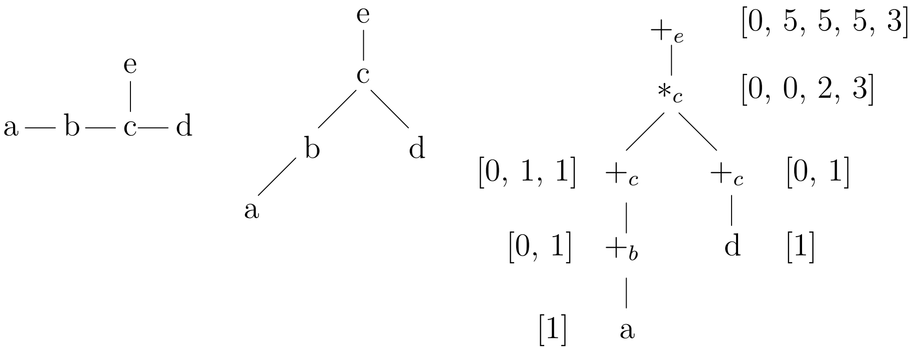
$$c_i = \sum_{j=0}^{m} s^i_j$$

holds.

The following algorithm has worst-case complexity $O(n^3)$.

Looking at the `count` array of the root node of the following SDG, we see that the total number of join trees for our example query graph is 18.

SDG example



Annotating the SDG

Adorn(v)
Input: a node v of the SDG
Output: v and nodes below are adorned by count and summands
Let $\{w_1, \ldots, w_n\}$ be the children of v
switch (n) {
  case 0: count(v) $= [1]$ // no summands for v
  case 1:
    Adorn(w_1)
    assume count(w_1) $= [c_0^1, \ldots, c_{m_1}^1]$;
    count(v) $= [0, c_1, \ldots, c_{m_1+1}]$ where $c_k = \sum_{i=k-1}^{m_1} c_i^1$
    summands(v) $= [s^0, \ldots, s^{m_1+1}]$ where $s^k = [s_0^k, \ldots, s_{m_1+1}^k]$ and

$$s_i^k = \begin{cases} c_i^1 & \text{if } 0 < k \text{ and } k-1 \leq i \\ 0 & \text{else} \end{cases}$$

Annotating the SDG (2)

case 2:

Adorn(w_1)
Adorn(w_2)
assume count(w_1) $= [c^1_0, \ldots, c^1_{m_1}]$
assume count(w_2) $= [c^2_0, \ldots, c^2_{m_2}]$
count(v) $= [c_0, \ldots, c_{m_1+m_2}]$ where
$c_k = \sum_{i=0}^{m_1} \binom{k}{i} c^1_i c^2_{k-i}$; // $c^2_i = 0$ for $i \notin \{0, \ldots, m_2\}$
summands(v) $= [s^0, \ldots, s^{m_1+m_2}]$ where $s^k = [s^k_0, \ldots, s^k_{m_1}]$ and

$$s^k_i = \begin{cases} \binom{k}{i} c^1_i c^2_{k-i} & \text{if } 0 \leq k - i \leq m_2 \\ 0 & \text{else} \end{cases}$$

}

Unranking: top-level procedure

The algorithm `UnrankLocalTreeNoCross` called by `UnrankTreeNoCross` adorns the standard decomposition graph with `insert-at` and `merge-using` annotations. These can then be used to extract the join tree.

UnrankTreeNoCross(r,v)
Input: a rank r and the root v of the SDG
Output: adorned SDG
let $\text{count}(v) = [x_0, \ldots, x_m]$
$k = \min_j r \leq \sum_{i=0}^{j} x_i$
$r' = r - \sum_{i=0}^{k-1} x_i$
$\text{UnrankLocalTreeNoCross}(v,\ r',\ k)$

Unranking: Example

The following table shows the intervals associated with the partitions $\mathcal{T}_G^{e(k)}$ for our standard decomposition graph:

Partition	*Interval*
$\mathcal{T}_G^{e(1)}$	$[1, 5]$
$\mathcal{T}_G^{e(2)}$	$[6, 10]$
$\mathcal{T}_G^{e(3)}$	$[11, 15]$
$\mathcal{T}_G^{e(4)}$	$[16, 18]$

Unranking: the last utility function

The unranking procedure makes use of unranking decompositions and unranking triples. For the latter and a given X, Y, Z, we need to assign each member in

$$\{(x, y, z) | 1 \leq x \leq X, 1 \leq y \leq Y, 1 \leq z \leq Z\}$$

a unique number in $[1, XYZ]$ and base an unranking algorithm on this assignment. We call the function `UnrankTriplet`(r, X, Y, Z). r is a rank and X, Y, and Z are the upper bounds for the numbers in the triplets.

Unranking Without Cross Products

UnrankingTreeNoCrossLocal(v, r, k)
Input: an SDG node v, a rank r, a number k identifying a partition
Output: adornments of the SDG as a side-effect
Let $\{w_1, \ldots, w_n\}$ be the children of v
switch n {
  case 0:
    // no additional adornment for v

Unranking Without Cross Products (2)

case 1:

let count$(v) = [c_0, \ldots, c_n]$

let summands$(v) = [s^0, \ldots, s^n]$

$k_1 = \min_j r \leq \sum_{i=0}^{j} s_i^k$

$r_1 = r - \sum_{i=0}^{k_1-1} s_i^k$

insert-at$(v) = k$

UnrankingTreeNoCrossLocal(w_1, r_1, k_1)

Unranking Without Cross Products (3)

case 2:

let count(v) $= [c_0, \ldots, c_n]$
let summands(v) $= [s^0, \ldots, s^n]$
let count(w_1) $= [c^1_0, \ldots, c^1_{n_1}]$
let count(w_2) $= [c^2_0, \ldots, c^2_{n_2}]$
$k_1 = \min_j r \leq \sum_{i=0}^{j} s^k_i$
$q = r - \sum_{i=0}^{k_1-1} s^k_i$
$k_2 = k - k_1$
$(r_1, r_2, a) =$ UnrankTriplet$(q, c^1_{k_1}, c^2_{k_2}, \binom{k}{i})$
$\alpha =$ UnrankDecomposition(a)
merge-using$(v) = \alpha$
UnrankingTreeNoCrossLocal(w_1, r_1, k_1)
UnrankingTreeNoCrossLocal(w_2, r_2, k_2)

}

Quick Pick

- problem: build (pseudo-)random join trees fast
- unranking without cross products is quite involved
- idea: randomly select an edge in the query graph
- extend join tree by selected edge

No longer uniformly distributed, but very fast

Quick Pick (2)

QuickPick(Query Graph G)
Input: a query graph $G = (\{R_1, \ldots, R_n\}, E)$
Output: a bushy join tree
$E' = E$;
Trees $= \{R_1, \ldots, R_n\}$;
while $|\text{Trees}| > 1$ {
  choose a random $e \in E'$
  $E' = E' \setminus \{e\}$
  if e connects two relations in different subtrees $T_1, T_2 \in$ Trees
    Trees $=$ Trees$\setminus\{T_1, T_2\}\cup$CreateJoinTree$(T_1, T_2)$
}
return $T \in$Trees

- repeated multiple times to find a good tree