This vignette document sets the following goals: expound the Prize-Collecting Steiner tree problem, give some background on Belief propagation algorithm, explain in details how BP is used for solving the Prize-Collecting Steiner tree problem, discuss several additional features for graph analysis and illustrate a typical workflow with the package.

Contents

1 Steiner tree problem .................................... 2

2 Belief propagation ................................... 3

  2.1 Preliminaries: graphical models and statistical inference .............. 3
     2.1.1 Bayesian networks .................................. 3
     2.1.2 Markov random fields ................................ 3
     2.1.3 Factor graphs ...................................... 3
     2.1.3 Statistical inference ................................ 4

  2.2 Belief propagation .................................. 4
     2.2.1 Sum-product ....................................... 4
     2.2.2 Max-product ....................................... 5
     2.2.3 Max-sum ........................................... 5
     2.2.4 Complexity and correctness .......................... 6

  2.3 Loopy belief propagation ................................ 6
     2.3.1 Complexity and correctness .......................... 6

3 BP for PCST ........................................... 8

  3.1 Graphical model ...................................... 8

  3.2 Message-passing equations for loop-free graphs ......................... 8

  3.3 Message-passing equations for graphs with cycles ..................... 10
     3.3.1 Implementation details ................................ 10
     3.3.2 Complete algorithm .................................. 11

4 Workflow with the package ................................ 13
1 Steiner tree problem

The Steiner problem is a classical combinatorial optimisation problem. The interest in it arises from a wide range of practical applications in the areas such as chemistry, biology, telecommunication and many more. For example, the goal of the Steiner tree problem in systems biology is to detect biological relationship between a set of distinguished proteins, metabolites or genes.

Let $G = (V, E)$ be an undirected graph where $V$ is a vertex set and $E$ is an edge set. Suppose that graph is weighted, i.e. graph with a cost function $c : E \rightarrow \mathbb{R}_+$. Given a required node subset $T \subseteq V$ the Steiner tree problem for $T$ in $G$ is to find connected, minimum cost subgraph $G_1 = (V_1, E_1)$ with $T \subseteq V_1 \subseteq V$ and $E_1 \subseteq E$. Note, that resulting subgraph is necessary a tree. The elements of $T$ are called terminals and elements of $V_1 \setminus T$ are Steiner nodes.

**Problem 1.1 (Steiner tree problem).**

*Input:* an undirected graph $G$, a set $T \subseteq V$ and weights $c : E \rightarrow \mathbb{R}_+$

*Output:* a minimum weight tree connecting all vertices of $T$ in $G$

**Theorem 1.1.** The Steiner tree problem is $NP$-hard, even for unit weights (Karp 1972).

There are two special cases of the Steiner tree problem. If $|T| = 2$, then the problem is reduced to the Shortest Path problem between a pair of vertices in a graph, which is in the class $P$. The second special case to be considered is when $|T| = |V|$ and the problem is reduced to finding Minimum Spanning tree, which is again solvable in polynomial time.

In many cases nodes have an additional numerical value, which represents their significance. That is where the Prize-Collecting Steiner Tree problem arises. Roughly speaking, the goal is to find a subgraph $G_1$ in $G$ connecting all the terminals $T$ with the most expensive nodes and least expensive edges. Note, that the result will be a tree as in the Steiner problem.

**Problem 1.2 (Prize-Collecting Steiner Tree problem).**

*Input:* an undirected graph $G$, a set $T \subseteq V$, costs $c : E \rightarrow \mathbb{R}_+$ and prizes $p : V \rightarrow \mathbb{R}_+$

*Output:* find a tree $G_1$ by minimizing the following function $f(G_1) = \sum_{e \in E_1} c_e + \lambda \sum_{i \in V_1} p_i$

Since a constant value does not change a maximum, we can subtract a total node prizes from objective function $C = \sum_{i \in V} p_i$.

**Problem 1.2 (revisited).**

*Input:* an undirected graph $G$, a set $T \subseteq V$, costs $c : E \rightarrow \mathbb{R}_+$ and prizes $p : V \rightarrow \mathbb{R}_+$

*Output:* find a tree $G_1$ by minimizing the following function $f(G_1) = \sum_{e \in E_1} c_e - \lambda \sum_{i \in V_1} p_i$

The Prize-Collecting Steiner Tree problem reduces to the Steiner problem when all prizes are equal to one, so it is at least $NP$-hard. Thus, we do not expect to have an efficient algorithm for solving PCST problem exactly.

**References**

2 Belief propagation

The approximation algorithm for PCST problem internally utilizes loopy belief propagation equations. In this section we remind several basic definitions in probability theory and statistics which are crucial for understanding the material presented in the later sections of this vignette document.

For more details in implementation of belief propagation and loopy belief propagation we refer to this GitHub repository.

2.1 Preliminaries: graphical models and statistical inference

*Graphical model* is a compact representation of a collection of probability distributions. It consists of a graph \( G = (V, E) \), directed or undirected, where each vertex \( v \in V \) is associated with a random variable. Edges of the graph represent statistical relationship between nodes. There are several main types of graphical models:

- Bayesian networks
- Markov random fields
- Factor graphs

### 2.1.1 Bayesian networks

*Bayesian network* is a directed graph. Each node represents random variable \( x_i \) which has an associated conditional probability distribution or local probabilistic model. A direct edge from \( x_i \) to \( x_j \) represents a conditional probability of a random variable given its parents \( P(x_i | x_j) \). Bayesian network defines a joint probability distribution in the following way:

\[
P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | \text{Par}(x_i))
\]

### 2.1.2 Markov random fields

*Markov random field* is based on undirected graphical models. As in a Bayesian network, nodes in a graph represent variables. An associated probability distribution factorizes into functions, each function associated with a clique of the graph:

\[
P(x_1, \ldots, x_n) = Z^{-1} \prod_{C \in \mathcal{C}} \psi_C(x_C)
\]

where \( Z \) is a constant chosen to ensure that the distribution is normalized. The set \( \mathcal{C} \) is often taken to be the set of all maximal cliques of the graph. For a general undirected graph the compatibility functions \( \psi_C \) need not have any obvious or direct relation to probabilities or conditional distributions defined over the graph cliques.

A special case of Markov random field is a *pairwise Markov random field* where the probability distribution factorizes into functions of two variables.

### 2.1.3 Factor graphs

The most general parameterization is a *factor graph*. Factor graphs explicitly draw out the factors in a factorization of the joint probability. Note, that it is possible to convert arbitrary MRF or Bayesian network into equivalent factor graph.

**Definition 2.1.** Factor graph is a pair \( \mathcal{F} = (G, \{f_1, \ldots, f_n\}) \), where

- \( G = (V, E) \) is an undirected bipartite graph such that \( V = X \cup F \), where \( X \cup F = \emptyset \). The nodes \( X \) are variable nodes, while the nodes \( F \) are factor nodes.
Further, $f_1, \ldots, f_n$ are positive functions and the number of functions equals the number of nodes in $F = \{F_1, \ldots, F_n\}$. Each node $F_i$ is associated with the corresponding function $f_i$ and each $f_i$ is a function of the neighboring variables of $F_i$, i.e. $f_i = f_i(\text{Nb}(F_i))$.

Joint probability distribution of a factor graph of $N$ variables with $M$ functions factorizes as follows:

$$P(\{x\}) = Z^{-1} \prod_{a=1}^{M} \psi(\{x\}_a)$$

(2.3)

where $Z$ is a normalization constant.

### 2.1.3 Statistical inference

Both directed and undirected graphical models represent a full joint probability distribution. It is important to solve the following computational inference problems:

- computing the marginal distribution $P(\{x\}_A)$ over a particular subset $A \in V$ of nodes, i.e. sum over all the possible states of all the other nodes in the system
- computing the conditional distribution $P(\{x\}_A | \{x\}_B)$, where $A \cap B = \emptyset$ and $A, B \in V$
- computing the maximum a posteriori (MAP), i.e. finding the most likely joint assignment to a particular set of variables: $\text{argmax}_{\{x\}_A} P(\{x\}_A | \{x\}_B)$

**Definition 2.2.** Marginal probabilities that are computed approximately are called beliefs. The belief at node $i$ is denoted by $b(x_i)$.

**Theorem 2.1.** Every type of inference in graphical models is NP-hard. Even simplest problem of computing the distribution over a single binary variable is NP-hard.

### 2.2 Belief propagation

In this section we will consider only singly-connected probabilistic graphical models. In later subsection, we will extend the algorithm to graphs with loops.

Belief propagation is a message-passing algorithm for solving inference tasks, at least approximately. The BP equations for Bayesian networks, MRFs and factor graphs slightly differs from each other, but, in fact, they are all mathematically equivalent. To keep things simple, we will stick with pairwise MRFs, since the version for them has only one kind of message, while the BP equations, for example, for factor graphs are described using two kinds of messages: from factor to variable and vice versa. According to our assumption, distribution factorizes as follows:

$$P(\{x\}) = Z^{-1} \prod_{(ij)} \psi_{(ij)}(x_i, x_j)$$

(2.4)

#### 2.2.1 Sum-product

Here we will introduce an algorithm for performing marginalization in loop-free graphical models. Let’s consider the following simple pairwise MRF: $P(\{x\}) = Z^{-1} \psi_1(x_1, x_2)\psi_2(x_2, x_4)\psi_3(x_2, x_3)$. Our goal is to compute marginal distribution of $x_1$:
\[P(x_1) = Z^{-1} \sum_{x_2,x_3,x_4} \psi_1(x_1,x_2)\psi_2(x_2,x_4)\psi_3(x_2,x_3) =
\]
\[= Z^{-1} \sum_{x_2} \psi_1(x_1,x_2) \left[ \sum_{x_4} \psi_2(x_2,x_4) \right] \left[ \sum_{x_3} \psi_3(x_2,x_3) \right] =
\]
\[= Z^{-1} \sum_{x_2} \psi_1(x_1,x_2) \mu_{4\to2}(x_2) \mu_{3\to2}(x_2) =
\]
\[= Z^{-1} \mu_{2\to1}(x_1)
\]

The generalization of the procedure above is exactly belief propagation or sum-product algorithm.

**Algorithm 2.1:** Sum-product algorithm for trees

**Input:** PGM, variable node \(n \in V\);

**Output:** marginal distribution \(P(x_n)\);

recursively compute messages;

\[\mu_{i\to j}(x_j) = \sum_{x_i} \psi_{ij}(x_i,x_j) \prod_{k \in \text{Nb}(x_i) \setminus \{x_j\}} \mu_{k\to i}(x_i);
\]

return marginal distribution \(P(x_n) = Z^{-1} \prod_{i \in \text{Nb}(x_n)} \mu_{i\to n}(x_n)\)

**Definition 2.3.** \(\mu_{i\to j}(x_j)\) is a so-called message from variable \(x_i\) to variable \(x_j\).

### 2.2.2 Max-product

The Max-product algorithm computes the *max-marginal* at each node of a graph \(\bar{p}_{x_i}(x_i) = \max_{x_j: j \neq i} p_x(\{x\})\).

**Proposition 2.1.** If there are no ties at each node, then \((x_1^*, ..., x_n^*)\), where \(x_i^* = \text{argmax}_{x_i} \bar{p}_{x_i}(x_i)\) is a unique global MAP assignment.

The idea behind Max-product is nearly the same as for Sum-product: instead of summing over the states of other nodes, we should find the max over those states. The max operator passes through variables just as the summation sign did.

**Algorithm 2.2:** Max-product algorithm for trees

**Input:** PGM;

**Output:** MAP assignment;

recursively compute messages;

\[\mu_{i\to j}(x_j) = \max_{x_i} \psi_{ij}(x_i,x_j) \prod_{k \in \text{Nb}(x_i) \setminus \{x_j\}} \mu_{k\to i}(x_i);
\]

calculate max-marginals \(\bar{p}_{x_i}(x_i) = \prod_{j \in \text{Nb}(x_i)} \mu_{j\to i}(x_i) \forall i \in V\);

retrieve MAP assignment

### 2.2.3 Max-sum

In sum-product and max-product algorithms we perform a lot of factor and message multiplications and it potentially can lead to numerical overflow or underflow. One way to avoid it is just to normalize each message, while the other one is more sophisticated and requires to perform all computations in a log space. We will consider the latter. Taking log from both parts of Max-product equations, we get the following:

\[\tilde{\mu}_{i\to j}(x_j) = \max_{x_i} \left\{ \ln \psi_{ij}(x_i,x_j) + \sum_{k \in \text{Nb}(x_i) \setminus \{x_j\}} \tilde{\mu}_{k\to i}(x_i) \right\}
\]
\[\tilde{p}_{x_i}(x_i) = \exp \left\{ \sum_{j \in \text{Nb}(x_i)} \tilde{\mu}_{k\to i}(x_i) \right\}
\]
All multiplications are now replaced by additions, and equations take the so-called MS form.

2.2.4 Complexity and correctness

**Theorem 2.2.** The BP algorithm is exact if the topology of the PGM is that of a tree or a chain.

Recall, that every type of inference in graphical models is $NP$-hard. Suppose, that each variable has $k$ number of states. If we know nothing about the structure of the joint probability, evaluation of desired marginal probability distribution would require $O(k^{|V|})$ computations, where $|V|$ - number of nodes in the graph. Fortunately, factorization of the sum as we did in Eq. (2.5) reduces the total complexity to $O(|V| \cdot k^2)$.

So, BP algorithm is exact and efficient algorithm for performing inference in loop-free probabilistic graphical models.

2.3 Loopy belief propagation

Since BP equations are local we can apply them to PGMs with cycles. The corresponding modification is called loopy belief propagation. There is still the question of what update rules we use to recompute the messages. We illustrate a parallel schedule by the example of Max-sum equations.

**Algorithm 2.3:** Loopy belief propagation (max-sum)

**Input:** PGM, $t_{\text{max}}$

**Output:** MAP assignment

initialize all messages $\mu_{i \rightarrow j}^{(0)} = 1 \ \forall (i, j) \in E$

for $t = 1, \ldots, t_{\text{max}}$ do

\[
\tilde{m}^{t+1}_{i \rightarrow j}(x_j) = \max_{x_i} \left\{ \ln \psi_{ij}(x_i, x_j) + \sum_{k \in \text{Nb}(x_i) \setminus \{x_j\}} \tilde{m}^t_{k \rightarrow i}(x_i) \right\}
\]

end

compute max-marginals $\tilde{p}_{x_i}(x_i) = \exp \left\{ \sum_{j \in \text{Nb}(x_i)} \tilde{m}^{t_{\text{max}}+1}_{j \rightarrow i}(x_i) \right\}$

retrieve MAP assignment

Other schedules for message-passing are possible [8,9].

2.3.1 Complexity and correctness

Unfortunately, Loopy belief propagation may **not converge** on graphs with cycles, however, in many cases it provides quite good approximation, because BP fixed-points correspond to local stationary points of the Bethe free energy.

**References**

3 BP for PCST

This section is dedicated to the algorithm for solving the Prize-Collecting Steiner tree problem via belief propagation introduced by the working group of Bayati, Braunstein et al. [1-3]. The D-bounded rooted PCST is being considered which means that the goal is to solve PCST problem with given root \( r \) and maximum depth \( D \). This section provides only an overview of the algorithm and does not cover material in great details.

**Problem 3.1 (D-bounded rooted PCST).**

*Input:* an undirected graph \( G \), a set \( T \subseteq V \), costs \( c : E \rightarrow R_+ \), prizes \( b : V \rightarrow R_+ \), root \( r \in V \), depth \( D \).  

*Output:* \( r \)-rooted tree \( G_1 = (V_1, E_1) \) with depth \( D \) which minimizes the following function \( f(G_1) = \sum_{e \in E_1} c_e + \lambda \sum_{i \in V_1} b_i \)

3.1 Graphical model

The problem is modelled using pairwise Markov Random Field. Each node \( i \in V \) is assigned to a couple \((p_i, d_i)\), which has the following meaning:

- \( d_i \in \{1, ..., D\} \) denotes the distance to the given root \( r \)
- \( p_i \in \text{Nb}(i) \cup \{\ast\} \), where \( \text{Nb}(i) = \{j : (ij) \in E\} \) and \( p_i = \{\ast\} \iff i \notin V_1 \). It represents a parent of a node \( i \).

Variables \((p_i, d_i)\) can not take arbitrary values, that is, they have to satisfy the following constraints \( \forall (ij) \in E \):

\[
\text{if } p_i = j \Rightarrow \begin{cases} d_i = d_j + 1 \\ p_j \neq \{\ast\} \end{cases}
\]

(3.1)

Should we introduce the auxiliary function \( f_{ij} = \mathbb{1}_{p_i=j} \mathbb{1}_{d_i=d_j+1} \mathbb{1}_{p_j \neq \{\ast\}} = 1 - \delta_{p_i,j} [1 - \delta_{d_i,d_j+1} (1 - \delta_{p_i,\ast})] \), the constraints will take the following form:

\[
g_{ij} = f_{ij} f_{ji} = 1
\]

(3.2)

If \( c_{i\ast} = \lambda b_i \), then the Problem 3.1 can be rewritten as follows:

\[
\min \{ \mathcal{H}(p) = \sum_{i \in V} c_{ip_i} : (d, p) \in T \}
\]

(3.3)

where \( d = \{d_i\}_{i \in V}, p = \{p_i\}_{i \in V} \), \( T = \{(d, p) : g_{ij} = 1 \ \forall (ij) \in E\} \).

The corresponding MRF probability distribution can be written in the following way:

\[
P(d, p) = Z^{-1} \prod_{(ij) \in E} g_{ij} \prod_{i \in V} e^{-c_{ip_i}}
\]

(3.4)

The key idea of the method is to apply message passing equations (2.6) to maximize probability distribution (3.4) or equivalently to find a solution to (3.3).

3.2 Message-passing equations for loop-free graphs

In this section we are going to introduce message update rules for loop-free graphs. Taking into consideration max-sum equations (2.6):

\[
m_{i \rightarrow j}(d_j, p_j) = \max_{d_i, p_i} \left\{ -c_{ip_i} + \sum_{k \in \text{Nb}(i) \setminus j} m_{k \rightarrow i}(d_k, p_k) \right\}
\]

(3.5)
A bit of analysis reveals update rules for $A, B, C$:

$$m_{i \rightarrow j}(d_j, p_j) = \begin{cases} \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, & p_j = i \\ \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, & p_j = \ast \\ \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, & p_j \neq \{ i, \ast \} \\ \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, p_j = i \\ \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, p_j = \ast \\ \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, p_j \neq \{ i, \ast \} \\ \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, p_j = i \\ \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, p_j = \ast \\ \max_{d_i, p_i: g_{j,i}=1} \{ \cdot \}, p_j \neq \{ i, \ast \} \\ \end{cases}$$

Introducing new variables $A_{i \rightarrow j}^d = \max_{p_i \neq \{ j, \ast \}} \{ \cdot \}|_{d_i=d}, B_{i \rightarrow j}^d = \max_{p_i = \ast} \{ \cdot \}|_{d_i=d}, C_{i \rightarrow j}^d = \max_{p_i = \ast} \{ \cdot \}|_{d_i=d}$ allows us to rewrite messages in more convenient way:

$$m_{i \rightarrow j}(d_j, p_j) = \begin{cases} A_{i \rightarrow j}^{d_j-1} & p_j = i \\ \max_{d_i} \max[A_{i \rightarrow j}^d; B_{i \rightarrow j}^d] & p_j = \ast \\ \max \left[ C_{i \rightarrow j}^{d_j-1}; \max_{d_i} \max[A_{i \rightarrow j}^d; B_{i \rightarrow j}^d] \right] & p_j \neq \{ i, \ast \} \\ A_{i \rightarrow j}^{d_j-1} & p_j = i \\ D_{i \rightarrow j} & p_j = \ast \\ \max \left[ C_{i \rightarrow j}^{d_j-1}; D_{i \rightarrow j} \right] & p_j \neq \{ i, \ast \} \\ \end{cases}$$

(3.7)

A bit of analysis reveals update rules for $A, B, C$:

$$A_{i \rightarrow j}^d = \max_{p_i \neq \{ j, \ast \}} \{ \cdot \}|_{d_i=d} = \max_{p_i \neq \{ j, \ast \}} \left\{ -c_{ip_i} + \sum_{l \in \text{Nb}(i) \setminus j} m_{l \rightarrow i}(d, p_l) \right\}$$

(3.8)

$$B_{i \rightarrow j}^d = \max_{p_i = \ast} \{ \cdot \}|_{d_i=d} = -c_{i \ast} + \sum_{k \in \text{Nb}(i) \setminus j} m_{k \rightarrow i}(d, \ast)$$

(3.9)
\[ C_{i \rightarrow j}^d = \max_{p_i = j} \left\{ \right\} |d_i = d = -c_{ij} + \sum_{k \in \text{Nb}(i) \setminus j} m_{k \rightarrow i}(d, j) \]
\[ = -c_{ij} + \sum_{k \in \text{Nb}(i) \setminus j} E_{k \rightarrow j}^d \]  

(3.10)

By analogy with (2.6) one can derive equation for computing marginal distribution of node \( x_j \):

\[ b(d_j, p_j) = -c_{jp_j} + \sum_{k \in \text{Nb}(i)} m_{k \rightarrow j}(d_j, p_j) \]  

(3.11)

Consideration of different cases for \( p_j \) simplify the overall analysis:

\[ b(d_j, p_j) = \begin{cases}  
-c_{ji} + \sum_{k \in \text{Nb}(j)} m_{k \rightarrow j}(d_j, i), & p_j = i \in \text{Nb}(j) \\
-c_{j*} + \sum_{k \in \text{Nb}(j)} m_{k \rightarrow j}(d_j, *), & p_j = \{*\} 
\end{cases} \]

\[ = \begin{cases}  
\sum_{k \in \text{Nb}(j)} E_{k \rightarrow j}^d + (-c_{ji} - E_{i \rightarrow j}^d + A_{i \rightarrow j}^{d-1}), & p_j = i \in \text{Nb}(j) \\
-c_{j*} + \sum_{k \in \text{Nb}(j)} D_{k \rightarrow j}, & p_j = \{*\} 
\end{cases} \]

(3.12)

3.3 Message-passing equations for graphs with cycles

3.3.1 Implementation details

Noise

Taking into account Proposition 2.1 we need somehow to resolve a ties, thus according to [3] a random noise \( \xi \in \text{Uniform}[0,1] \) is added to messages at the initialization step. Consequently, we can assume that a solution will be unique. Note, that variables still have to satisfy the equations (3.7).

Stopping criteria

The following stopping criteria are utilized:

- \texttt{max_iter} parameter is required, so the program will abort the computation as soon as number of iteration exceed the given value.
- \texttt{eps} parameter is required. If maximum of the message difference between two consecutive iterations will be less, than given value, then the program will stop.

Returning value

Like in many other optimization problems error may increase during computation, that is why the program will return a tree with the least expensive cost among all which was obtained until the last iteration.
3.3.2 Complete algorithm

Algorithm 3.1: BP algorithm for PCST problem for graph with cycles

**Input:** Graph, terminals, $\lambda$, root, depth, eps, max_iter

**Output:** Steiner Tree

```plaintext
/* initialize messages so that they satisfy (3.7)
create arrays $A^d_{i \rightarrow j}$, $E^d_{i \rightarrow j}$, $B_i \rightarrow j$, $D_i \rightarrow j$, $G_i$,
forall $ij \in E(G)$, forall $j \in V(G)$, $d \in \{1, ..., depth\}$

for $ij \in E(G)$:
    $A^0_{i \rightarrow j} = [\sum_{k=1}^{\lambda} E^0_{k \rightarrow j}(0), ..., \sum_{k=1}^{\lambda} E^0_{k \rightarrow j}(0)]$
    $B_i \rightarrow j(0) = -\lambda$
    $D_i \rightarrow j(0) = \max(B_i \rightarrow j(0), \max_{d \in \{1, ..., depth-1\}} A^d_{i \rightarrow j}(0))$

for $d \in \{1, ..., depth-1\}$:
    $C = -\lambda$
    $E^0_{i \rightarrow j}(0) = \max(C, D_i \rightarrow j(0))$
end

$E^{\text{depth}}_{i \rightarrow j} = D_i \rightarrow j(0)$

end

/* main loop
iter = 0
while TRUE:
    /* iteratively update messages using message update rules
    for $i \in \text{permute_vertices}(V(G))$:
        if $i$ is root:
            /* update root
            for $j \in N(i)$:
                $A_{i \rightarrow j}(iter + 1) = [0, -\infty, ..., -\infty]$
                $E_{i \rightarrow j}(iter + 1) = [0, ..., 0]$
                $B_i \rightarrow j(iter + 1) = -\infty$
                $D_i \rightarrow j(iter + 1) = 0$
            end
        else:
            /* update non-root vertex
            for $j \in N(i)$:
                $A^{d}_{i \rightarrow j}(iter + 1) = \sum_{k=1}^{\lambda} E^d_{k \rightarrow j}(iter) + \max_{k \in N(i)} A^{d-1}_{i \rightarrow j}(iter)$
                $B_i \rightarrow j(iter + 1) = -\lambda + \sum_{k \in N(i)} D_k \rightarrow j(\text{iter})$
                $D_i \rightarrow j(\text{iter + 1}) = \max(B_i \rightarrow j(\text{iter + 1}), \max_{d \in \{1, ..., depth\}} A^d_{i \rightarrow j}(\text{iter}) + 1)$
                $C = -\infty$
            for $d \in \{\text{depth}, ..., 2\}$:
                $E^d_{i \rightarrow j}(iter + 1) = \max(D_i \rightarrow j(\text{iter + 1}), C)$
                $C = -c_{ij} + \sum_{k \in N(i)} E^d_{k \rightarrow j}(\text{iter})$
                $F^d_{i \rightarrow j}(iter + 1) = C + A^{d-1}_{i \rightarrow j}(\text{iter})$
            end
            $E^1_{i \rightarrow j}(iter + 1) = D_i \rightarrow j(\text{iter + 1})$
            $F^1_{i \rightarrow j}(iter + 1) = -\infty$
        end
    end
end

$G_i(\text{iter + 1}) = -\lambda i + \sum_{k \in N(i)} D_k \rightarrow i(\text{iter})$
end
```

11
Algorithm 3.1 (cont.): BP algorithm for PCST problem for graph with cycles

```plaintext
/* retrieve tree from messages and compute its cost */
for i ∈ V(G):
    if i is not root:
        if max_k F_{i→k} > G_i:
            add edge (i, argmax_k F_{i→k}) to tree
    end
end

return least expansive tree
```

References

4 Workflow with the package

suppressMessages(library(pcSteiner))

g <- graph('Bull')

# Prize for 1-st node is 10
E(g)$costs <- c(3, 3, 3, 3, 3)
V(g)$prizes <- c(10, 2, 2, 2, 2)

terminals <- c(4, 5)

Run the analysis

treeData <- pcs.tree(
    graph=g,
    terminals=c(4,5),
    lambda=1,
    root=3,
    depth=5,
    eps=-1,
    max_iter=10
)

Plot graph

V(g)$color <- "gray"
V(g)$color[terminals] <- "red"

E(g)$color <- "gray"
E(g)$color[treeData$edges] <- "red"
plot(g)