‘pcSteiner’ vignette

Aleksei Krasikov

2020-01-14

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

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 $c : E \to R_+$. 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 $c : E \to R_+$
Output: a minimum weight tree connecting all vertices of T in G

Theorem 1.1. The Steiner tree problem is NP-complete, 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 represent 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 $c : E \to R_+$ and prizes $p : V \to R_+$
Output: find a tree G_1 by minimizing the following function $f(G_1) = \sum_{e \in E_1} c_e + \lambda \sum_{i \notin V_1} p_i$

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 $c : E \to R_+$ and prizes $p : V \to R_+$
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-complete. Thus, we do not expect to have an efficient algorithm for solving PCST problem.

References

1. B. Korte and J. Vygen, "Combinatorial Optimization. Theory and Algorithms". Springer, 2008.

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 accosiated 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:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid \mathbf{Par}(x_i)) \tag{2.1}$$

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:

$$P(x_1, ..., x_n) = Z^{-1} \prod_{C \in \mathcal{C}} \psi_C(x_C) \tag{2.2}$$

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 ψ_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.

Defenition 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. $f_i = f_i(\mathbf{Nb}(F_i))$.

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

$$P(\{x\}) = Z^{-1} \prod_{a=1}^{M} \psi(\{x\}_a) \tag{2.3}$$

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 $P(\{x\}_A \mid \{x\}_B)$, 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: $\text{argmax}_{\{x\}_A} P(\{x\}_A \mid \{x\}_B)$

Defenition 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:

$$P(\{x\}) = Z^{-1} \prod_{(ij)} \psi_{(ij)}(x_i, x_j) \tag{2.4}$$

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:

$$\begin{aligned} P(x_1) &= Z^{-1} \sum_{x_2,x_3,x_4} \psi_1(x_1, x_2)\psi_2(x_2, x_4)\psi_3(x_2, x_3) = \\ &= Z^{-1} \sum_{x_2} \psi_1(x_1, x_2) \left[\sum_{x_4} \psi_2(x_2, x_4)\right] \left[\sum_{x_3} \psi_3(x_2, x_3)\right] = \\ &= Z^{-1} \sum_{x_2} \psi_1(x_1, x_2)\mu_{4\to 2}(x_2)\mu_{3\to 2}(x_2) = \\ &= Z^{-1}\mu_{2\to 1}(x_1) \end{aligned} \tag{2.5}$$

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;
$\mu_{i\to j}(x_j) = \sum_{x_i} \psi_{(ij)}(x_i, x_j) \prod_{k\in \mathbf{Nb}(x_i)\setminus\{x_j\}} \mu_{k\to i}(x_i)$;
return marginal distribution $P(x_n) = Z^{-1} \prod_{i\in\mathbf{Nb}(x_n)} \mu_{i\to n}(x_n)$

Defenition 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;
$\mu_{i\to j}(x_j) = \max_{x_i} \psi_{(ij)}(x_i, x_j) \prod_{k\in \mathbf{Nb}(x_i)\setminus\{x_j\}} \mu_{k\to i}(x_i)$;
calculate max-marginals $\bar{p}_{x_i}(x_i) = \prod_{j\in\mathbf{Nb}(x_j)} \mu_{j\to i}(x_i)\ \forall i \in V$;
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:

$$\tilde{m}_{i\to j}(x_j) = \max_{x_i} \left\{ \ln \psi_{ij}(x_i, x_j) + \sum_{k \in \mathbf{Nb}(x_i)\setminus\{x_j\}} \tilde{m}_{k\to i}(x_i) \right\}$$
$$\bar{p}_{x_i}(x_i) = \exp \left\{ \sum_{j \in \mathbf{Nb}(x_i)} \tilde{m}_{k\to i}(x_i) \right\} \tag{2.6}$$

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 statets. If we know nothing about the structure of the joint probability, evaluation of desired marginal probability distribution would require $\mathcal{O}(k^{|V|})$ 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 $\mathcal{O}(|V|\ k^2)$.

So, BP algorithm is exact and efficient agorithm for performing inference in loop-free probabalistic 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_{max}
Output: MAP assignment
initialize all messages $\mu^{(0)}_{i\to j} = 1\ \forall (i,j) \in E$
for $t = 1, \ldots, t_{max}$ **do**
  $\tilde{m}^{t+1}_{i\to j}(x_j) = \max_{x_i} \left\{ \ln \psi_{ij}(x_i, x_j) + \sum_{k \in \mathbf{Nb}(x_i)\setminus\{x_j\}} \tilde{m}^{t}_{k\to i}(x_i) \right\}$
compute max-marginals $\bar{p}_{x_i}(x_i) = \exp \left\{ \sum_{j \in \mathbf{Nb}(x_i)} \tilde{m}^{t_{max}+1}_{k\to i}(x_i) \right\}$
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

1. D. Koller and N. Friedman, "Probabilistic Graphical Models: Principles and Techniques". The MIT Press, 2009.
2. J. Yedidia, W. Freeman and Y. Weiss, "Understanding belief propagation and its generalizations". 2001.
3. F. Pernkopf, R. Peharz and S. Tschiatschek, "Introduction to Probabilistic Graphical Models". 2014.

4. M. Wainwright and M. Jordan “Graphical Models, Exponential Families, and Variational Inference”. 2008.
5. C. Sutton and A. McCallum, “An Introduction to Conditional Random Fields”. 2012.
6. J. Yedidia, W. Freeman, “Constructing Free-Energy Approximations and Generalized Belief Propagation Algorithms”. 2005.
7. F. Kschischang, B. Frey and H. Loeliger, “Factor Graphs and The Sum-Product Algorithm”. IEEE Transactions on Information Theory, 2001.
8. C. Sutton and A. McCallum, “Improved dynamic schedules for belief propagation”. 2007.
9. G. Elidan, I. McGraw and D. Koller, “Residual belief propagation: Informed scheduling for asynchronous message passing”. 2006.
10. Y. Weiss, “Correctness of Local Probability Propagation in Graphical Models with Loops”. 2000.
11. F. Jensen, “An introduction to Bayesian networks. UCL Press Limited”. 1996.

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 $c : E \to R_+$, prizes $p : V \to R_+$, root $r \in V$, depth D
Output: r-rooted tree $G_1 = (V_1, E_1)$ with depth D which minimizes the following function $f(G_1) = \sum_{e \in E_1} c_e + \lambda \sum_{i \notin V_1} p_i$

3.1 Graphical model

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

- $d_i \in \{1, ..., D\}$ denotes the distance to the given root r
- $p_i \in \mathbf{Nb}(i) \cup \{*\}$, where $\mathbf{Nb}(i) = \{j : (ij) \in E\}$ and $p_i = \{*\} \Leftrightarrow i \notin V_1$. 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 $\forall (ij) \in E$:

$$\text{if } p_i = j \Rightarrow \begin{cases} d_i = d_j + 1 \\ p_j \neq \{*\} \end{cases} \tag{3.1}$$

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

$$g_{ij} = f_{ij} f_{ji} = 1 \tag{3.2}$$

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

$$\min\{\mathcal{H}(\mathbf{p}) = \sum_{i \in V} c_{ip_i} : (\mathbf{d}, \mathbf{p}) \in \mathcal{T}\} \tag{3.3}$$

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

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

$$P(\mathbf{d}, \mathbf{p}) = Z^{-1} \prod_{(ij) \in E} g_{ij} \prod_{i \in V} e^{-c_{ip_i}} \tag{3.4}$$

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):

$$m_{i\to j}(d_j,p_j)=\max_{d_i,p_i:g_{ij}=1}\left\{-c_{ip_i}+\sum_{k\in\mathbf{Nb}(i)\setminus j}m_{k\to i}(d_i,p_i)\right\} \tag{3.5}$$

Messages will be computed separately for each of the following groups of p_j:

$$\begin{aligned}
m_{i\to j}(d_j,p_j)&=\begin{cases}\max_{d_i,p_i:\ g_{ij}=1}\{.\}, & p_j=i\\ \max_{d_i,p_i:\ g_{ij}=1}\{.\}, & p_j=*\\ \max_{d_i,p_i:\ g_{ij}=1}\{.\}, & p_j\neq\{i,*\}\end{cases}\\
&=\begin{cases}\max_{d_i,p_i:\ g_{ij}=1\Leftrightarrow(d_i=d_j-1\wedge p_i\neq\{j,*\})}\{.\}, & p_j=i\\ \max_{d_i,p_i:\ g_{ij}=1\Leftrightarrow p_i\neq j}\{.\}, & p_j=*\\ \max_{d_i,p_i:\ g_{ij}=1\Leftrightarrow(d_i=d_j+1\wedge p_i=j)\vee p_i\neq\{j,*\}\vee p_i=*}\{.\}, & p_j\neq\{i,*\}\end{cases}\\
&=\begin{cases}\max_{d_i=d_j-1\wedge p_i\neq\{j,*\}}\{.\} & p_j=i\\ \max[\max_{d_i,p_i\neq\{j,*\}}\{.\};\max_{d_i,p_i=*}\{.\}] & p_j=*\\ \max\left[\max_{d_i=d_j+1\wedge p_i=j}\{.\};\max[\max_{d_i,p_i\neq\{j,*\}}\{.\};\max_{d_i,p_i=*}\{.\}]\right] & p_j\neq\{i,*\}\end{cases}\\
&=\begin{cases}\max_{d_i=d_j-1\wedge p_i\neq\{j,*\}}\{.\} & p_j=i\\ \max_{d_i}\max[\max_{p_i\neq\{j,*\}}\{.\};\max_{p_i=*}\{.\}] & p_j=*\\ \max\left[\max_{d_i=d_j+1\wedge p_i=j}\{.\};\max_{d_i}\max[\max_{p_i\neq\{j,*\}}\{.\};\max_{p_i=*}\{.\}]\right] & p_j\neq\{i,*\}\end{cases}
\end{aligned}$$

Introducing new variables $A^d_{i\to j}=\max_{p_i\neq\{j,*\}}\{.\}|_{d_i=d}$, $B^d_{i\to j}=\max_{p_i=*}\{.\}|_{d_i=d}$, $C^d_{i\to j}=\max_{p_i=j}\{.\}|_{d_i=d}$ allows us to rewrite messages in more convenient way:

$$\begin{aligned}
m_{i\to j}(d_j,p_j)&=\begin{cases}A^{d_j-1}_{i\to j} & p_j=i\\ \max_{d_i}\max[A^{d_i}_{i\to j};B^{d_i}_{i\to j}] & p_j=*\\ \max\left[C^{d_j+1}_{i\to j};\max_{d_i}\max[A^{d_i}_{i\to j};B^{d_i}_{i\to j}]\right] & p_j\neq\{i,*\}\end{cases}\\
&=\begin{cases}A^{d_j-1}_{i\to j} & p_j=i\\ D_{i\to j} & p_j=*\\ \max\left[C^{d_j+1}_{i\to j};D_{i\to j}\right] & p_j\neq\{i,*\}\end{cases}\\
&=\begin{cases}A^{d_j-1}_{i\to j} & p_j=i\\ D_{i\to j} & p_j=*\\ E^{d_j}_{i\to j} & p_j\neq\{i,*\}\end{cases}
\end{aligned} \tag{3.7}$$

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

$$\begin{aligned}
A^d_{i\to j}=\max_{p_i\neq\{j,*\}}\{.\}|_{d_i=d}&=\max_{p_i\neq\{j,*\}}\left\{-c_{ip_i}+\sum_{l\in\mathbf{Nb}(i)\setminus j}m_{l\to i}(d,p_i)\right\}\\
&=\max_{k\in\mathbf{Nb}(i)\setminus j}\left\{-c_{ik}+\sum_{l\in\mathbf{Nb}(i)\setminus j}m_{l\to i}(d,k)\right\}\\
&=\sum_{k\in\mathbf{Nb}(i)\setminus j}E^d_{k\to j}+\max_{k\in\mathbf{Nb}(i)\setminus j}\left\{-c_{ik}-E^d_{k\to j}+A^{d-1}_{k\to j}\right\}
\end{aligned} \tag{3.8}$$

$$
\begin{aligned}
B^d_{i\to j} = \max_{p_i=*}\{.\}\,|_{d_i=d} &= -c_{i*} + \sum_{k\in\mathbf{Nb}(i)\setminus j} m_{k\to i}(d,*) \\
&= -c_{i*} + \sum_{k\in\mathbf{Nb}(i)\setminus j} D_{k\to j}
\end{aligned}
\tag{3.9}
$$

$$
\begin{aligned}
C^d_{i\to j} = \max_{p_i=j}\{.\}\,|_{d_i=d} &= -c_{ij} + \sum_{k\in\mathbf{Nb}(i)\setminus j} m_{k\to i}(d,j) \\
&= -c_{ij} + \sum_{k\in\mathbf{Nb}(i)\setminus j} E^d_{k\to j}
\end{aligned}
\tag{3.10}
$$

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

$$
b(d_j,p_j) = -c_{jp_j} + \sum_{k\in\mathbf{Nb}(i)} m_{k\to j}(d_j,p_j) \tag{3.11}
$$

Consideration of different cases for p_j simplify the overall analysis:

$$
\begin{aligned}
b(d_j,p_j) &= \begin{cases} -c_{ji} + \sum_{k\in\mathbf{Nb}(i)} m_{k\to j}(d_j,i), & p_j = i \in \mathbf{Nb}(j) \\ -c_{j*} + \sum_{k\in\mathbf{Nb}(i)} m_{k\to j}(d_j,*), & p_j = \{*\} \end{cases} \\
&= \begin{cases} \sum_{k\in\mathbf{Nb}(i)} E^d_{k\to j} + \left(-c_{ji} - E^d_{i\to j} + A^{d-1}_{i\to j}\right), & p_j = i \in \mathbf{Nb}(j) \\ -c_{j*} + \sum_{k\in\mathbf{Nb}(i)} D_{k\to j}, & p_j = \{*\} \end{cases} \\
&= \begin{cases} F_{j\to i}, & p_j = i \in \mathbf{Nb}(j) \\ G_{j\to i}, & p_j = \{*\} \end{cases}
\end{aligned}
\tag{3.12}
$$

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 criterias are utilized:

- `max_iter` parameter is required, so the program will abort the computation as soon as number of iteration exceed the given value.
- `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

References

1. M. Bayati, C. Borgs, A. Braunstein, J. Chayes, A. Ramezanpour, and R. Zecchina, "Statistical Mechanics of Steiner Trees". PRL, 2008.
2. M. Bayati, A. Braunstein, and R. Zecchina, "A rigorous analysis of the cavity equations for the minimum spanning tree". Journal of Mathematical Physics, 2008.
3. I. Biazzo, A. Braunstein and R. Zecchina, "Performance of a cavity-method-based algorithm for the prize-collecting Steiner tree problem on graphs". PRL, 2012.
4. A. Braunstein, R. Zecchina, "Learning by Message Passing in Networks of Discrete Synapses". PRL, 2006.
5. A. Braunstein, A. Muntoni, "Practical optimization of Steiner Trees via the cavity method". ArXiv, 2016.

4 Workflow with the package

```
suppressMessages(library(pcSteiner))
#> Warning: package 'igraph' was built under R version 3.5.2

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

```
res_edges <- pcs.tree(
        graph=g,
        terminals=terminals,
        lambda=1,
        root=3,
        depth=5,
        eps=1e-04,
        max_iter=10,
        terminal_infty=10000
)
#> [1] "Number of edges in the tree:  4"
#> [1] "Tolerance:                    0"
#> [1] "Iterations:                   2"
#> [1] "Is connected:                 TRUE"
#> [1] "Cost of the tree:             12"
```

Plot graph

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

E(g)$color            <- "gray"
E(g)$color[res_edges] <- "red"
plot(g)
```