A rigorous analysis of the cavity equations for the minimum spanning tree

Mohsen Bayati,1 Alfredo Braunstein,2 and Riccardo Zecchina2,a

1Microsoft Research, One Microsoft Way, Redmond, Washington 98052, USA
2Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

(Received 27 June 2008; accepted 12 August 2008; published online 9 December 2008)

We analyze a new general representation for the minimum weight Steiner tree problem that translates the topological connectivity constraint into a set of local conditions, which can be analyzed by the so-called cavity equation techniques. For the limit case of the spanning tree, we prove that the fixed point of the algorithm arising from the cavity equations leads to the global optimum. © 2008 American Institute of Physics. [DOI: 10.1063/1.2982805]

I. INTRODUCTION

Given a graph with positive weights on the edges, the minimum weight Steiner tree (MST) problem consists of finding a tree of minimum weight that contains a given set of “terminal” vertices. Such construction may require the inclusion of some nonterminal nodes, which are called Steiner nodes. Besides its practical importance in many fields, MST is a basic optimization problem over networks, which lies at the root of computer science, being both NP complete and difficult to approximate. In statistical physics, the Steiner tree problem has similarities with basic models such as polymers and self-avoiding walks with a nontrivial interplay between local and global constraints, e.g., energy minimization versus global connectivity. In recent years, many algorithmic results have appeared, showing the efficacy of the cavity approach for optimization and inference problems defined over both sparse and dense random networks of constraints. These performances are understood in terms of factorization properties of the Gibbs measure over ground states, which can also be seen as the onset of correlation decay along the iterations of the cavity equations. Here we make a step further in this direction by presenting evidence for the exactness of the cavity approach for problems having an additional rigid global constraint that couples all variables. We show that the cavity approach can be used to derive a new algorithm for MST, which has exact fixed points in the limit case of the spanning tree. More specifically, we show how the analysis of the computational tree, which characterizes the evolution of the so-called cavity marginals, can be used to prove optimality.

II. DEFINITIONS AND PROBLEM STATEMENT

Consider an undirected simple graph $G=(V,E)$, with vertices $V=\{1,\ldots,n\}$ and edges $E$. Let each edge $\{i,j\}$ has weight $w_{ij}\in\mathbb{R}$. Denote the set of neighbors of each vertex $i$ in $G$ by $N(i)$. Let $U$ be a subset of vertices called terminals. A connected subgraph $T$ of $G$ is called Steiner tree if it has no cycle and contains all vertices of $U$. For the special case of $U=V$, the tree $T$ is called a spanning tree. The set of all Steiner trees of the graph $G$ with terminals $U$ is denoted by $\text{St}(G,U)$.

The weight of the Steiner tree $T$, denoted by $W_T$, is defined by $W_T=\sum_{\{i,j\}\in T}w_{ij}$. The MST, $T^*(U)$, is defined by $T^*(U)=\arg\min_{T\in\text{St}(G,U)} W_T$, and for spanning trees (when $U=V$), we drop the reference to $U$ and denote it by $T^*$. The goal of this paper is to present a belief propagation (BP) based algorithm for finding $T^*(U)$ and analyze it. Throughout the paper, we will assume that $T^*(U)$

---

aElectronic mail: riccardo.zecchina@polito.it.
is unique. If the optimum, $T^*(U)$, is not unique then the degeneracy can be lifted by a small random perturbation of the weights, which does not change the optimum tree.

III. ALGORITHM AND MAIN RESULT

In this section we explain the BP algorithm for finding the MST. Let us quickly explain the model. This is done in more details in Ref. 10.

A. The pointer-depth model

We model the Steiner tree problem as a rooted tree (such a construction is often associated with the term *arborescence*). Name the vertex $1 \in V$ as the root. Then each node $i$ is endowed with a pair of variables $(p_i, d_i)$, a pointer $p_i$ to some other nodes in the neighborhood $N(i)$ of $i$, and a depth $d_i \in \{1, \ldots, d_{\text{max}}\}$ defined as the distance from the root. Terminal nodes (vertices in $U$) must point to some other nodes in the final tree and hence $p_i \in N(i)$. The root node conventionally points to itself. Nonroot nodes either point to some other nodes in $N(i)$ if they are part of the tree (Steiner and terminal nodes) or just do not point to any node if they are not part of the tree (allowed only for nonterminals), a fact that we represent by allowing a “null” state for the pointer $p_i$. i.e., $p_i \in N(i) \cup \{\text{null}\}$. The depth of the root is set to zero, $d_1 = 0$ while for the other nodes in the tree the depths measure the distance from the root along the unique simple path from the node to the root.

In order to impose the global connectivity constraint for the tree we need to impose the condition that if $p_i = j$, then $d_i = d_j - 1$. This condition forbids cycles and guarantees that the pointers describe a tree. In building the BP equations, we need to introduce the characteristic functions $f_{ij} = f_{ij}(p_i, d_i, p_j, d_j)$, which impose such constraints over configurations of the decision variables $(p_i, d_i)$. For any edge $(i, j)$, we have the indicator function $f_{ij} = g_{ij}$, where $g_{ij}(p_j, d_j, p_k, d_k) = (1 - \delta_{p_j}(1 - \delta_{p_j, d_j - 1})(1 - \delta_{p_k, d_k})(1 - \delta_{p_k, d_k - 1}))$. Therefore any set of the decision variables $(p_i, d_i)$ that satisfies the condition $\prod_{(i,j) \in E} f_{ij}(p_i, d_i, p_j, d_j) = 1$ corresponds to a Steiner tree in $St(G, U)$.

B. BP equations and the algorithm

Let us define $w_{i, \text{null}} = \infty$ for any $i \in U$. Then the max-sum BP equations will be as follows:

$$
\psi_{j \rightarrow i}(d_j, p_j) = -w_{j|i} + \sum_{k \in j \setminus i} \phi_{k \rightarrow j}(d_j, p_j),
$$

for any $i \in U$. (1)

$$
\phi_{k \rightarrow j}(d_j, p_j) = \max_{d_k, p_k \in j \setminus i} \psi_{k \rightarrow j}(d_k, p_k).
$$

(2)

On a tree $\psi_{j \rightarrow i}(d_j, p_j)$ can be interpreted as the minimum cost change in removing a vertex $j$ with forced configuration $d_j, p_j$ from the subgraph with link $(i, j)$ already removed.

On a fixed point, one computes marginals $\psi_j$, $\phi_j$ and the BP guess of the optimum tree is given by arg max $\psi_j$.

For efficient implementation of Eqs. (1) and (2) we introduce the variables $A^d_{k \rightarrow j} = \max_{p_k \neq j, \text{null}} \psi_{k \rightarrow j}(d_k, p_k)$, $B^d_{k \rightarrow j} = \psi_{k \rightarrow j}(d, \text{null})$, $C^d_{k \rightarrow j} = \psi_{k \rightarrow j}(d, j)$, $D^d_{k \rightarrow j} = \max_{d} \max_{d_k, p_k} \{A^d_{k \rightarrow j}, B^d_{k \rightarrow j}, C^d_{k \rightarrow j}, D^d_{k \rightarrow j}\}$, and $E^d_{k \rightarrow j} = \max(C^d_{k \rightarrow j - 1}, D^d_{k \rightarrow j - 1})$. This is enough to compute $\phi_{k \rightarrow j}(d_j, p_j) = A^d_{k \rightarrow j - 1}, D_{k \rightarrow j}, E^d_{k \rightarrow j}$ for $p_j = k$, $p_j = \text{null}$, and $p_j \neq k$, null, respectively. Equations (1) and (2) can then be solved by repeated iteration of the following set of equations:

$$
A^d_{k \rightarrow j}(t + 1) = \sum_{k \in j \setminus i} E^d_{k \rightarrow j - 1}(t) + \max_{k \in j \setminus i} \{A^{d - 1}_{k \rightarrow j}(t) - E^d_{k \rightarrow j - 1}(t) - w_{jk}\},
$$

(4)
we say that the BP algorithm does not point analysis of these heuristics in literature. In general versions of the Steiner tree problem but in this paper we use the pointer-depth model. To the best of our knowledge, there is no rigorous introducing a link representation of the pointer variables. Notes:

Note 1: If the BP algorithm converges to $\{p_i, d_i\}$, the decision variables converge to $\{p_i, d_i\}$, i.e., there exists an integer $N\geq 0$, such that for all $t> N$ and all $i$: $p_i(t)=p_i$, $d_i(t)=d_i$.

Note 2: There are examples for which this BP algorithm does not converge and one needs to use some heuristics to make it converge. To the best of our knowledge, there is no rigorous analysis of these heuristics in literature.

### IV. Analysis

Before proving Theorem 1, we quickly review the notion of computation tree. Computation trees have been used in most of the previous analysis of the BP algorithms; see Refs. 11 for a list of those works.

#### A. Computation tree

For any $i \in V$, let $T_i^t$ be the $t$-level computation tree corresponding to $i$, defined as follows: $T_i^t$ is a weighted tree of height $t+1$, rooted at $i$. All tree nodes have labels from the set $\{1, \ldots, n\}$ according to the following recursive rules.

(a) The root has label $i$.
(b) The set of labels of the $\deg_C(i)$ children of the root is equal to $N(i)$.
(c) If $s$ is a nonleaf node whose parent has label $r$, then the set of labels of its children is $N(s)\setminus \{r\}$.

\begin{align*}
B_{j-i}(t+1) &= -w_{j\text{null}} + \sum_{k \in j\text{null}} D_{k-j}(t), \\
C_{j-i}(t+1) &= -w_{ji} + \sum_{k \in j\text{null}} E_{k-j}(t), \\
D_{j-i}(t) &= \max(\max_d A^d_{j-i}(t), B_{j-i}(t)), \\
E_{j-i}(t) &= \max(C^d_{j-i}(t), D_{j-i}(t)).
\end{align*}
**Notation:** We denote a vertex $u$ of the computation tree by $[u,i]$ if it has label $i$. We also denote root of the computation tree $T^i$ by $[\text{root},i]$.

Similar to the pointer-depth model in graph $G$, we assign to each nonleaf vertex $[v,j]$ of $T^i$ two decision variables $(p_v,d_v)$ with $p_v \in N([v,j])$ and $d_v \in \{1, \ldots, d_{\max}\}$. We call such an assignment valid if the following constraints are satisfied.

(a) If for two neighbors $[u,j], [v,k]$ in $T^i$, $p_u = [u,j]$ then $d_u = d_v + 1$.
(b) For any vertex $[u,j]$ in $T^i$ whose label is the same as the root in $G$ (i.e., $j=1$), then $d_u = 0$.

Now for any valid assignment, the subtree $T = \{(v,j),p_v\}$ is called an oriented spanning tree of the computation tree. Figure 1 shows a graph with one of its computation trees, and an oriented spanning tree on it. Denote the minimum weight oriented spanning tree (MWOST) of the computation tree $T^i$ by $T'(T^i)$. Similar argument as in Ref. 11 shows that iterations of Eqs. (4)–(8) can be seen as a dynamic programming procedure that finds the MWOST over the computation tree. Moreover, Lemma 1 that comes next without proof is analogous to Corollary 1 from Ref. 11.

**Lemma 1:** The BP algorithm, which is initialized with zero messages, solves the MWOST problem on the computation tree. In particular, for each vertex $i$ of $G$ the decision variables $(p_i(t),d_i(t))$ are exactly equal to the decision variables $(p_i,d_i)$ corresponding to the vertex $[\text{root},i]$ in $T'(T^i)$.

**Note 3:** Lemma 1 can be generalized to any unbalanced computation tree (a tree that is obtained from $T^i$ by removing a subset of vertices and all of their descendants) as well. For an unbalanced tree, there is a unique set of BP initial conditions that should be used instead of the zero messages. Lemma 1 holds for any model where BP is used and does not depend on the problems studied in this paper (see Refs. 12–14 for more details).

**Note 4:** We would like to point out that the main result holds for the BP algorithm with any initial condition and we assume zero initial condition just to simplify the calculations. For arbitrary initial condition, the BP algorithm runs over a slightly modified computation tree. The new computation tree is almost the same computation tree as $T^i$ except that the leaf edges of the tree have arbitrary weights and not $w_{ij}$’s from $G$.

**B. Proof of the main result**

Proof consists of two parts. First we will show that in case of convergence, the estimated MST $T(i)$ is a spanning tree. Next we will prove that this limit is, in fact, the minimum spanning tree.

**1. Limit is a spanning tree**

First we will show that the limit of the BP algorithm is a spanning tree.

**Lemma 2:** If the BP algorithm converges to $\{(p_i,d_i)\}$, then the set of edges $\{(i,p_i)\}$ is a spanning tree of $G$.

**Proof:** Let us denote the set of edges $\{(i,p_i)\}$ by $T$. Note that from Lemma 1 (and the note
after that) we obtain the following. Since BP algorithm converges to \( \{(p_i, d_i)\} \), then for any vertex \( i \) and any radius \( r_i \) one can find a large enough computation tree with root \([\text{root}, i]\) such that in the MWOST of that computation tree, all of the vertices within distance \( r_i \) of the root have decision variables that are dictated by \( \{(i, p_i)\} \). In other words, there exist a number \( N_i \) such that in the MWOST of the computation tree \( T_{\text{root}}^i \), any vertex \([u, j]\) with distance less than \( r_i \) from \([\text{root}, i]\) has \( d_{u, j} = d_i \) and \( p_u = p_{\ast, p_j} \in N([u, j]) \).

Now consider the MWOST \( T(T_{\text{root}}^i) \). It consists of many connected pieces. Let \( A_i \) be the connected component of \( T(T_{\text{root}}^i) \) that contains the \([\text{root}, i]\). Note that each edge of \( A_i \) corresponds to some \(([u, j], p_u)\) by definition. We list and prove a few properties about the subtree \( A_i \).

(i) \( A_i \) has bounded radius. All vertices of \( A_i \) are within distance at most \( 2d_{\text{max}} \) from \([\text{root}, i]\).

Proof: Consider the unique path \( P([u, j], [\text{root}, i]) \) in \( A_i \) that connect a vertex \([u, j]\) to the \([\text{root}, i]\). The depth variable along the path \( P([u, j], [\text{root}, i]) \) either always increases by 1 (thus \( |P([u, j], [\text{root}, i])| = d_{\text{max}} \)) or it always decreases by 1 until it reaches zero and then increases by 1 up to \( d_{\text{max}} \) (or \( |P([u, j], [\text{root}, i])| \leq 2d_{\text{max}} \)).

(ii) \( A_i \) has no duplicate vertex. No two vertices of \( A_i \) have the same labels from the set \( V \). That means no two vertices of the form \([u, j]\) and \([v, j]\) belong to \( A_i \).

Proof: Assume the contrary, then let \([u, j]\) and \([v, j]\) be two such vertices in \( A_i \) which have the smallest depth variables \( d_u = d_v \) [note that by property (i) both \([u, j]\) and \([v, j]\) are within distance \( 2d_{\text{max}} \) of \([\text{root}, i]\) which shows \( d_u = d_v = d_i \)]. First assume \( j \neq 1 \). Consider the vertices of the computation tree that are pointed to by \([u, j]\) and \([v, j]\) (i.e., \( p_u = [u', p_j] \) and \( p_v = [v', p_j] \)). By design both \([u', p_j]\) and \([v', p_j]\) belong to \( A_i \) since they are connected to \([u, j]\) and \([v, j]\), respectively, and \( d_{u'} = d_u - 1 = d - 1 \) and \( d_{v'} = d_v - 1 = d - 1 \). Hence we should have \([u', p_j] = [v', p_j] \) (by definition of \([u, j]\) and \([v, j]\) that have smallest value for \( d_u = d_v \)). However, this means that the vertex \([u', p_j]\) of the computation tree has two distinct neighbors \([u, j]\) and \([v, j]\) with the same label, which is a contradiction because the computation tree and \( G \) have the same local structure at any nonleaf vertex. The case \( j = 1 \) is trivially impossible since the depth variable along the path between \([u, j]\) and \([v, j]\) should go from zero to zero.

(iii) \( A_i \) has all labels from \( V \). First note that \( A_i \) has a vertex with label 1. Because starting form \( i \) and following the pointers, the depth variable decreases and it becomes zero at some point. That vertex which has a depth of zero is in \( A_i \) and has to have label 1. Now we show that for any \( j \in V \), there exists a vertex \([u, j]\) \in \( A_i \). Consider the sequence \( S = j, p_j, p_{p_j}, p_{p_{p_j}}, \ldots \). This sequence has to stop at 1 since the depth variable for elements of the sequence is strictly decreasing. So it eventually intersects labels that appear in \( A_i \). Consider the first time that the intersection happens (for an element \([u, k]\) of \( A_i \) we have \( k \in S \)). If \( k \) is the element before \( k \) in \( S \) (i.e., \( p_k = k \)). We prove that \( \ell \) is also a label in \( A_i \). This is because \([u, k]\) has the same local structure in the computation tree as \( k \) in \( G \) and \( \ell \) is a neighbor of \( k \) in \( G \). Thus there exists a \([v, \ell]\) \in \( N([u, k]) \) and the distance between \([v, \ell]\) and \([\text{root}, i]\) is at most \( 2d_{\text{max}} + 1 \). So \( p_u = p_{\ell} \) and \( d_{u, \ell} = d_i \). This means that \([v, \ell]\) is connected to \([u, k]\) and hence is in \( A_i \). Repeating the process, we obtain that \( j \) is a label in \( A_i \).

Properties (i)–(iii) show that under \([u, j] \rightarrow j \) the tree \( A_i \) is isomorphic to \( T = \{(p_i, d_i)\} \) and therefore \( T \) is a spanning tree of \( G \).

**C. Limit is the minimum weight spanning tree**

To prove that the set \( T = \{(p_i, d_i)\} \) is the minimum spanning tree we assume the contrary \( (T \neq T) \). Then we will construct an oriented spanning tree \( T(T_{\text{root}}^i) \) that has less weight than \( T(T_{\text{root}}^i) \) which is a contradiction.

For our proof, we need to give a quick review of Prim’s well-known algorithm\(^{15}\) for finding the minimum spanning tree of graph \( G \). The algorithm continuously increases the size of a tree starting with a single vertex until it spans all the vertices. It starts from an initial subtree \( T_0 \) of \( G \) that contains a single vertex. Then for any \( r = 0, 1, \ldots, n - 2 \) the following step is repeated: find the minimum weight edge \((u, v)\) that connects \( T_r \) to \( G \setminus T_r \) and set \( T_{r+1} = T_r \cup \{(u, v)\} \). The tree \( T_{n-1} \) is the minimum spanning tree.
Assume that the Prim algorithm starts with vertex 1. Let $e_1, e_2, \ldots, e_{n-1}$ be the order of the edges that are added during the algorithm. That is, $T_i = \{e_1, e_2, \ldots, e_{n-1}\}$. Now let $e_k$ be the first edge that does not belong to $T$. The subgraph $T \cup \{e_k\}$ has a cycle. Thus it has an edge $e$ in $T$ that connects $T_{k-1}$ to outside of $T_{k-1}$. By Prim’s algorithm, $w(e) < w(e_k)$. The inequality is strict since $T^*$ is unique.

Let $T' = (T \cup \{e_k\}) \backslash \{e\}$. It is not hard to see that $T'$ is also a spanning tree of $G$ and $w(T') < w(T)$. Consider the pointer-depth representation for the tree $T'$ and denote the corresponding decision variables by $\{(p_i, d_i)\}$. Let also $(x, p_i)$ corresponds to the edge $e$ in this new pointer-depth representation. Since $1 \in T_k \subset T \cap T'$, then for any $i \in T_k$ we have $(p_i, d_i) = (p_i', d_i')$.

Now we consider the oriented spanning tree $\mathcal{T}(T_i')$ which contains the [root, i]. Let $[u, x] \in A_i$ be the unique vertex that has label $x$. We will change the decision variables of any vertex $[v, j]$ of $A_i$ from $(p_v, d_v)$ to $(p_v', d_v')$, where $p_v'$ is the unique vertex in $N([v, j])$ that has label $p_j$. Denote the new subgraph of the computation tree by $\mathcal{T}(T_i')$. Clearly, $w(\mathcal{T}(T_i')) < w(\mathcal{T}(T_i))$. Now we only need to show that $\mathcal{T}(T_i')$ is an oriented spanning tree of the computation tree to achieve a contradiction.

Since $T' \backslash T = \{e\}$, therefore we only need to check that local constraints at edge $([u, x], p_{u}')$ of $\mathcal{T}(T_i')$ satisfy the ones of an oriented spanning tree. Note that all neighbors of the vertex $[u, x]$ are within the distance $2d_{\max} + 1$ of [root, i]. Thus if, $p_{u}' = [v, p_{v}']$, then $(p_v, d_v)$ will be equal to $(v, p_{u}', d_{u}')$. On the other hand, $p_{v}'$ is a vertex in $T_k$, and for all vertices of $T_k$ the decision variables $(p, d)$ and $(p', d')$ are the same. Thus $(p_{u}', d_{u}')$ and $(p_{v}', d_{v}')$ will satisfy the local constraints since $(p_{u}', d_{u}')$ and $(p_{v}', d_{v}')$ satisfy the same constraint in $T'$. Therefore we obtained a new oriented spanning tree of the computation tree which has weight less than the optimum, $\mathcal{T}(T_i')$, which is a contradiction. So the assumption $T \neq T^*$ was incorrect.

ACKNOWLEDGMENTS

Mohsen Bayati acknowledges the support of the Theory Group at Microsoft Research and Microsoft Technical Computing Initiative.