arXiv:1710.07600v2 [stat.ML] 12 Jul 2018

Belief Propagation Min-Sum Algorithm for Generalized Min-Cost Network Flow

Andrii Riazanov[1], Yury Maximov[2] and Michael Chertkov[3]

***Abstract*— Belief Propagation algorithms are instruments used broadly to solve graphical model optimization and statistical inference problems. In the general case of a loopy Graphical Model, Belief Propagation is a heuristic which is quite successful in practice, even though its empirical success, typically, lacks theoretical guarantees. This paper extends the short list of special cases where correctness and/or convergence of a Belief Propagation algorithm is proven.**

We generalize formulation of Min-Sum Network Flow problem by relaxing the flow conservation (balance) constraints and then proving that the Belief Propagation algorithm converges to the exact result.

I. INTRODUCTION

Belief Propagation algorithms were designed to solve optimization and inference problems in graphical models. Since a variety of problems from different fields of science (communication, statistical physics, machine learning, computer vision, signal processing, etc.) can be formulated in the context of graphical model, Belief Propagation algorithms are of great interest for research during the last decade [1], [2]. These algorithms belong to message-passing heuristic, which contains distributive, iterative algorithms with little computation performed per iteration.

There are two types of problems in graphical models of the greatest interest: computation of the marginal distribution of a random variable, and finding the assignment which maximizes the likelihood. Sum-product and Min-sum algorithms were designed for solving these two problems under the heuristic of Belief Propagation. Originally, the sum-product algorithm was formulated on trees ([3], [4], [5]), for which this algorithm represents the idea of dynamic programming in the message-passing concept, where variable nodes transmit messages between each other along the edges of the graphical model. However, these algorithms showed surprisingly good performance even when applied to the graphical models of non-tree structure ([6], [7], [8], [9], [10]). Since Belief Propagation algorithms can be naturally implemented and paralleled using the simple idea of message passing, these instruments are widely used for finding approximate solutions to optimization of inference problems.

However, despite the good practical performance of this heuristic in many cases, the theoretical grounding of these algorithm remain unexplored (to some extend), so there are no actual proofs that the algorithms give correct (or even approximately correct) answers for the variety of problem statements. That's why one of important tasks is to explore the scope of the problems, for which these algorithms indeed can be applied, and to justify their practical usage.

In [11] the authors proved that the Belief Propagation algorithm give correct answers for Min-Cost Network Flow problem, regardless of the underlying graph being a tree or not. Moreover, the pseudo-polynomial time convergence was proven for these problems, if some additional conditions hold (the uniqueness of the solution and integral input). This work significantly extended the set of problems, for which Belief Propagation algorithms are justified.

In this paper we formulate the extension of Min-Cost Network Flow problem, which we address as Generalized Min-Cost Network Flow problem (GMNF). This problem statement is much broader then the original formulation, but we amplify the ideas of [11] to prove that Belief Propagation algorithms also give the correct answers for this generalization of the problem. This extension might find a lot of applications in various fields of study, since GMNF problem is the general problem of linear programming with additional constraints on the cycles of the underlying graphs (more precise, on the coefficient of corresponding vertices), which might be natural for some practical formulations.

II. GENERALIZED MIN-COST NETWORK FLOW

A. Problem statement

Let $G = (V, E)$ be a directed graph, where V is the set of vertices and E is the set of edges, $|V| = n, |E| = m$. For any vertex v we denote E_v as the set of edges incident to v, and a_v^e is the coefficient related to this pair (v, e), such that $a_v^e > 0$ if e is an out-arc with respect to v (e.g. $e = (v, w)$ for some vertex w), and $a_v^e < 0$ if e is an in-arc with respect to v (e.g. $e = (w, v)$ for some vertex w).

For any vertex v and edges e_1, e_2 incident to v we define $\delta(v, e_1, e_2) \triangleq \left|\frac{a_v^{e_1}}{a_v^{e_2}}\right|$. Then we consider the following property of the graph:

Definition 2.1: The graph G is called *ratio-balanced* if for every non-directed cycle C which consists of vertices

[1]Andrii Riazanov is with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA. riazanov@cs.cmu.edu

[2]Yury Maximov is with Skolkovo Institute of Science and Technology, Center for Energy Systems, and Los Alamos National Laboratory, Theoretical Division T-4 & CNLS, Los Alamos, NM 87544, USA yury@lanl.gov

[3]Michael Chertkov is with Skolkovo Institute of Science and Technology, Center for Energy Systems, and Los Alamos National Laboratory, Theoretical Division T-4, Los Alamos, NM 87544, USA chertkov@lanl.gov

$\{v_1, v_2, v_3, \ldots, v_k\}$ and edges $\{e_1, e_2, e_3, \ldots, e_k\}$ it holds:

$$\prod_{i=1}^{k} \delta(v_i, e_i, v_{i-1}) = \delta(v_1, e_k, e_1) \cdot \delta(v_2, e_1, e_2) \cdot \ldots \cdot \times$$
$$\times \delta(v_{k-1}, e_{k-2}, e_{k-1}) \cdot \delta(v_k, e_{k-1}, e_k) = 1. \qquad (1)$$

Here by non-directed cycle we mean that for every pair (v_i, v_{i-1}) and the pair (v_1, v_k) it holds that either $(v_i, v_{i-1}) \in E$ or $(v_{i-1}, v_i) \in E$. It is not hard to verify that it suffices for equation (1) to hold only for every *simple* non-directed cycle of G, since then the equation (1) can be easily deduced to hold for arbitrary non-directed cycle.

To check that the given graph is ratio-balances, one then need to check whether (1) holds for any simple cycle. If m is the number of edges, and C is the number of simple cycles of G, one obviously needs at least $O(m+C)$ time to iterate trough all simple cycles. In fact, the optimal algorithm for this task was introduced in [12], which runs for $O(m+C)$ time. Then, to check whether a graph is ratio-balances, one may use this algorithm to iterate through all simple cycles and to check (1) for every one of them.

We formulate the Generalized Min-Cost Network Flow problem for ratio-balanced graph G as follows:

$$\begin{aligned} &\texttt{minimize} && \sum_{e \in E} c_e x_e \\ &\texttt{subject to} && \sum_{e \in E_v} a_v^e x_e = f_v, \quad \forall v \in V, \\ & && 0 \le x_e \le u_e, \quad \forall e \in E. \end{aligned} \qquad \text{(GMNF)}$$

Here the first set of constrains are *balance constraints* which must hold for each vertex. The second set of constrains consists of *capacity constraints* on each edge of G. Coefficients c_e and u_e, defined for each edge $e \in E$, are called the *cost* and the *capacity* of the edge, respectively. Any assignment of x in this problem which satisfies the balance and capacity constraints is referred as *flow*. Finally, the objective function $g(x) = \sum_{e \in E} c_e x_e$ is called *the total flow*.

B. Definitions and properties

For the given (GMNF) problem on the graph G and flow x on this graph, the *residual network* $G(x)$ is defined as follows: $G(x)$ has the same vertex set as G, and for each edge $e = (v, w) \in E$ if $x_e < u_e$ then e is an arc in $G(x)$ with the cost $c_e^x = c_e$ and coefficients $(a_v^e)^x = a_v^e$, $(a_w^e)^x = a_w^e$. Finally, if $x_e > 0$ then there is an arc $e' = (w, v)$ in $G(x)$ with the cost $c_{e'}^x = -c_e$ and coefficients $(a_v^e)^x = -a_v^e$, $(a_w^e)^x = -a_w^e$. It is not hard to see that $G(x)$ is ratio-balanced whenever G is, since only the absolute values of the coefficients a_v^e occur in the definition of this property.

Then for each directed cycle $C = (\{v_1, v_2, v_3, \ldots, v_k\}, \{e_1, e_2, e_3, \ldots, e_k\})$ we define the cost of this cycle as follows:

$$c(C) \triangleq c_1 + \delta(v_2, e_1, e_2) \times$$
$$\times \left(c_2 + \delta(v_3, e_2, e_3) \left(c_3 + \cdots + \delta(v_k, e_{k-1}, e_k) c_k \right) \cdots \right) =$$
$$= c_1 + \sum_{i=2}^{k} c_i \prod_{j=2}^{i} \delta(v_j, e_{j-1}, e_j)$$

It is easy to see that $c(C)$ is properly defined whenever the graph G is ratio-constrained.

Then we define $\sigma(x) \triangleq \min_C \{c^x(C)\}$, where the minimum is taken over all cycles C in the residual network $G(x)$.

Lemma 2.1: If (GMNF) has a unique solution x^*, then $\sigma(x^*) > 0$.

Proof: We will show that for every directed cycle C from $G(x^*)$ we can push the additional flow through the edges of this cycle such that the linear constraints in (GMNF) will still be satisfied, but the total flow in the cycle will change by $\varepsilon \cdot c^x(C)$ for some $\varepsilon > 0$.

Let $C = (\{v_1, v_2, v_3, \ldots, v_k\}, \{e_1, e_2, e_3, \ldots, e_k\})$. From the definition of the residual network it follows that we can increase the flow in every edge of the cycle for some positive quantity such that the capacity constraints will still be satisfied. Let's push additional flow $\varepsilon > 0$ trough e_1. In order to satisfy the balance constraint in v_1, we need to adjust the flow in e_2. We have 4 cases: either one of e_1, e_2 can be in G, or their opposites can be in G. If $e \in C$ is in G, we will say that it is 'direct' arc, otherwise it will be 'opposite'. Then there are four cases:

1) e_1, e_2 are direct arcs. Then we have the new flow on edge $e-1$: $y_1 = x_1^* + \varepsilon$. In order to satisfy the balance constraint for the vertex v_2, is must hold $a_{v_2}^{e_1} x_1^* + a_{v_2}^{e_2} x_2^* = a_{v_2}^{e_1} y_1 + a_{v_2}^{e_2} y_2 = a_{v_2}^{e_1} x_1^* + a_{v_2}^{e_1} \varepsilon + a_{v_2}^{e_2} y_2 \Rightarrow a_{v_2}^{e_2}(x_2^* - y_2) = a_{v_2}^{e_1} \varepsilon \Rightarrow y_2 = x_2^* - \varepsilon \frac{a_{v_2}^{e_1}}{a_{v_2}^{e_2}}$. Since both e_1 and e_2 are direct, it means that $e_1 = (v_1, v_2) \in E$, and $e_2 = (v_2, v_3) \in E$, and thus, by definition, $a_{v_2}^{e_1} < 0$, and $a_{v_2}^{e_2} > 0$. Therefore, we have $y_2 = x_2^* + \varepsilon \delta(v_2, e_1, e_2)$.
2) e_1 is direct, e_2 is opposite. Then again, $y_1 = x_1^* + \varepsilon$, but now 'pushing' the flow through e_2 (as the edge in the residual network) means decreasing x_2. The same equalities holds, so $y_2 = x_2^* - \varepsilon \frac{a_{v_2}^{e_1}}{a_{v_2}^{e_2}} = x_2^* - \varepsilon \delta(v_2, e_1, e_2)$. Since e_2 is opposite arc, that means that we should push additional $\varepsilon \delta(v_2, e_1, e_2)$ through e_2.
3) e_1 is opposite, e_2 is direct – similar to the case 2).
4) e_1, e_2 are opposite arcs – similar to the case 1).

So, if we push ε through e_1, we need to push $\varepsilon_2 = \varepsilon \delta(v_2, e_1, e_2)$ through e_2 to keep the balance in v_2. Then, analogically, to maintain the balance in v_3, we need to push additional $\varepsilon_3 = \varepsilon_2 \delta(v_3, e_2, e_3) = \varepsilon \delta(v_2, e_1, e_2) \delta(v_3, e_2, e_3)$

through e_3. Then, consequently adjusting the balance in all the vertexes of C, we will retrieve that to keep the balance in v_k, we need to push $\varepsilon_k = \varepsilon_{k-1}\delta(v_k, e_{k-1}, e_k) = \varepsilon\prod_{i=2}^{k}\delta(v_i, e_{i-1}, e_i)$ through e_k. Now it suffices to show that the balance in v_1 is also satisfied. Similarly, we know that if we push ε_k in e_k, then we need to push $\varepsilon_1 = \varepsilon_k\delta(v_1, e_k, e_1) = \varepsilon\prod_{i=1}^{k}\delta(v_i, e_{i-1}, e_1) = \varepsilon$ (since G is ratio-balanced) through e_1, and that is exactly the amount which we assumed to push at the beginning of this proof. So we indeed push consistent flow through all the edges of C in such a way that the balance constraints in all the vertices is satisfied. We now only need to mention that we can take $\varepsilon > 0$ as small as it is needed to satisfy also all the capacity constraints in the cycle.

Now the additional total cost of such adjusting will be

$$\sum_{i=1}^{k} c_i^x\varepsilon_i = c_1^x\varepsilon + c_2^x\varepsilon\delta(v_2, e_1, e_2) +$$

$$+c_2^x\varepsilon\delta(v_2, e_1, e_2)\delta(v_3, e_2, e_3) + \cdots + c_k^x\varepsilon\prod_{i=2}^{k}\delta(v_i, e_{i-1}, e_i) =$$

$$= \varepsilon\cdot c^x(C)$$

Now it is obvious that if $c^{x^*}(C) \le 0$ for some C, we can change the flow in G such that the total cost will not increase. It means that either x^* is not an optimal flow, or it is not the unique solution of (GMNF). ■

Next we define the cost of a directed path in G or $G(x)$:

Definition 2.2: Let $S = (\{v_1, \cdots, v_k\}, \{e_1, \cdots, e_{k-1}\})$ be a directed path. Then the cost of this path is defined as

$$l(S) \triangleq c_1 + \delta(v_2, e_1, e_2)\Bigg(c_2 + \delta(v_3, e_2, e_3)\times$$

$$\times\Bigg(\cdots\big(c_{k-2} + \delta(v_{k-1}, e_{k-2}, e_{k-1})c_{k-1}\big)\cdots\Bigg)\Bigg) =$$

$$= c_1 + \sum_{i=2}^{k-1} c_i\prod_{j=2}^{i}\delta(v_j, e_{j-1}, e_j)$$

We also define the 'reducer' of the path as follows:

$$t(S) \triangleq \min_{j=2,\ldots,k-1}\prod_{i=2}^{j}\delta(v_i, e_{i-1}, e_i)$$

To prove the main result of this paper we will use the following crucial lemma:

Lemma 2.2: Let G be any ratio-balanced graph, or a residual network of some ratio-balanced graph (as we already mentioned, the residual network will also be ratio-balanced in this case). Let $S = (\{v_1, \cdots, v_k\}, \{e_1, \cdots, e_{k-1}\})$ be a directed path in G, and $C = (\{v'_1, v'_2, \cdots, v'_m\}, \{e'_1, e'_2, \cdots, e'_m\})$ be a cycle with $v'_1 = v_p$. Let R be the path $R = \{v_1, v_2, \cdots, v_p = v'_1, v'_2, \ldots, v'_m, v'_1 = v_p, v_{p+1}, \ldots, v_k\}$. Then $l(R) \ge l(S) + Tc(C)$, where $T = \min_S t(S)$ is the minimum of all the reducers among all directed paths S in G.

Proof:

$$l(R) = c_1 + \delta(v_2, e_1, e_2)\Bigg(c_2 + \delta(v_3, e_2, e_3)\times$$

$$\times\Bigg(\cdots\big(c_{k-2} + \delta(v_{k-1}, e_{k-2}, e_{k-1})c_{k-1}\big)\cdots\Bigg)\Bigg) =$$

$$= \left(c_1 + \sum_{i=2}^{p-1}\left[c_i\prod_{j=2}^{i}\delta(v_j, e_{j-1}, e_j)\right]\right) +$$

$$+\underbrace{\left(\prod_{j=2}^{p-1}\delta(v_j, e_{j-1}, e_j)\right)\cdot\delta(v_p, e_{p-1}, e'_1)\times}_{\ge T}$$

$$\times\underbrace{\left(c'_1 + \sum_{i=2}^{m}\left[c'_i\prod_{j=2}^{i}\delta(v'_j, e'_{j-1}, e'_j)\right]\right)}_{c(C)} +$$

$$+\left(\prod_{j=2}^{p-1}\delta(v_j, e_{j-1}, e_j)\right)\cdot\delta(v_p, e_{p-1}, e'_1)\times$$

$$\times\underbrace{\left(\prod_{j=2}^{m}\delta(v'_j, e'_{j-1}, e'_j)\right)}_{(\delta(v'_1, e'_m, e'_1))^{-1}}\cdot\delta(v_p, e'_m, e_p)\times$$

$$\times\left(c_p + \sum_{i=p+1}^{k-1}\left[c_i\prod_{j=p+1}^{i}\delta(v_j, e_{j-1}, e_j)\right]\right) \ge$$

$$\ge \left(c_1 + \sum_{i=2}^{p-1}\left[c_i\prod_{j=2}^{i}\delta(v_j, e_{j-1}, e_j)\right]\right) +$$

$$+\left(\prod_{j=2}^{p-1}\delta(v_j, e_{j-1}, e_j)\right)\cdot\underbrace{\left|\frac{a_{v_p}^{e_{p-1}}}{a_{v_p}^{e'_1}}\right|\left|\frac{a_{v'_1}^{e'_1}}{a_{v'_1}^{e'_m}}\right|\left|\frac{a_{v_p}^{e'_m}}{a_{v_p}^{e_p}}\right|}_{\delta(v_p, e_{p-1}, e_p)}\times$$

$$\times\left(c_p + \sum_{i=p+1}^{k-1}\left[c_i\prod_{j=p+1}^{i}\delta(v_j, e_{j-1}, e_j)\right]\right) +$$

$$+Tc(C) =$$

$$= \left(c_1 + \sum_{i=2}^{p-1}\left[c_i\prod_{j=2}^{i}\delta(v_j, e_{j-1}, e_j)\right]\right) +$$

$$+\prod_{j=2}^{p}\delta(v_j, e_{j-1}, e_j)\times$$

$$\times\left(c_p + \sum_{i=p+1}^{k-1}\left[c_i\prod_{j=p+1}^{i}\delta(v_j, e_{j-1}, e_j)\right]\right) + Tc(C) =$$

$$= c_1 + \sum_{i=2}^{p-1}\left[c_i \prod_{j=2}^{i} \delta(v_j, e_{j-1}, e_j)\right] +$$

$$+ \sum_{i=p}^{k-1}\left[c_i \prod_{j=2}^{i} \delta(v_j, e_{j-1}, e_j)\right] + Tc(C) =$$

$$= l(S) + Tc(C)$$

■

III. Belief Propagation algorithm for GMNF

A. Min-Sum algorithm

Algorithm 1 represents the Belief Propagation Min-Sum algorithm for (GMNF) from [11]. In the algorithm the functions $\phi_e(z)$ and $\psi_v(z)$ are the variable and factor functions, respectively, defined for $v \in V, e \in E$ as follows:

$$\phi_e(z) = \begin{cases} c_e z_e & \text{if } 0 \le z_e \le u_e, \\ +\infty & \text{otherwise.} \end{cases}$$

$$\psi_v(z) = \begin{cases} 0 & \text{if } \sum_{e \in E_v} a_v^e z_e = f_v, \\ +\infty & \text{otherwise.} \end{cases}$$

Algorithm 1 BP for (GMNF)

1: Initialize $t = 0$, messages $m^0_{e\to v}(z) = 0,\ m^0_{e\to w}(z) = 0,\ \forall z \in \mathbb{R}$ for each $e = (v, w) \in E$.
2: **for** $t = 1, 2, \dots N$ **do**
3: For each $e = (v, w) \in E$ update messages as follows:

$$m^t_{e\to v}(z) = \phi_e(z) + \\ + \min_{\vec{z}\in\mathbb{R}^{|E_w|}, \vec{z}_e = z}\left\{\psi_w(\vec{z}) + \sum_{\tilde{e}\in E_w\setminus e} m^{t-1}_{e\to w}(\vec{z}_{\tilde{e}})\right\}, \quad \forall z \in \mathbb{R}$$

$$m^t_{e\to w}(z) = \phi_e(z) + \\ + \min_{\vec{z}\in\mathbb{R}^{|E_v|}, \vec{z}_e = z}\left\{\psi_v(\vec{z}) + \sum_{\tilde{e}\in E_v\setminus e} m^{t-1}_{e\to v}(\vec{z}_{\tilde{e}})\right\}, \quad \forall z \in \mathbb{R}$$

4: t := t + 1
5: **end for**
6: For each $e = (v, w) \in E$, set the belief function as

$$b^N_e = \phi_e(z) + m^N_{e\to v}(z) + m^N_{e\to w}(z)$$

7: Calculate the belief estimate by finding $\hat{x}^N_e \in \arg\min b^N_e(z)$ for each $e \in E$.
8: Return $\hat{x}^N$ as an estimation of the optimal solution of (GMNF).

We address the reader to the article [11] for more details, intuition and justifications on the Belief Propagation algorithm for general optimization problems, linear programs, or Min-Cost Network Flow in particular.

B. Computation trees

One of the important notions used for proving correctness and/or convergence for BP algorithm is the *computation tree* ([11], [13], [14], [15]) (unwrapped tree in some sources). The idea under this construction is the following: for the fixed edge e of the graph G, one might want to build a tree of depth N, such that performing N iterations of BP on graph G gives the same estimation of flow on e, as the optimal solution of the appropriately defined (GMNF) problem on the computation tree T^N_e.

Since the proof of our result is based on the computation trees approach, in this subsection we describe the construction in details. We will use the same notations for computation tree, as in [11] (section 5).

In this paper we consider the computation trees, corresponding to edges of G. We say that $e \in E$ is the "root" for N-level computation tree T^N_e. Each vertex or edge of T^N_e is a duplicate of some vertex or edge of G. Define the mapping $\Gamma^N_e : V(T^N_e) \to V(G)$ such that if $v' \in V(T^N_e)$ is a duplicate of $v \in V(G)$, then $\Gamma^N_e(v') = v$. In other words, this function maps each duplicate from $V(T^N_e)$ to its inverse in $V(G)$.

The easiest way to describe the construction is inductively. Let $e = (v, w) \in E(G)$. Then the tree T^0_e consists of two vertices v', w', such that $\Gamma^0_e(v') = v, \Gamma^0_e(w') = w$, and an edge $e' = (v', w')$. We say that v', w' belong to 0-level of T^0_e. Note that for any two vertices $v', w' \in V(T^0_e)$ it holds that $(v', w') \in E(T^0_e) \Leftrightarrow (\Gamma^0_e(v'), \Gamma^0_e(w')) \in E(G)$, so the vertices in a tree are connected with an edge if and only if their inverse in the initial graph are connected. This property will hold for all trees T^N_e. Now assume that we defined a tree T^N_e, such that for any $v', w' \in V(T^N_e)$ it holds that $(v', w') \in E(T^N_e) \Leftrightarrow (\Gamma^N_e(v'), \Gamma^N_e(w')) \in E(G)$. Denote by $L(T^N_e)$ the set of leafs of T^N_e (vertices which are connected by edge with exactly one another vertex). For any $u' \in L(T^N_e)$, denote by $P(u')$ the vertex, with which u' is connected by edge (so either $(u', P(u')) \in E(T^N_e)$ or $(P(u'), u') \in E(T^N_e)$). We now build T^{N+1}_e by extending the three T^N_e as follows: for every $u' \in L(T^N_e)$ let $u = \Gamma^N_e(u')$, and consider the set $B_{u'} = S_u \setminus \{\Gamma^N_e(P(u'))\}$, where S_u is the set of neighbors of u in G. Then for every vertex $w \in B_{u'}$ add vertex w' to expand $V(T^N_e)$ and an edge (u', w') if $(u, w) \in E$ or an edge (w', u') if $(w, u) \in E$ to expand $E(T^N_e)$, and set $\Gamma^{N+1}_e(w') = w$. Also set the level of w' to be equal $(N + 1)$.

So, the tree T^{N+1}_e contains T^N_e as an induced subtree, and also contains vertices on level $N + 1$, which are connected to leafs of T^N_e (in fact, it is easy to see that new vertices are connected only with leafs from N^{th} level). From the construction, one may see that for any $v', w' \in V(T^{N+1}_e)$ it holds that $(v', w') \in E(T^{N+1}_e) \Leftrightarrow (\Gamma^{N+1}_e(v'), \Gamma^{N+1}_e(w')) \in E(G)$. In fact, any vertex of T^{N+1}_e with level less then $(N+1)$ is a local copy of the corresponding vertex from G. More precisely: let $v' \in V(T^{N+1}_e)$ and the level of v' is less or equal then N. Denote $v = \Gamma^{N+1}_e(v')$. Then for any vertex $w \in V(G)$ such that either $(v, w) \in E(G)$ or $(w, v) \in E(G)$,

there exist exactly one vertex $w' \in V(T_e^{N+1})$ such that $\Gamma_e^{N+1}(w') = w$ and v' is connected with w' in the same way (direction) as v and w are connected in G. Then it is clear that we can extend the mapping Γ on edges by saying $\Gamma_e^{N+1}(e' = (v', w')) = (\Gamma_e^{N+1}(v'), \Gamma_e^{N+1}(w'))$. Now for every vertex $v' \in V(T_e^{N+1})$ and an incident edge $\tilde{e}$, we can define the coefficient $a_{v'}^{\tilde{e}} = a_v^e$, where $v = \Gamma_e^{N+1}(v')$, and $e = \Gamma_e^{N+1}(\tilde{e})$. We also set the cost and capacity on the computation tree correspondingly to the initial graph, so $c_{\tilde{e}} = c_{\Gamma_e^{N+1}(\tilde{e})}$ and $u_{\tilde{e}} = u_{\Gamma_e^{N+1}(\tilde{e})}$

Now assume there is a (GMNF) problem stated for a graph G. We define the *induced* (GMNF)$_e^N$ problem on a computation tree T_e^N in the following way. Let $V^0(T_e^N) \subset V(T_e^N)$ be a set of vertices with levels less than N. Then consider the problem:

$$\begin{aligned} \texttt{minimize} \quad & \sum_{\tilde{e} \in E(T_e^N)} c_{\tilde{e}} x_{\tilde{e}} \\ \texttt{subject to} \quad & \sum_{\tilde{e} \in E_{v'}} a_{v'}^{\tilde{e}} x_{\tilde{e}} = f_{v'}, \quad \forall v' \in V^0(T_e^N), \\ & 0 \le x_{\tilde{e}} \le u_{\tilde{e}}, \quad \forall \tilde{e} \in E(T_e^N). \end{aligned} \tag{GMNF$_e^N$}$$

Roughly speaking, (GMNF$_e^N$) is just a simple (GMNF) on a computation tree, except that there are no balance constraints for the vertices of N^{th} level. Keeping in mind that the computation tree is *locally equivalent* to the initial graph, and that Min-Sum algorithm belongs to message-passing heuristic, which means that the algorithm works *locally* at each step, one can intuitively guess that BP for (GMNF$_e^N$) works quite similar as BP for the initial (GMNF). This reasoning can be formalized in the following lemma from [11].

Lemma 3.1: Let $\hat{x}_e^N$ be the value produced by BP for (GMNF) at the end of iteration N for the flow value on edge $e \in E$. Then there exists an optimal solution y^* of (GMNF$_e^N$) such that $y_{e'}^* = \hat{x}_e^N$, where e' is the root of T_e^N.

Though this lemma was proven only for ordinary Min-Cost Network Flow problem, where $|a_v^e| = 1$ for all v, e, its proof doesn't rely on these coefficients at any point, which allows us to extend it for any values of these coefficients.

C. Main results

We will now use lemma 3.1 to prove our main result of correctness of BP Min-Sum for (GMNF). The following theorem is the generalization of Theorem 4.1 from [11], and our proof shares the ideas from the original proof.

Let $n = |V(G)|$, and denote by $\hat{x}^N$ the estimation of flow after N iterations of Algorithm 1.

Theorem 3.2: Suppose (GMNF) has a unique solution x^*. Define L to be the maximum absolute value of the cost of a simple directed path in $G(x^*)$, and T as the minimum of the reducers among all such paths. Then for any $N \ge \left(\frac{L}{2\sigma(x^*)T} + 1\right) n$, $\hat{x}^N = x^*$.

Proof: Suppose to the contrary that there exists $e_0 = (v_\alpha, v_\beta) \in E$ and $N \ge \left(\frac{L}{2\sigma(x^*)T} + 1\right) n$ such that $\hat{x}_{e_0}^N \ne x_{e_0}^*$. By Lemma 3.1, there exist an optimal solution y^* of GMNF$_{e_0}^N$ such that $y_{e_0}^* = \hat{x}_{e_0}^N$, and thus $y_{e_0}^* = x_{e_0}^*$. Then, without loss of generality, assume $y_{e_0}^* > x_{e_0}^*$. We will show that it it possible to adjust y^* in such way, that the flow in GMNF$_{e_0}^N$ will decrease, which will contradict to the optimality of y^*.

Let $e_0' = (v_\alpha', v_\beta')$ be the root edge of the computation tree $T_{e_0}^N$. Since y^* is a feasible solution of GMNF$_{e_0}^N$ and x^* is a feasible solution of GMNF:

$$f_{\Gamma(v_\alpha')} = \sum_{\tilde{e} \in E_{v_\alpha'}} a_{v_\alpha'}^{\tilde{e}} y_{\tilde{e}}^* = a_{v_\alpha'}^{e_0'} y_{e_0'}^* + \sum_{\tilde{e} \in E_{v_\alpha' \setminus e_0}} a_{v_\alpha'}^{\tilde{e}} y_{\tilde{e}}^*$$

$$f_{\Gamma(v_\alpha')} = \sum_{\tilde{e} \in E_{\Gamma(v_\alpha')}} a_{\Gamma(v_\alpha')}^{\tilde{e}} x_{\tilde{e}}^* = a_{\Gamma(v_\alpha')}^{e_0'} x_{e_0}^* + \sum_{\tilde{e} \in E_{v_\alpha' \setminus e_0}} a_{\Gamma(v_\alpha')}^{\tilde{e}} x_{\tilde{e}}^*$$

Since the nodes and the edges in the computation tree $T_{e_0}^N$ are copies of nodes and vertexes in G, $a_{\Gamma(v_\alpha')}^{\tilde{e}} = a_{v_\alpha'}^{\tilde{e}}$. Then from above equalities it follows that there exists $e_1' \ne e_0'$ incident to v_α' in $T_{e_0}^N$ such that $a_{v_\alpha'}^{e_1'}(x_{\Gamma(e_1')}^* - y_{e_1'}^*) > 0$. If $a_{v_\alpha'}^{e_1'} > 0$, then e_1' is an in-arc for v_α', and we say that e_1' has the same orientation, as e_0'. In such case, $x_{\Gamma(e_1')}^* > y_{e_1'}^*$. Otherwise, we say that e_1' has the opposite orientation, and $x_{\Gamma(e_1')}^* < y_{e_1'}^*$. Using the similar arguments, we will find $e_{-1}' \ne e_0'$ incident to v_β' satisfying similar condition. Then we can apply the similar reasoning for the other ends of e_1', e_{-1}', using the balance constraints and inequalities on components of x^* and y^* for corresponding vertexes. In the end, we will have a non-directed path starting and ending in leaves of $T_{e_0}^N$: $X = \{e_{-N}', e_{-N+1}', \dots, e_{-1}', e_0', e_1', \dots, e_N'\}$ such that for $-N \le i \le N$ one of two cases holds:

- $y_{e'}^* > x_{\Gamma(e')}^*$. Then $e_i' = (v', w')$ has the same orientation as e_0. In this case, define $Aug(e') = (v', w')$ and $e' \in D$.
- $y_{e_i'}^* < x_{\Gamma(e_i')}^*$. Then $e_i' = (v', w')$ and e_0 have opposite orientations. Define $Aug(e') = (w', v')$ and $e' \in O$.

Note that the capacity constraints are similar for corresponding vertices from GMNF$_{e_0}^N$ and GMNF, and since y^* is feasible for GMNF$_{e_0}^N$, hence, every $e' \in X$ it holds $0 \le y_{e'}^* \le u_{e'} = u_{\Gamma e'}$. Then, for any $e' \in D$, we have $x_{\Gamma(e')}^* < y_{e'}^* \le u_{e'}$, which means that $\Gamma(e') \in G(x^*)$ (from the definition of the residual network). Note that in this case $e' = Aug(e')$, so $\Gamma(Aug(e')) \in E(G(x^*))$. Next, let's now $e' = (v', w') \in O$, and thus $x_{\Gamma(e')}^* > y_{e'}^* \ge 0$. Again, out of the definition of the residual network, $(\Gamma(w'), \Gamma(v')) \in E(G(x^*))$. But since $e' \in O$, we have $Aug(e') = (w', v')$, thus $\Gamma(Aug(e')) \in E(G(x^*))$. Therefore, for every edge $e' \in X$ it holds $\Gamma(Aug(x^*)) \in E(G(x^*))$.

From the definition of $Aug(e')$ for $e' \in X$ one can see that all $Aug(e')$ have the same direction as e_0. Therefore, $W = \{Aug(e_{-N}'), Aug(e_{1-N}'), \dots, Aug(e_{-1}'), Aug(e_0'), Aug(e_1'), \dots, Aug(e_{N-1}'), Aug(e_N')\}$ is the directed path in $T_{e_0}^N$, and we will call it *augmenting path* of y^* with respect to x^*. $\Gamma(W)$ is also a directed walk on $G(x^*)$, which can be decomposed into a simple directed path P and a collection of k simple directed cycles $C_1, C_2, \dots, C_k$. Since

each simple cycle or path has at most n edges and W has $2N+1$ edges, it holds $l(W) \leq n + kn$. On the other hand, $l(W) = 2N + 1 = \dfrac{Ln}{\sigma(x^*)T} + n + 1$. Then we obtain $k > \dfrac{L}{\sigma(x^*)T}$. Further we would denote by $c^*(\cdot)$ and $l^*(\cdot)$ costs of cycles and paths in the residual network $G(x^*)$. For each cycle C_i we have $c^*(C_i) \geq \sigma(x^*) > 0$ by Lemma (2.1), while the cost of P is at least $-L$. Then using Lemma (2.2) we have:

$$l^*(W) \geq l^*(P) + kT\sigma(x^*) > -L + \frac{L}{\sigma(x^*)T}T\sigma(x^*) = 0$$

We will now "extract" flow from W in $T^N_{e_0}$. Let's redefine the numeration of arcs in W for convenience: $W = \{w_1, w_2, \ldots, w_{2N+1}\}$. This edges correspond to $X = \{w'_1, w'_2, \ldots, w'_{2N+1}\}$, where w_i and w'_i have the same orientation if $w'_i \in D$, and have the opposite orientation if $w'_i \in O$. Hence extraction of flow from w_i means decreasing $y^*_{w_i}$ if $w'_i \in D$, and increasing $y^*_{w_i}$ if $w'_i \in O$.
Then similarly to the proof of Lemma 2.1, to keep the balance in all the vertexes of W (except the start and the end, since there are no balance constraints for them in GMNF$^N_{e_0}$), we have to adjust the flow in the following way:

$$\tilde{y}_{w'_1} = \begin{cases} y^*_{w'_1} - \lambda, & \text{if } w'_i \in D \\ y^*_{w'_1} + \lambda, & \text{if } w'_i \in O \end{cases} \tag{2}$$

$$\tilde{y}_{w'_i} = \begin{cases} y^*_{w'_i} - \lambda \prod_{j=2}^{i} \delta(v_j, w'_{i-1}, w'_i), & \text{if } w'_i \in D \\ y^*_{w'_i} + \lambda \prod_{j=2}^{i} \delta(v_j, w'_{i-1}, w'_i), & \text{if } w'_i \in O \end{cases} \tag{3}$$

for $i = 2, 3, \ldots, 2N+1$.

Obviously, there exist small enough $\lambda > 0$ such that $\tilde{y}_e > x^*_{\Gamma(e)} \geq 0 \ \forall e \in D$, and $\tilde{y}_e < x^*_{\Gamma(e)} \leq u_{\Gamma(e)} \ \forall e \in O$, so the capacity constraints are satisfied for $\tilde{y}$. (The balance constraints are satisfied by the construction). So, $\tilde{y}$ is a feasible solution of GMNF$^N_{e_0}$. Now we explore how the total cost changes after such transformation of the flow:

$$\sum_{e' \in E(T^N_{e_0})} c_{\Gamma(e')} y^*_{e'} - \sum_{e' \in E(T^N_{e_0})} c_{\Gamma(e')} \tilde{y}_{e'} =$$

$$= \sum_{e' \in E(T^N_{e_0})} c_{\Gamma(e')} (y^*_{e'} - \tilde{y}_{e'}) =$$

$$= \sum_{e' \in D} c_{\Gamma(e')} \lambda_{e'} - \sum_{e' \in O} c_{\Gamma(e')} \lambda_{e'} =$$

$$= \sum_{e' \in D} c^*_{\Gamma(e')} \lambda_{e'} + \sum_{e' \in O} c^*_{\Gamma(e')} \lambda_{e'} = \sum_{e' \in W} c^*_{\Gamma(e')} \lambda_{e'} =$$

$$= \sum_{i=1}^{2N+1} \left(c^*_{\Gamma(w_i)} \lambda \prod_{j=2}^{i} \delta(v_j, w_{j-1}, w_j) \right) = l(W)\lambda > 0$$

Here we used that $c^*_{\Gamma(e')} = c_{\Gamma(e')}$ for $e' \in D$ and $c^*_{\Gamma(e')} = -c_{\Gamma(e')}$ for $e' \in O$, that is obvious from the construction of W, D, and O.

So we found the feasible solution of GMNF$^N_{e_0}$ with the total cost less then of y^*. It means that y^* is not an optimal solution of GMNF$^N_{e_0}$, which leads us to the contradiction. ■

IV. CONCLUSIONS

The proved correctness of the Belief Propagation algorithms for General Min-Cost Network Flow problems may serve as justification of applying these algorithms in practice for real problems. The statement of GMNF problem is broad enough, and thus many practical problems may fall under this formulation, which means that these problems may be solved correctly using BP algorithms.

The future research is required to determine the speed of convergence of BP for this Generalized Min-Cost Network Flow, since in our paper we have only proved that after the finite iterations of the algorithms the answer will not change and be the correct one. However, since $\sigma(x^*)$ in 3.2 may be arbitrary small for some problems, the number of iterations until the algorithm will give the correct answer may be arbitrary big, and further analysis is needed to reasonably bound the number of steps that should be done.

REFERENCES

[1] M. J. Wainwright, M. I. Jordan *et al.*, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1–2, pp. 1–305, 2008.

[2] K. P. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 467–475.

[3] J. Pearl, *Reverend Bayes on inference engines: A distributed hierarchical approach*. Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

[4] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," *Exploring artificial intelligence in the new millennium*, vol. 8, pp. 236–239, 2003.

[5] V. Kolmogorov and M. J. Wainwright, "On the optimality of tree-reweighted max-product message-passing," in *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2005, pp. 316–323.

[6] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE transactions on Information Theory*, vol. 46, no. 2, pp. 325–343, 2000.

[7] G. B. Horn, "Iterative decoding and pseudo-codewords," Ph.D. dissertation, California Institute of Technology, 1999.

[8] M. Mézard, G. Parisi, and R. Zecchina, "Analytic and algorithmic solution of random satisfiability problems," *Science*, vol. 297, no. 5582, pp. 812–815, 2002.

[9] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 599–618, 2001.

[10] Y. Weiss and W. T. Freeman, "On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 736–744, 2001.

[11] D. Gamarnik, D. Shah, and Y. Wei, "Belief propagation for min-cost network flow: Convergence and correctness," *Operations Research*, vol. 60, no. 2, pp. 410–428, 2012.

[12] E. Birmelé, R. Ferreira, R. Grossi, A. Marino, N. Pisanti, R. Rizzi, and G. Sacomoto, "Optimal listing of cycles and st-paths in undirected graphs," in *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2013, pp. 1884–1896.

[13] M. Bayati, C. Borgs, J. Chayes, and R. Zecchina, "On the exactness of the cavity method for weighted b-matchings on arbitrary graphs and its relation to linear programs," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 06, p. L06001, 2008.

[14] M. Bayati, D. Shah, and M. Sharma, "Max-product for maximum weight matching: Convergence, correctness, and lp duality," *IEEE Transactions on Information Theory*, vol. 54, no. 3, pp. 1241–1251, 2008.

[15] S. Sanghavi, D. Malioutov, and A. S. Willsky, "Linear programming analysis of loopy belief propagation for weighted matching," in *Advances in neural information processing systems*, 2008, pp. 1273–1280.