A Colored Gauss-Seidel Approach for the Distributed Network Flow Problem

Alessio Maffei, Luigi Iannelli, Luigi Glielmo

***Abstract*— A distributed solution for the minimum cost network flow problem is here proposed. The method uses a block version of the Gauss-Seidel algorithm in order to iteratively solve the dual problem whose structure is such that the Hessian matrix coincides with the Laplacian matrix of the corresponding graph. Thus, the updating rule turns out to be fully distributed. In order to increase the parallelism degree, a graph coloring scheme allows the clustering of nodes that reduces the sequentiality of the Gauss-Seidel technique. Numerical experiments show the effectiveness of the approach compared with methods recently presented in the literature.**

I. Introduction

The minimum cost network flow problem consists of finding the most economic way to transport objects through a given distribution network, from a set of suppliers to a set of consumers [1]–[3]. Problems of minimum cost flow can be easily found in many practical engineering and management systems, e.g., minimizing the road traveled in moving goods through a network [2], [4]; optimizing the cost associated with using particular routes in data communication networks [2], [5], [6]; addressing communication and computation challenges in cloud computing [7]; optimizing resource allocation in wireless data network [8]; distributing power flows in a smart grid power network minimizing the overall cost [9].

One of the most discussed topic in this research field consists of solving the problem in a distributed way, thus allowing to manage large-scale networks in a scalable way. A distributed iterative method for minimizing the network flow cost is proposed in [1], [2], where the authors use dual subgradient descent methods to achieve the optimal network flow. Subgradient methods are also proposed in [10], [11] where averaging algorithms for locally sharing information over the network are used.

Performances improvements with respect to these methods can be obtained by using second order Newton methods [2], [12], that lead to a distributed implementation as long as particularly structured matrices approximating the inverse of the Hessian are used. Another possibility is to implement a superlinear convergent Newton-like method via the conjugate gradient by graph operations, without explicit computation or storage of any Hessian matrix [13]. Distributed solutions can be also achieved by using the ϵ-relaxation method [1], as well. For instance [14] proposes a parallel asynchronous version of the ϵ-relaxation method.

Recently, the research has been directed toward the use of matrix splitting techniques for Newton direction approximations, like the accelerated dual descent approach (ADD) that allows a distributed optimization with fast converge rate [15], [16]. Authors use the Neumann series expansion of the pseudo-inverse of the Hessian for computing the Newton direction. The algorithm is called ADD-N since each node uses information from N hops away.

In this paper we propose an alternative distributed algorithm for solving the classic minimum cost network flow problem. Starting from the dual problem we compute the Newton direction by using a parallel Gauss-Seidel method typically used for solving linear systems [2]. The algorithm that comes out is cyclic, partially ordered and asynchronous, i.e., computation results are available to other nodes as soon as they are produced, matching the behavior of real-life large-scale systems [17], [18]. The proposed approach is inspired by [19] where the authors carried out a distributed solution for maximizing the network source utility by using the Hessian diagonal in a scaled Newton method. Differently from that method that was synchronous (i.e., the updated information was available to other nodes only after the complete iteration), in this paper an asynchronous implementation is looked at, in order to exploit advantages inherent to such kind of approach [17]. Indeed, much of the data dependencies in the Gauss-Seidel algorithms could be eliminated by performing nodes ordering to maximize the degree of parallelism. To this aim we used the *graph coloring* technique as a preconditioning phase [20] that allows to cluster nodes in subsets (colors) so that all nodes in the same subset are not connected among them. The method carried out is called block (or colored) Gauss-Seidel: at each iteration only a single cluster of nodes performs in parallel the updating rule. Such an approach has been successfully used in the literature dealing with linear equations [20] and partial differential equations [21], [22]. Here we will show how it can be effectively used also for solving minimum cost network flow problems for large-scale networks, in particular when the available hardware parallelism is not comparable with the dimension of the network.

II. Preliminaries of graph theory

A weighted directed graph is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, where $\mathcal{V}$ is a finite non-empty set of $n = |\mathcal{V}|$ elements called vertices (or nodes), $\mathcal{E}$ is a set of ordered pairs of distinct vertices called edges (or links), with $E = |\mathcal{E}|$, and $w \in \mathbb{R}^E$

Dipartimento di Ingegneria, Università del Sannio, Piazza Roma 21, 82100 Benevento, Italy, email: {amaffei, luigi.iannelli, luigi.glielmo}@unisannio.it. The research leading to these results has received funding from the EU FP7/2007-2013 under grant agreement n. 318184 (I3RES project).

is the vector of edges weights. Each edge is denoted by $e = (i,j) \in \mathcal{E}$ where we refer to $i \in \mathcal{V}$ and $j \in \mathcal{V}$ as *tail* and *head* of the edge e, respectively. Each node $i \in \mathcal{V}$ has a set of neighbours denoted as $\mathcal{N}_i$. In the sequel, without loss of generality, all vertices and edges will be identified with the integers $\{1, 2, \dots, n\}$ and $\{1, 2, \dots, E\}$, respectively. We can now introduce the *vertex-edge incidence matrix* $\mathbf{A} \in \mathbb{R}^{n \times E}$ of a directed graph $\mathcal{G}$, which is the matrix whose entries are

$$A_{ie} = \begin{cases} 1, & \text{if node } i \text{ is the } \textit{tail} \text{ of the } e\text{-th edge} \\ -1, & \text{if node } i \text{ is the } \textit{head} \text{ of the } e\text{-th edge} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Let $\mathbf{L} \in \mathbb{R}^{n \times n}$ be the weighted *Laplacian matrix* [23] of the graph $\mathcal{G}$, defined as $\mathbf{L} = \mathbf{A}\mathbf{W}\mathbf{A}^\top$, where $\mathbf{W} \in \mathbb{R}^{n \times n}$ is the diagonal matrix of the edge weights w, and the superscript $^\top$ denotes the transpose matrix. The Laplacian matrix $\mathbf{L}$ has the property of being symmetric positive semi-definite. Moreover it holds that $0 = \mu_1(\mathbf{L}) \leqslant \mu_2(\mathbf{L}) \leqslant \cdots \leqslant \mu_n(\mathbf{L})$ where $\mu_i(\mathbf{X})$ is the i-th eigenvalue of matrix $\mathbf{X}$.

III. Network flow problem

Let us consider a network of nodes $i \in \mathcal{V}$ connected among them through links $e \in \mathcal{E}$ with a given direction. On each e it is possible to choose a quantity x_e of *flow* so that x_e is positive if the flow goes along the same direction of e and it is negative if it goes in the opposite direction. In this way we are modeling the network as a directed graph without specifying the edges weights (see Fig. 1). The graph is assumed to be strictly connected.

We associate to each node i a real number b_i representing its supply/demand. If $b_i > 0$, node i is a *supply node*; if $b_i < 0$, node i is a *demand node* with a demand of $-b_i$; and if $b_i = 0$, node i is called *transshipment node* [1]–[3]. Therefore, the quantity b_i can be regarded as an exogenous/external flow. We want that all flows must satisfy the *conservation constraint*: $\sum_{e=(i,j)} x_e - \sum_{e=(j,i)} x_e = b_i \forall i$. It is required that the vector $b \in \mathbb{R}^n$ satisfies the constraint $\sum_i^n b_i = 0$ in order to make the problem feasible.

The aim of the *minimum cost network flow problem* is to determine the flow x_e of each link in order to minimize a given flow cost function. We assume the flow cost on each link is a scalar function $\phi_e(x_e)$ denoting the cost of x_e units of flow traversing the link e. Thus, the network flow problem

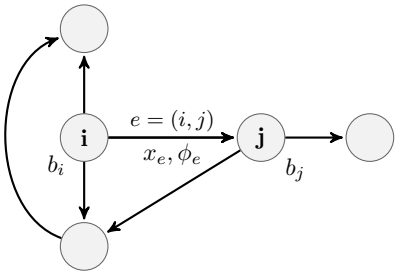

Fig. 1. Graph model for the network flow.

can be written as

$$\begin{aligned} \min_x \; & f(x) = \sum_{e=1}^{E} \phi_e(x_e) \\ \text{s.t.} \; & \mathbf{A}x = b \end{aligned} \quad (2)$$

where $x \in \mathbb{R}^E$ is the vector obtained by stacking in one column all x_e and the vertex-edge incidence matrix $\mathbf{A}$ has been used for expressing the conservation constraint. In line with part of the literature (e.g., see [2], [15]) we make the

Assumption 1: The cost functions ϕ_e are assumed to be strongly convex and twice continuously differentiable.

A. The dual problem

By following the same approach as in [15] we formulate the dual problem by introducing Lagrange multipliers $\lambda \in \mathbb{R}^n$. The Lagrangian function $\mathcal{L} : \mathbb{R}^E \times \mathbb{R}^n \to \mathbb{R}$ associated with the problem (2) is

$$\mathcal{L}(x, \lambda) = f(x) - \lambda^\top (\mathbf{A}x - b), \quad (3)$$

and the dual function [12] is defined by minimizing $\mathcal{L}(x, \lambda)$ over x, i.e.,

$$\tilde{q}(\lambda) \triangleq \min_x f(x) - \lambda^\top (\mathbf{A}x - b), \quad (4)$$

that is always a concave function. For notational convenience we introduce the function $q(\lambda) \triangleq -\tilde{q}(\lambda)$ and thus the optimal solution of the dual problem is

$$\lambda^* = \arg\max_\lambda \tilde{q}(\lambda) = \arg\min_\lambda q(\lambda). \quad (5)$$

Since $f(x)$ is strictly convex (as implied by Assumption 1) and the optimization problem is feasible, the duality gap is zero [12]. Thus, we can solve the dual problem (5) and then compute the associated primal optimal variables as the minimizer of $\mathcal{L}(x, \lambda^*)$; in particular, using the separability form of $f(x)$, we can compute the optimal flow associated with the edge $e = (i,j)$ from only the dual variables of the corresponding nodes:

$$x_e^*(\lambda^*) = [\phi_e']^{-1}(\lambda_i^* - \lambda_j^*) \;\; \forall e \quad (6)$$

where $[\phi_e']^{-1}(\cdot)$ denotes the inverse function of the derivative $\frac{\mathrm{d}\phi_e(x_e)}{\mathrm{d}x_e}$, i.e., $[\phi_e']^{-1}(y) = \{x_e \,|\, \frac{\mathrm{d}\phi_e(x_e)}{\mathrm{d}x_e} = y\}$.

We introduce now a preliminary result that will be useful for proving the convergence of the proposed solution.

Lemma 1: The vector function $x(\lambda)$ defined as the column stack of all (6), i.e.,

$$x(\lambda) \triangleq \mathrm{col}\left([\phi_e']^{-1}\left(\mathbf{A}_{e\bullet}^\top \lambda\right)\right) \quad (7)$$

is globally Lipschitz continuous.

Proof: The Assumption 1 leads $\phi_e''(\cdot)$ to be lower bounded by some positive value ξ^{-1}. Therefore, the derivative of $[\phi_e']^{-1}$ is upper bounded by ξ and, thus,

$$\left|[\phi_e']^{-1}(y_1) - [\phi_e']^{-1}(y_2)\right| \leqslant \xi |y_1 - y_2|, \quad \forall y_1, y_2 \in \mathbb{R}. \quad (8)$$

It follows

$$\begin{aligned} \|x(\lambda) - x(\overline{\lambda})\| &= \|[\phi']^{-1}(\mathbf{A}^\top \lambda) - [\phi']^{-1}(\mathbf{A}^\top \overline{\lambda})\| && \text{(9a)} \\ &\leqslant \xi \|\mathbf{A}\| \cdot \|\lambda - \overline{\lambda}\|, \quad \forall \lambda, \overline{\lambda} \in \mathbb{R}^n && \text{(9b)} \end{aligned}$$

where $\|\cdot\|$ is the usual induced matrix 2-norm. ■

B. Newton's method

Newton's method is an iterative procedure based on a simple quadratic approximation for minimizing convex functions. Starting with an initial guess $\lambda^{(0)}$, at each step k the current duals $\lambda^{(k)}$ are updated by minimizing the second-order approximation of $q(\lambda)$. The ordinary Newton iteration is thus

$$\lambda^{(k+1)} = \lambda^{(k)} + \alpha d^{(k)}, \tag{10}$$

where α is a properly chosen step size and d is the descent Newton direction. Iteration (10) leads itself well to a synchronous model whereby each node i computes its state and communicates the updated value $\lambda_i^{(k+1)}$ to the others.

For a second order Newton method, d is solution of the linear system of equations

$$\mathbf{H}^{(k)} d^{(k)} = -g^{(k)}, \tag{11}$$

where $g^{(k)} = g(\lambda^{(k)})$ is the gradient of the negative of the dual function at the step k, i.e., $g(\lambda^{(k)}) = \nabla q(\lambda^{(k)})$, and $\mathbf{H}^{(k)} = \mathbf{H}(\lambda^{(k)})$ denotes its Hessian, $\nabla^2 q(\lambda^{(k)})$.

Considering (4), we can straightforwardly compute the gradient $g(\lambda)$ that is a function of the primal optimizers:

$$g^{(k)} = \mathbf{A} x(\lambda^{(k)}) - b. \tag{12}$$

From the definition of the vertex-edge incidence matrix $\mathbf{A}$, it follows that the computation of the gradient is naturally distributed: at each k, the node i is able to evaluate its gradient component $g_i^{(k)}$ by using local primal iterates $x_e(\lambda^{(k)})$ for all $e = (i, j)$ and $e = (j, i)$. Note that, from eq. (6), such information depends only on dual iterates $\lambda_i^{(k)}$ and $\lambda_j^{(k)}\ \forall j \in \mathcal{N}_i$.

The gradient $g(\lambda)$ satisfies nice regularities properties as proved in the following

Lemma 2: Under Assumption 1 the gradient $\nabla q(\lambda)$, is globally Lipschitz:

$$\|\nabla q(\lambda) - \nabla q(\overline{\lambda})\| \leqslant K \|\lambda - \overline{\lambda}\|, \quad \forall \lambda, \overline{\lambda} \in \mathbb{R}^n \tag{13}$$

where $K = \xi \mu_n(\mathbf{A}\mathbf{A}^\top)$.

Proof: Considering the induced matrix 2-norm properties, we have

$$\begin{aligned} \|\nabla q(\lambda) - \nabla q(\overline{\lambda})\| &\leqslant \|\mathbf{A}\| \, \|x(\lambda) - x(\overline{\lambda})\| && \text{(14a)} \\ &\leqslant \xi \|\mathbf{A}\| \cdot \|\mathbf{A}\| \cdot \|\lambda - \overline{\lambda}\| && \text{(14b)} \\ &= \xi \mu_n(\mathbf{A}\mathbf{A}^\top) \|\lambda - \overline{\lambda}\|, && \text{(14c)} \end{aligned}$$

where (14b) comes out from Lemma 1, while (14c) holds because $\|\mathbf{A}\| = \|\mathbf{A}^\top\| = \sqrt{\mu_n(\mathbf{A}\mathbf{A}^\top)}$, i.e., the square root of the max eigenvalue of the unweighted Laplacian of $\mathcal{G}$. ■

In order to compute the dual Hessian matrix $\mathbf{H}^{(k)}$, in [15] authors consider the second order approximation of the primal objective evaluated at the current step, this leads to

$$\mathbf{H}^{(k)} = \mathbf{A} \left[\nabla^2 f \left(x(\lambda^{(k)}) \right) \right]^{-1} \mathbf{A}^\top. \tag{15}$$

Due to the properties of strict convexity and separability of $f(x)$, it follows that $\left[\nabla^2 f \left(x(\lambda^{(k)})\right)\right]^{-1}$ exists and is a positive diagonal matrix. From the graph theory, the dual Hessian $\mathbf{H}^{(k)}$ can also be interpreted as the Laplacian matrix of a weighted connected graph with weights $[1/\phi_e''(x_e)]^{-1} \forall e$.

C. Colored Gauss-Seidel method

We now propose an algorithm which is based on Newton's iteration (10) where the Newton direction (11) is computed in a distributed approximate way. A possible choice is the Gauss-Seidel iterative method [1] which computes an approximated descent direction $\hat{d}^{(k)}$ whose components are

$$\hat{d}_i^{(k)} = - \left[\hat{\mathbf{H}}_{ii}^{(k)} \right]^{-1} \hat{g}_i^{(k)} \quad \forall i, \tag{16}$$

with

$$\hat{\mathbf{H}}^{(k)} = \mathbf{A} \left[\nabla^2 f \left(x(z^{(k)}) \right) \right]^{-1} \mathbf{A}^\top, \tag{17}$$

$$\hat{g}^{(k)} = \mathbf{A} x(z^{(k)}) - b. \tag{18}$$

The vector $z^{(k)}$ is such that the node i uses the new values of Lagrange multipliers as soon as they are computed:

$$z_i^{(k)} = \left(\lambda_1^{(k+1)}, \cdots, \lambda_{i-1}^{(k+1)}, \lambda_i^{(k)}, \cdots, \lambda_n^{(k)} \right). \tag{19}$$

In other words, once λ_i is determined by the node i, its value is used by all other nodes $j \in \mathcal{N}_i$ that still have to compute their own iterates. Therefore, the ordering of nodes affects the scheduling of the updates.

The resulting Gauss-Seidel algorithm for each node is thus characterized by the following iterations

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} - \alpha \left[\hat{\mathbf{H}}_{ii}^{(k)} \right]^{-1} \hat{g}_i^{(k)} \quad \forall i. \tag{20}$$

The structure of the matrix $\hat{\mathbf{H}}$ (the weighted Laplacian) makes the iteration fully distributed. Indeed, in order to compute the i-th element of $\lambda^{(k+1)}$, the node i requires values of the Lagrange multipliers only from its neighbors.

According to [17] the updating rule that comes out from the Gauss-Seidel approach could be defined as *cyclic ordered asynchronous* because the update of individual states follows a cyclic pattern defined by the nodes identifiers. Indeed, from (19)-(20) the node i computes the Lagrange multiplier $\lambda_i^{(k+1)}$ as soon as its *update condition* is satisfied, i.e., the information $\lambda_j^{(k+1)}$ is already available at all $j \in \mathcal{N}_i$ such that $j < i$. That means it is necessary to solve a synchronization problem by considering the node ordering.

Beyond the benefit of distributed computation, the Gauss-Seidel approach is motivated by the acceleration of convergence rate, because the information are used for the computation as soon as they become available (asynchronous update). However, this approach restricts the parallelism of the updates with respect to the synchronous case, since only the nodes that satisfy the update condition could simultaneously perform the update rule (20).

In order to maximize the parallelism, a nodes clustering could be applied yielding the so-called block Gauss-Seidel method. A natural way of clustering the network is suggested by the graph coloring. In graph theory, graph coloring allows to partition the vertices set of a generic (neither necessarily

Algorithm 1: Communication protocol for node i at iteration k

wait ($\forall j \in \mathcal{N}_i \,|\, \gamma(j) < \gamma(i)$ have completed $update\_function$);
$\lambda_j \leftarrow read\_lambda(j)$, $\forall j \in \mathcal{N}_i$;
$update\_function(\lambda)$;

weighted, nor directed) graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in C subsets, according to a set of "colors" $\mathcal{C} = \{1, 2, \ldots, C\}$ such that no subset contains a pair of neighboring vertices. In other words, a coloring of the graph $\mathcal{G}$ is a mapping $\gamma : \mathcal{V} \to \mathcal{C}$ such that $\gamma(i) \neq \gamma(j)$ if $(i, j) \in \mathcal{E}$, where $\gamma(i)$ is the color of the vertex i [24]. The graph coloring can be carried out just once during a preconditioning phase or, possibly, every time the network topology changes (e.g., a node is added or a link is broken).

The graph coloring problem, i.e., minimizing C, is a well-known NP-hard problem [25], however we are interested in coloring the graph through a distributed algorithm using a small numbers of colors, only, not necessarily the minimum.

Let us denote by $\beta_c = \{i \in \mathcal{V} \,|\, \gamma(i) = c\}$ the set of n_c nodes with color c. We use the notation $\lambda_{\beta_c} \in \mathbb{R}^{n_c}$ for referring to the vector whose components are taken from λ by selecting only the elements indexed by β_c. Now, instead of (19), we should consider the vector $z_c^{(k)}$ that, according to the colored Gauss-Seidel idea, is defined as

$$z_c^{(k)} = \left(\lambda_{\beta_1}^{(k+1)}, \cdots, \lambda_{\beta_{c-1}}^{(k+1)}, \lambda_{\beta_c}^{(k)}, \cdots, \lambda_{\beta_C}^{(k)}\right). \tag{21}$$

By some abuse of notation we will denote by $\hat{g}_{\beta_c}^{(k)}$ and $\hat{\mathbf{H}}_{\beta_c}^{(k)}$, respectively, the vector whose elements are selected from the β_c components of $g(z_c^{(k)})$, and the principal minor of $\mathbf{H}(z_c^{(k)})$ obtained by selecting β_c rows and columns.

Of course, such reordering is not needed, and only the color ordering establishes the update execution that can be analyzed through the introduced notation. Indeed, we can write in a matrix form the updating rule of the colored Gauss-Seidel algorithm for all nodes with color $c \in \mathcal{C}$ as

$$\lambda_{\beta_c}^{(k+1)} = \lambda_{\beta_c}^{(k)} - \alpha \left[\hat{\mathbf{H}}_{\beta_c}^{(k)}\right]^{-1} \hat{g}_{\beta_c}^{(k)}. \tag{22}$$

The benefit of this approach is that it reduces the number of sequential updates to the number of colors C, that is generally much less than n, e.g., for a planar graph C can be as low as 4 independently on n [26]. Indeed, the coloring allows to introduce a parallelism among all nodes with the same color: each node with the same color can simultaneously execute the updating function (22). The synchronization among nodes with different colors is guaranteed by the color mapping as before, see Algorithm 1.

The proposed method (22) maintains its distributed properties because the matrix $\hat{\mathbf{H}}_{\beta_c}^{(k)} \in \mathbb{R}^{n_c \times n_c}$ is diagonal and positive definite, since the nodes in β_c are not connected among them and $\mathcal{G}$ is assumed to be strictly connected. The description of the distributed $update\_function$ for the proposed method is given in the Algorithm 2.

Algorithm 2: $update\_function$ for node i

Data: $\phi(\cdot)$, λ, b_i, α;
$\hat{g}_i = 0$; $\hat{h}_i = 0$;
`/* Update primals */`
forall the $e = (i, j)$ $j \in \mathcal{N}_i$ **do**
  $x_e(\lambda) = [\phi_e']^{-1}(\lambda_j - \lambda_i)$;
  $\hat{g}_i = \hat{g}_i - x_e(\lambda)$;
  $\hat{h}_i = \hat{h}_i + [\phi_e''(x_e(\lambda))]^{-1}$;
end
forall the $e = (l, i)$, $l \in \mathcal{N}_i$ **do**
  $x_e(\lambda) = [\phi_e']^{-1}(\lambda_i - \lambda_l)$;
  $\hat{g}_i = \hat{g}_i + x_e(\lambda)$;
  $\hat{h}_i = \hat{h}_i + [\phi_e''(x_e(\lambda))]^{-1}$;
end
$\hat{g}_i = \hat{g}_i - b_i$;
`/* Update duals */`
$\lambda_i \leftarrow \lambda_i - \alpha \hat{h}_i^{-1} \hat{g}_i$;

D. Convergence analysis

A sufficient condition for the convergence of the colored Gauss-Seidel algorithm is here given.

***Theorem** 1:* Consider the colored Gauss-Seidel algorithm given by the iteration rule (22). If the Assumption 1 holds with $\xi^{-1} \leqslant \phi_e''(\cdot)$ $\forall e$, and the step size α is chosen as

$$0 < \alpha < 2 \frac{\mu_1(\hat{\mathbf{H}}_{\beta_c}^{(k)})}{\xi \mu_n(\mathbf{A}\mathbf{A}^\top)} \quad \forall c, k \tag{23}$$

then $\lim_{k\to\infty} \|\nabla q(\lambda^{(k)})\| = 0$ and $\lim_{k\to\infty} \lambda^{(k)} = \lambda^*$, where $q^* = q(\lambda^*)$ is the optimal value of the dual problem.

Proof: Let $s_c^{(k)} \in \mathbb{R}^n$ be a vector with all zero elements, but components in β_c that are equal to the vector $-[\hat{\mathbf{H}}_{\beta_c}^{(k)}]^{-1} \hat{g}_{\beta_c}^{(k)}$:

$$\left(s_c^{(k)}\right)^\top = \left(0, \ldots, 0, -(\hat{g}_{\beta_c}^{(k)})^\top [\hat{\mathbf{H}}_{\beta_c}^{(k)}]^{-1}, 0, \ldots, 0\right). \tag{24}$$

Note that

$$z_{c+1}^{(k)} = z_c^{(k)} + \alpha s_c^{(k)}, \quad 1 \leqslant c < C \tag{25a}$$
$$\lambda^{(k+1)} = z_C^{(k)} + \alpha s_C^{(k)}. \tag{25b}$$

Since $\nabla q(\cdot)$ is *Lipschitz continuous* (Lemma 2), from the *descent Lemma* [1, Ch. 3] we have

$$q(z_c^{(k)} + \alpha s_c^{(k)}) \leqslant q(z_c^{(k)}) + \alpha (s_c^{(k)})^\top \nabla q(z_c^{(k)}) + \frac{\alpha^2 \xi}{2} \mu_n(\mathbf{A}\mathbf{A}^\top) \|s_c^{(k)}\|^2. \tag{26}$$

Using the definition of $s_c^{(k)}$ and since $\hat{\mathbf{H}}_{\beta_c}^{(k)} = (\nabla^2 q(z_c^{(k)}))_{\beta_c}$ is diagonal positive definite, then

$$(s_c^{(k)})^\top \nabla q(z_c^{(k)}) = -(s_c^{(k)})^\top \hat{\mathbf{H}}_{\beta_c}^{(k)} s_c^{(k)} \tag{27a}$$
$$\leqslant -\mu_1(\hat{\mathbf{H}}_{\beta_c}^{(k)}) \|s_c^{(k)}\|^2, \tag{27b}$$

where $\mu_1(\hat{\mathbf{H}}_{\beta_c}^{(k)})$ is the minimum eigenvalue of the matrix $\hat{\mathbf{H}}_{\beta_c}^{(k)}$. By replacing (27a) in (26) and denoting by σ_c the

quantity $\alpha\left(\mu_1(\hat{\mathbf{H}}_{\beta_c}^{(k)}) - \frac{\alpha\xi}{2}\mu_n(\mathbf{A}\mathbf{A}^\top)\right)$, we obtain

$$q(z_c^{(k)} + \alpha s_c^{(k)}) \leqslant q(z_c^{(k)}) - \sigma_c \|s_c^{(k)}\|^2. \quad (28)$$

The condition (23) on α guarantees the quantity σ_c being positive. The inequality (28) is true for all $1 \leqslant c \leqslant C$, and then, by iterating recursively on relations (25) and (28), we can write

$$q(\lambda^{(k+1)}) = q(z_C^{(k)} + \alpha s_C^{(k)}) \leqslant q(\lambda^{(k)}) - \sum_{c=1}^{C} \sigma_c \|s_c^{(k)}\|^2. \quad (29)$$

Furthermore, since q is bounded below by q^*, we get:

$$q^* \leqslant q(\lambda^{(k+1)}) \leqslant q(\lambda^{(0)}) - \sum_{\tau=0}^{k}\sum_{c=1}^{C} \sigma_c \|s_c^{(\tau)}\|^2. \quad (30)$$

By considering (30) for all k, we can conclude:

$$\sum_{\tau=0}^{\infty}\sum_{c=1}^{C} \sigma_c \|s_c^{(\tau)}\|^2 \leqslant q(\lambda^{(0)}) - q^* < \infty. \quad (31)$$

Therefore $\lim_{k\to\infty} \|s_c^{(k)}\| = 0 \ \forall c$. By looking at the expression (24) and considering that $\hat{\mathbf{H}}_{\beta_c}$ is invertible, it follows that $\lim_{k\to\infty} \|\hat{g}_{\beta_c}^{(k)}\| = \lim_{k\to\infty} \|(\nabla q(z_c^{(k)}))_{\beta_c}\| = 0$ and, due to (25) $\lim_{k\to\infty} \|z_c^{(k)} - \lambda^{(k)}\| = 0 \ \forall c$. It follows

$$\lim_{k\to\infty} \|(\nabla q(\lambda^{(k)}))_{\beta_c}\| = 0 \quad \forall c. \quad (32)$$

The optimality of $\lim_{k\to\infty} \lambda^{(k)} = \lambda^*$ comes from the convexity of $q(\cdot)$ (concavity of the dual function $\tilde{q}(\cdot)$). ■

IV. Numerical results

The proposed approach has been compared with one of the fastest distributed algorithm used to solve the network flow problem currently available in the literature, i.e., the ADD-N [15]. Such algorithm has an updating rule for λ that is synchronous and requires $N+1$ sub-iterations during which all nodes need to communicate with their N-hops neighbors (for example by using a consensus protocol).

As a case study, it has been considered a small-world network with $n = 300$ and $E = 6000$, generated according to the Watts-Strogatz model [27]. Anyway, similar results have been obtained also for scale-free and random networks. The coloring phase has been implemented in MATLAB by using the ordering heuristic *Smallest Degree Last* for parallel "greedy" graph-coloring algorithms [24], [28]. This allows to cluster the set of nodes into $C = 16$ subsets in about 6.23 s. The execution time required for the preconditioning phase depends, of course, on the complexity of the network.

As regards the flow costs, we used the objective functions $\phi_e(x_e) = \exp(-x_e) + \exp(x_e) \quad \forall e \in \mathcal{E}$, that satisfy Assumption 1, while the vector b has been randomly chosen satisfying the feasibility condition.

Simulations have been performed in C++ by using the *GraphChi* software [29]. It allows to analyze, process and mine huge real-world graphs on just a laptop, by using an asynchronous computation paradigm such that if the nodes are not neighbors their update function can be executed in parallel, otherwise a round robin scheduling is applied. Moreover the information are available to other nodes as soon as they have been updated, matching the asynchronous behavior of Gauss-Seidel method. Nevertheless, a synchronous updating paradigm can be 'simulated' by using temporary storage of updated variables. Therefore, GraphChi has been used for implementing and testing both colored Gauss-Seidel and ADD-N method, for different values of N.

Simulation results show how the optimal solution is achieved. In Fig. 2 the convergence behavior for the dual objective function is reported, for both Gauss-Seidel and ADD-N. Given a fixed step size $\alpha = 1$ (for both the algorithms), the colored Gauss-Seidel algorithm shows performances comparable to ADD-4. In Fig. 3 the error on the flow conservation constraint is depicted, while in Fig. 4 the rate of convergence for Gauss-Seidel and ADD-N is shown, as well. From such experiments the colored Gauss-Seidel algorithm shows a convergence rate comparable with that of ADD-N, in particular only for $N \geqslant 4$ the latter algorithm shows greater performances, requiring, anyway, more resources in terms of communication and elapsed time due to the sub-iterations phase.

The scalability of both algorithms, in terms of the execution time in dependence of the parallelism degree of the whole computation, has been then analyzed. Consider the case in which both the algorithms are run on a cluster of P processors. Let $\Delta t_\bullet$ be the time needed for executing the $update\_function$ for each method (in the case of ADD-N that includes the elapsed time for the $N+1$ sub-iterations). The total time $\Delta T_\bullet$ required for running a number $\mathcal{I}_\bullet$ of iterations are

$$\Delta T_{\mathrm{ADD-N}} = \left\lceil \frac{n}{P} \right\rceil (N+1) \cdot \mathcal{I}_{\mathrm{ADD-N}} \Delta t_{\mathrm{ADD-N}} \quad (33a)$$

$$\Delta T_{\mathrm{GS}} = \left\lceil \frac{n}{CP} \right\rceil C \cdot \mathcal{I}_{\mathrm{GS}} \Delta t_{\mathrm{GS}}, \quad (33b)$$

where $\lceil x \rceil = \min\{n \in \mathbb{Z} | n \geqslant x\}$.

Therefore, the speed-up S of colored Gauss-Seidel with respect to ADD-N is

$$S = \kappa \frac{\mathcal{I}_{\mathrm{ADD-N}}}{\mathcal{I}_{\mathrm{GS}}} \frac{\left\lceil \frac{n}{P} \right\rceil (N+1)}{\left\lceil \frac{n}{CP} \right\rceil C}, \quad (34)$$

where κ is the $\Delta t_\bullet$ ratio. As shown by the experiments discussed above, the convergence rate of both the algorithms is comparable and, thus, the number of iterations required for achieving a given accuracy on the solution is such that $\mathcal{I}_{\mathrm{ADD-N}} \approx \mathcal{I}_{\mathrm{GS}}$. If also $\kappa \approx 1$ (both the algorithms require approximately the same time for executing their update rule), it comes out that it is worth to use the colored Gauss-Seidel algorithm instead of ADD-N when $P \ll n/C$, thus obtaining $S \approx N+1$. When $P > n/C$, the execution time is still favorable for colored Gauss-Seidel if $C < (N+1)\, n/P$ that implies $S > 1$. Further, we analyzed the execution time obtained by stopping both the algorithms at the iteration 100, and such times have been normalized with respect to ΔT_{GS}, thus obtaining the speed-up values reported in Tab. I, both for the sequential processing ($P = 1$), both for the case in which the computation uses $P = 4$ processors (the machine

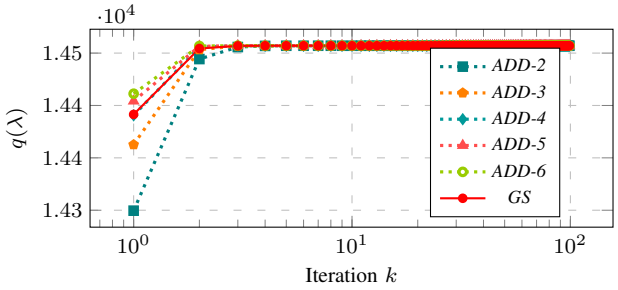

Fig. 2. Dual function comparison between ADD-N and Gauss-Seidel

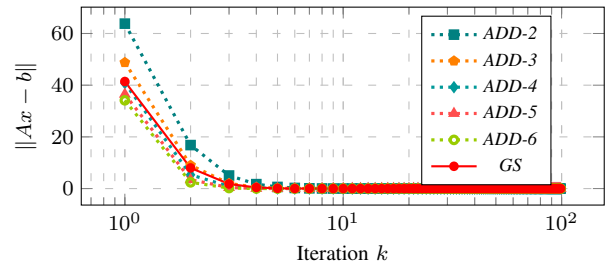

Fig. 3. Primal feasibility comparison between ADD-N and Gauss-Seidel

used a four cores processor). All measured execution times are of the order of seconds (maximum 14 s for ADD-6).

V. Conclusion

In this paper it has been proposed a distributed approach for solving the network flow problem with the aim of achieving a good scalability for large-scale networks. A colored Gauss-Seidel algorithm has been investigated as a distributed optimization strategy and numerical experiments have been shown to illustrate the effectiveness of the solution, also compared to existing approaches in the literature.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.

[2] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Belmont, MA, USA: Athena Scientific, 1998.

[3] R. T. Rockafellar, *Network Flows and Monotropic Programming*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 1984.

[4] J. B. Orlin, "Minimum convex cost dynamic network flows," *Mathematics of Operations Research*, vol. 9, no. 2, pp. 190–207, 1984.

[5] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1992.

[6] Y. Wu, A. Ribeiro, and G. Giannakis, "Robust routing in wireless multi-hop networks," in *41st IEEE Conference on Information Sciences and Systems*, 2007, pp. 637–642.

[7] S. Feizi, A. Zhang, and M. Medard, "A network flow approach in cloud computing," in *47th IEEE Conference on Information Sciences and Systems*, 2013, pp. 1–6.

P	GS	ADD-2	ADD-3	ADD-4	ADD-5	ADD-6
1	1	4.27	5.69	7.04	8.41	9.89
4	1	2.89	3.86	4.85	5.62	6.78

TABLE I

Speed-up in the GraphChi environment

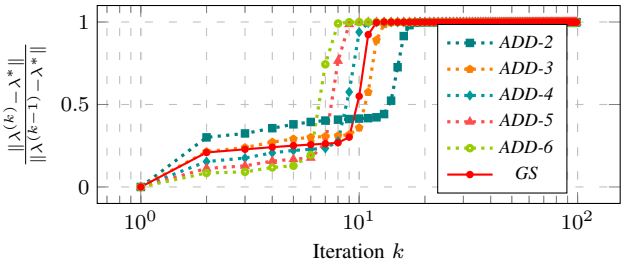

Fig. 4. Convergence rate comparison between ADD-N and Gauss-Seidel

[8] L. Xiao, M. Johansson, and S. P. Boyd, "Simultaneous routing and resource allocation via dual decomposition," *IEEE Transactions on Communications*, vol. 52, no. 7, pp. 1136–1144, 2004.

[9] J. Zhu, *Optimization of Power System Operation*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2009.

[10] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[11] I. Lobel and A. Ozdaglar, "Distributed subgradient methods for convex optimization over random networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 6, pp. 1291–1306, 2011.

[12] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[13] D. Bertsekas and E. Gafni, "Projected newton methods and optimization of multicommodity flows," *IEEE Transactions on Automatic Control*, vol. 28, no. 12, pp. 1090–1096, 1983.

[14] P. Beraldi, F. Guerriero, and R. Musmanno, "Parallel algorithms for solving the convex minimum cost flow problem," *Computational Optimization and Applications*, vol. 18, no. 2, pp. 175–190, 2001.

[15] M. Zargham, A. Ribeiro, A. Ozdaglar, and A. Jadbabaie, "Accelerated dual descent for network flow optimization," *IEEE Transactions on Automatic Control*, vol. 59, no. 4, pp. 905–920, 2014.

[16] M. Zargham, A. Ribeiro, and A. Jadbabaie, "Accelerated dual descent for constrained convex network flow optimization," in *52nd IEEE Conference on Decision and Control*, 2013, pp. 1037–1042.

[17] D. Cornforth, D. G. Green, and D. Newth, "Ordered asynchronous processes in multi-agent systems," *Physica D: Nonlinear Phenomena*, vol. 204, no. 1-2, pp. 70–82, 2005.

[18] D. Bertsekas, "Distributed asynchronous computation of fixed points," *Mathematical Programming*, vol. 27, no. 1, pp. 107–120, 1983.

[19] S. Athuraliya and S. Low, "Optimization flow control with newton-like algorithm," *Telecommunication Systems*, vol. 15, no. 3-4, pp. 345–358, 2000.

[20] D. P. Koester, S. Ranka, and G. Fox, "A parallel gauss-seidel algorithm for sparse power system matrices," in *ACM/IEEE Conference on Supercomputing*, 1994, pp. 184–193.

[21] J. M. Ortega and R. G. Voigt, "Solution of partial differential equations on vector and parallel computers," *SIAM Review*, vol. 27, no. 2, pp. 149–240, 1985.

[22] G. C. Fox, *Solving Problems on Concurrent Processors*. Upper Saddle River, NJ, USA: Prentice-Hall, 1987.

[23] C. Godsil and G. Royle, *Algebraic Graph Theory*. New York, NY, USA: Springer, 2001.

[24] J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. L. Martin, "A comparison of parallel graph coloring algorithms," Northeast Parallel Architecture Center, Syracuse University, Tech. Rep. SCCS-666, 1995.

[25] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[26] G. Gonthier, "Formal proof – the four-color theorem," *Notices of the AMS*, vol. 55, no. 11, pp. 1382–1393, 2008.

[27] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 409–10, 1998.

[28] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring," in *26th ACM Symposium on Parallelism in Algorithms and Architectures*, 2014, pp. 166–177.

[29] A. Kyrola, G. Blelloch, and C. Guestrin, "Graphchi: Large-scale graph computation on just a pc," in *10th USENIX Conference on Operating Systems Design and Implementation*, 2012, pp. 31–46.