A Distributed Newton Method for Network Utility Maximization, I: Algorithm*

Ermin Wei†, Asuman Ozdaglar†, and Ali Jadbabaie‡

May 11, 2011

Abstract

Most existing works use dual decomposition and first-order methods to solve Network Utility Maximization (NUM) problems in a distributed manner, which suffer from slow rate of convergence properties. This paper develops an alternative distributed Newton-type fast converging algorithm for solving NUM problems. By using novel matrix splitting techniques, both primal and dual updates for the Newton step can be computed using iterative schemes in a decentralized manner. We propose a stepsize rule and provide a distributed procedure to compute it in finitely many iterations. The key feature of our direction and stepsize computation schemes is that both are implemented using the same distributed information exchange mechanism employed by first order methods. We describe the details of the inexact algorithm here and in part II of this paper [30], we show that under some assumptions, even when the Newton direction and the stepsize in our method are computed within some error (due to finite truncation of the iterative schemes), the resulting objective function value still converges superlinearly in terms of primal iterations to an explicitly characterized error neighborhood. Simulation results demonstrate significant convergence rate improvement of our algorithm relative to the existing first-order methods based on dual decomposition.

*This work was supported by National Science Foundation under Career grant DMI-0545910, the DARPA ITMANET program, ONR MURI N000140810747 and AFOSR Complex Networks Program.

†Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology

‡Department of Electrical and Systems Engineering and GRASP Laboratory, University of Pennsylvania

1 Introduction

Most of today's communication networks are large-scale and comprise of agents with heterogeneous preferences. Lack of access to centralized information in such networks necessitate design of distributed control algorithms that can operate based on locally available information. Some applications include routing and congestion control in the Internet, data collection and processing in sensor networks, and cross-layer design in wireless networks. This work focuses on the rate control problem in wireline networks, which can be formulated in the *Network Utility Maximization (NUM)* framework proposed in [17] (see also [19], [26], and [9]). NUM problems are characterized by a fixed network and a set of sources, which send information over the network along predetermined routes. Each source has a local utility function over the rate at which it sends information. The goal is to determine the source rates that maximize the sum of utilities subject to link capacity constraints. The standard approach for solving NUM problems relies on using dual decomposition and subgradient (or first-order) methods, which through a price feedback mechanism among the sources and the links yields algorithms that can operate on the basis of local information [16], [19].[1] One major shortcoming of this approach is the slow rate of convergence.

In this paper, we propose a novel Newton-type second-order method for solving the NUM problem in a distributed manner, which leads to significantly faster convergence. Our approach involves transforming the inequality constrained NUM problem to an equality-constrained one through introducing slack variables and logarithmic barrier functions, and using an equality-constrained Newton method for the reformulated problem. There are two challenges in implementing this method in a distributed manner. First challenge is the computation of the Newton direction. This computation involves a matrix inversion, which is costly and requires global information. We solve this problem by using an iterative scheme based on a novel matrix splitting technique. Since the objective function of the (equality-constrained) NUM problem is *separable*, i.e., it is the sum of functions over each of the variables, this splitting enables computation of the Newton direction using decentralized algorithms based on limited *scalar* information exchange between sources and links, in a form similar to the feedback mechanism used by the subgradient methods. This exchange involves destinations iteratively sending route prices (aggregated link prices or dual variables along a route) to the sources, and sources sending the route price scaled by the corresponding Hessian element to the links along its route.

The second challenge is related to the computation of a stepsize rule that can guarantee local superlinear convergence of the primal iterations. Instead of the iterative backtracking rules typically used with Newton methods, we propose a stepsize choice based on the rule proposed in [23], which is inversely proportional to the inexact Newton decrement (where the inexactness arises due to errors in the computation of the Newton direction) if this decrement is above a certain threshold and takes the form of a pure Newton step otherwise. Computation of the inexact Newton decrement involves aggregating local information from the sources and links in the network. We propose a novel distributed procedure for computing the inexact Newton decrement in finite number of steps using again the same information exchange mechanism employed by first order methods. Therefore, our algorithm has *comparable level of information exchange* with the first-order methods applied to the NUM problem.

Our work contributes to the growing literature on distributed optimization and control of multi-agent networked systems. There are two standard approaches for designing distributed algorithms for such problems. The first approach, as mentioned above, uses dual decomposition

[1] The price feedback mechanism involves destinations (end nodes of a route) sending route prices (aggregated over the links along the route) to sources, sources updating their rates based on these prices and finally links updating prices based on new rates sent over the network.

and subgradient methods, which for some problems including NUM problems lead to iterative distributed algorithms (see [17], [19]). More recent work by Athuraliya and Low in [1] used diagonally scaled subgradient methods for NUM problems to approximate Newton steps and speed up the algorithm while maintaining their distributed nature. Despite improvements in speed over the first-order methods, as we shall see in the simulation section, the performance of this modified algorithm does not achieve the rate gains obtained by second-order methods.

The second approach involves considering *consensus-based* schemes, in which agents exchange local estimates with their neighbors to enable decentralized information aggregation over an exogenous (fixed or time-varying) network topology. It has been shown that under some mild assumption on the connectivity of the graph and updating rules, the distance from the vector formed by current estimates to consensus diminishes linearly. Consensus schemes can be used to compute the average of local values or more generally as a building block for developing first order distributed optimization algorithms (see [6], [27], [6], [22], [28], [13], [24], [14], [25] and [21]). Consensus updates can also be used to implement various computations involved in second order methods (e.g., stepsize calculation). However, we do not include such updates in our algorithm, due to their potentially slow convergence rates.

Other than the papers cited above, our paper is also related to [3] and [18]. In [3], Bertsekas and Gafni studied a projected Newton method for optimization problems with twice differentiable objective functions and simplex constraints. They proposed finding the Newton direction (exactly or approximately) using a conjugate gradient method. This work showed that when applied to multi-commodity network flow problems, the conjugate gradient iterations can be obtained using simple graph operations, however did not investigate distributed implementations. Similarly, in [18], Klincewicz proposed a Newton method for network flow problems that computes the dual variables at each step using an iterative conjugate gradient algorithm. He showed that conjugate gradient iterations can be implemented using a "distributed" scheme that involves simple operations and information exchange along a spanning tree. Spanning tree based computations involve passing all information to a centralized node and may therefore be restrictive for NUM problems which are characterized by decentralized (potentially autonomous) sources.

This paper contains the details of the inexact distributed Newton method and part II of the paper [30] contains convergence analysis of the method. The rest of the paper is organized as follows: Section 2 defines the problem formulation and related transformations. Section 3 describes the exact constrained primal-dual Newton method for this problem. Section 4 presents a distributed iterative scheme for computing the dual Newton step and the overall distributed inexact Newton-type algorithm. Section 5 presents simulation results to demonstrate convergence speed improvement of our algorithm to the existing methods with linear convergence rates. Section 6 contains our concluding remarks.

Basic Notation and Notions:

A vector is viewed as a column vector, unless clearly stated otherwise. We write $\mathbb{R}_+$ to denote the set of nonnegative real numbers, i.e., $\mathbb{R}_+ = [0, \infty)$. We use subscripts to denote the components of a vector and superscripts to index a sequence, i.e., x_i is the i^{th} component of vector x and x^k is the kth element of a sequence. When $x_i \geq 0$ for all components i of a vector x, we write $x \geq 0$.

For a matrix A, we write A_{ij} to denote the matrix entry in the i^{th} row and j^{th} column, and $[A]_i$ to denote the i^{th} column of the matrix A, and $[A]^j$ to denote the j^{th} row of the matrix A. We write $I(n)$ to denote the identity matrix of dimension $n \times n$. We use x' and A' to denote the transpose of a vector x and a matrix A respectively. For a real-valued function $f : X \to \mathbb{R}$, where X is a subset of $\mathbb{R}^n$, the gradient vector and the Hessian matrix of f at x in X are denoted by $\nabla f(x)$ and $\nabla^2 f(x)$ respectively. We use the vector e to denote the vector of all ones.

A real-valued convex function $g : X \to \mathbb{R}$, where X is a subset of $\mathbb{R}$, is *self-concordant* if it is three times continuously differentiable and $|g'''(x)| \leq 2g''(x)^{\frac{3}{2}}$ for all x in its domain.[2] For real-valued functions in $\mathbb{R}^n$, a convex function $g : X \to \mathbb{R}$, where X is a subset of $\mathbb{R}^n$, is self-concordant if it is self-concordant along every direction in its domain, i.e., if the function $\tilde{g}(t) = g(x+tv)$ is self-concordant in t for all x and v. Operations that preserve self-concordance property include summing, scaling by a factor $\alpha \geq 1$, and composition with affine transformation (see [7] Chapter 9 for more details).

2 Network Utility Maximization Problem

We consider a network represented by a set $\mathcal{L} = \{1, ..., L\}$ of (directed) links of finite nonzero capacity given by $c = [c_l]_{l\in\mathcal{L}}$ with $c > 0$. The network is shared by a set $\mathcal{S} = \{1, ..., S\}$ of sources, each of which transmits information along a predetermined route. For each link l, let $S(l)$ denote the set of sources use it. For each source i, let $L(i)$ denote the set of links it uses. We also denote the nonnegative source rate vector by $s = [s_i]_{i\in\mathcal{S}}$. The capacity constraint at the links can be compactly expressed as

$$Rs \leq c,$$

where R is the *routing matrix*[3] of dimension $L \times S$, i.e.,

$$R_{ij} = \begin{cases} 1 & \text{if link } i \text{ is on the route of source } j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

We associate a utility function $U_i : \mathbb{R}_+ \to \mathbb{R}$ with each source i, i.e., $U_i(s_i)$ denotes the utility of source i as a function of the source rate s_i. We assume the utility functions are additive, such that the overall utility of the network is given by $\sum_{i=1}^{S} U_i(s_i)$. Thus the Network Utility Maximization(NUM) problem can be formulated as

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{S} U_i(s_i) \\ \text{subject to} \quad & Rs \leq c, \\ & s \geq 0. \end{aligned} \tag{2}$$

We adopt the following assumptionwhich will be used in part II of this paper for convergence analysis (see [30] for more details).

Assumption 1. The utility functions $U_i : \mathbb{R}_+ \to \mathbb{R}$ are strictly concave, monotonically nondecreasing on $(0, \infty)$. The functions $-U_i : \mathbb{R}_+ \to \mathbb{R}$ are self-concordant on $(0, \infty)$.

To facilitate the development of a distributed Newton-type method, we consider a related equality-constrained problem by introducing nonnegative slack variables $[y_l]_{l\in\mathcal{L}}$ for the capacity constraints, defined by

$$\sum_{j=1}^{S} R_{lj}s_j + y_l = c_l \quad \text{for } l = 1, 2 \dots L, \tag{3}$$

[2]Self-concordant functions are defined through the following more general definition: a real-valued three times continuously differentiable convex function $g : X \to \mathbb{R}$, where X is a subset of $\mathbb{R}$, is *self-concordant*, if there exists a constant $a > 0$, such that $|g'''(x)| \leq 2a^{-\frac{1}{2}}g''(x)^{\frac{3}{2}}$ for all x in its domain [23], [15]. Here we focus on the case $a = 1$ for notational simplification in the analysis.

[3]This is also referred to as the *link-source incidence matrix* in the literature. Without loss of generality, we assume that each source flow traverses at least one link, each link is used by at least one source and the links form a connected graph.

and logarithmic barrier functions for the nonnegativity constraints (which can be done since the feasible set of (2) has a nonempty interior).[4] We denote the new decision vector by $x = ([s_i]'_{i\in\mathcal{S}}, [y_l]'_{l\in\mathcal{L}})'$. This problem can be written as

$$\begin{aligned} \text{minimize} \quad & -\sum_{i=1}^{S} U_i(x_i) - \mu \sum_{i=1}^{S+L} \log(x_i) \\ \text{subject to} \quad & Ax = c, \end{aligned} \tag{4}$$

where A is the $L \times (S+L)$-dimensional matrix given by

$$A = [R \quad I(L)], \tag{5}$$

and μ is a nonnegative barrier function coefficient. We use $f(x)$ to denote the objective function of problem (4), i.e., $f(x) = -\sum_{i=1}^{S} U_i(x_i) - \mu \sum_{i=1}^{S+L} \log(x_i)$, and f^* to denote the optimal value of this problem.

By Assumption 1, the function $f(x)$ is separable, strictly convex, and has a positive definite diagonal Hessian matrix on the positive orthant. Throughout the paper, we assume that $\mu \geq 1$, which guarantees that $f(x)$ is self-concordant, since both summing and scaling by a factor $\mu \geq 1$ preserve self-concordance property. This is without loss of generality since it was shown in [30] that, under self-concordance assumptions, the problem with a general $\mu \geq 0$ can be addressed by solving two instances of problem (4) with different coefficients $\mu \geq 1$.

3 Exact Newton Method

For each fixed $\mu \geq 1$, problem (4) is feasible[5] and has a convex objective function, affine constraints, and a finite optimal value f^*.[6] Therefore, we can use a strong duality theorem to show that, for problem (4), there is no duality gap and there exists a dual optimal solution (see [4]). Moreover, since matrix A has full row rank, we can use a (feasible start) equality-constrained Newton method to solve problem (4)(see [7] Chapter 10). In our iterative method, we use x^k to denote the primal vector at the k^{th} iteration.

3.1 Feasible Initialization

We initialize the algorithm with some feasible and strictly positive vector x^0. For example, one such initial vector is given by

$$\begin{aligned} x_i^0 &= \frac{\underline{c}}{S+1} \quad \text{for } i = 1, 2 \dots S, \\ x_{l+S}^0 &= c_l - \sum_{j=1}^{S} R_{lj} \frac{\underline{c}}{S+1} \quad \text{for } l = 1, 2 \dots L, \end{aligned} \tag{6}$$

where c_l is the finite capacity for link l, $\underline{c}$ is the minimum (positive) link capacity, S is the total number of sources in the network, and R is routing matrix [cf. Eq. (1)].

[4] We adopt the convention that $\log(x) = -\infty$ for $x \leq 0$.

[5] There exists a feasible solution x with $x_i = \underline{c}/(S+1)$ for all $i \in \mathcal{S}$ with $\underline{c} = \min_l\{c_l\}$.

[6] This problem has a feasible solution, hence f^* is upper bounded. Each of the variable x_i is upper bounded by $\bar{c}$, where $\bar{c} = \max_l\{c_l\}$, hence by monotonicity of utility and logarithm functions, the optimal objective function value is lower bounded. Note that in the optimal solution of problem (4) $x_i \neq 0$ for all i, due to the logarithmic barrier functions.

3.2 Iterative Update Rule

Given an initial feasible vector x^0, the algorithm generates the iterates by

$$x^{k+1} = x^k + d^k \Delta x^k, \tag{7}$$

where d^k is a positive stepsize, Δx^k is the (primal) Newton direction given as the solution of the following system of linear equations:[7].

$$\begin{pmatrix} \nabla^2 f(x^k) & A' \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta x^k \\ w^k \end{pmatrix} = - \begin{pmatrix} \nabla f(x^k) \\ 0 \end{pmatrix}. \tag{8}$$

We will refer to x^k as the *primal vector* and w^k as the *dual vector* (and their components as primal and dual variables respectively). We also refer to w^k as the *price vector* since the dual variables $[w_l^k]_{l \in \mathcal{L}}$ associated with the link capacity constraints can be viewed as prices for using links. For notational convenience, we will use $H_k = \nabla^2 f(x^k)$ to denote the Hessian matrix in the rest of the paper.

Solving for Δx^k and w^k in the preceding system yields

$$\Delta x^k = -H_k^{-1}(\nabla f(x^k) + A' w^k), \tag{9}$$

$$(A H_k^{-1} A') w^k = -A H_k^{-1} \nabla f(x^k). \tag{10}$$

This system has a unique solution for all k. To see this, note that the matrix H_k is a diagonal matrix with entries

$$(H_k)_{ii} = \begin{cases} -\frac{\partial^2 U_i(x_i^k)}{\partial x_i^2} + \frac{\mu}{(x_i^k)^2} & 1 \le i \le S, \\ \frac{\mu}{(x_i^k)^2} & S+1 \le i \le S+L. \end{cases} \tag{11}$$

By Assumption 1, the functions U_i are strictly concave, which implies $\frac{\partial^2 U_i(x_i^k)}{\partial x_i^2} \le 0$. Moreover, the primal vector x^k is bounded (since the method maintains feasibility) and, as we shall see in Section 4.5, can be guaranteed to remain strictly positive by proper choice of stepsize. Therefore, the entries $(H_k)_{ii} > 0$ and are well-defined for all i, implying that the Hessian matrix H_k is invertible. Due to the structure of A [cf. Eq. (5)], the column span of A is the entire space $\mathbb{R}^L$, and hence the matrix $A H_k^{-1} A'$ is also invertible.[8] This shows that the preceding system of linear equations can be solved uniquely for all k.

The objective function f is separable in x_i, therefore given the vector w_l^k for l in $L(i)$, the Newton direction Δx_i^k can be computed by each source i using local information available to that source. However, the computation of the vector w^k at a given primal solution x^k cannot be implemented in a decentralized manner since the evaluation of the matrix inverse $(A H_k^{-1} A')^{-1}$ requires global information. The following section provides a distributed inexact Newton method, based on computing the vector w^k using a decentralized iterative scheme.

4 Distributed Inexact Newton Method

In Section 4.1, we describe some preliminaries on matrix splitting techniques. In Section 4.2, we use ideas from matrix splitting to compute the dual vector w^k at each k using a distributed

[7]This is a primal-dual method with the vectors Δx^k and w^k acting as primal direction and dual variables respectively

[8]If for some $x \in \mathbb{R}^L$, we have $A H_k^{-1} A' x = 0$, then $x' A H_k^{-1} A' x = \left|\left| H_k^{-\frac{1}{2}} A' x \right|\right|_2 = 0$, which implies $||A'x||_2 = 0$, because the matrix H is invertible. The rows of the matrix A' span $\mathbb{R}^L$, therefore we have $x = 0$. This shows that the matrix $A H_k^{-1} A'$ is invertible.

iterative scheme, which introduces inexactness in the dual variables due to finite termination. In Section 4.3, we present the decentralized computation of the primal Newton direction given the dual vector. We also impose some error tolerance levels on the resulting primal iterates, which are necessary to establish quadratic rate of convergence in [30]. In Section 4.4, we develop a distributed error checking procedure to guarantee the error tolerance levels are satisfied. In Section 4.5, we propose a stepsize rule and provide a distributed procedure to compute it in finitely many iterations. This particular stepsize choice can maintain feasibility (see Theorem 4.12) and achieve quadratic rate of convergence (see [30]).

4.1 Preliminaries on Matrix Splitting

Matrix splitting can be used to solve a system of linear equations given by

$$Gy = a,$$

where G is an $n \times n$ matrix and a is an n-dimensional vector. Suppose that the matrix G can be expressed as the sum of an invertible matrix M and a matrix N, i.e.,

$$G = M + N. \tag{12}$$

Let y_0 be an arbitrary n-dimensional vector. A sequence $\{y^k\}$ can be generated by the following iteration:

$$y^{k+1} = -M^{-1}Ny^k + M^{-1}a. \tag{13}$$

It can be seen that the sequence $\{y^k\}$ converges as $k \to \infty$ if and only if the spectral radius of the matrix $M^{-1}N$ is strictly bounded above by 1. When the sequence $\{y^k\}$ converges, its limit y^* solves the original linear system, i.e., $Gy^* = a$ (see [2] and [11] for more details). Hence, the key to solving the linear equation via matrix splitting is the bound on the spectral radius of the matrix $M^{-1}N$. Such a bound can be obtained using the following result (see Theorem 2.5.3 from [11]).

Theorem 4.1. Let G be a real symmetric matrix. Let M and N be matrices such that $G = M + N$ and assume that M is invertible and both matrices $M + N$ and $M - N$ are positive definite. Then the spectral radius of $M^{-1}N$, denoted by $\rho(M^{-1}N)$, satisfies $\rho(M^{-1}N) < 1$.

By the above theorem, if G is a real, symmetric, positive definite matrix and M is a nonsingular matrix, then one sufficient condition for the iteration (13) to converge is that the matrix $M - N$ is positive definite. This can be guaranteed using Gershgorin Circle Theorem, which we introduce next (see [29] for more details).

Theorem 4.2. (Gershgorin Circle Theorem) Let G be an $n \times n$ matrix, and define $r_i(G) = \sum_{j \neq i} |G_{ij}|$. Then, each eigenvalue of G lies in one of the Gershgorin sets $\{\Gamma_i\}$, with Γ_i defined as disks in the complex plane, i.e.,

$$\Gamma_i = \{z \in \mathbb{C} \mid |z - G_{ii}| \leq r_i(G)\}.$$

One corollary of the above theorem is that if a matrix G is strictly diagonally dominant, i.e., $|G_{ii}| > \sum_{j \neq i} |G_{ij}|$, and $G_{ii} > 0$ for all i, then the real parts of all the eigenvalues lie in the positive half of the real line, and thus the matrix is positive definite. Hence a sufficient condition for the matrix $M - N$ to be positive definite is that $M - N$ is strictly diagonally dominant with strictly positive diagonal entries.

4.2 Distributed Computation of the Dual Vector

We use the matrix splitting scheme introduced in the preceding section to compute the dual vector w^k in Eq. (10) in a distributed manner for each primal iteration k. For notational convenience, we may suppress the explicit dependence of w^k on k. Let D_k be a diagonal matrix, with diagonal entries

$$(D_k)_{ll} = (AH_k^{-1}A')_{ll}, \tag{14}$$

and matrix B_k be given by

$$B_k = AH_k^{-1}A' - D_k. \tag{15}$$

Let matrix $\bar{B}_k$ be a diagonal matrix, with diagonal entries

$$(\bar{B}_k)_{ii} = \sum_{j=1}^{L} (B_k)_{ij}. \tag{16}$$

By splitting the matrix $AH_k^{-1}A'$ as the sum of $D_k + \bar{B}_k$ and $B_k - \bar{B}_k$, we obtain the following result.

Theorem 4.3. For a given $k > 0$, let D_k, B_k, $\bar{B}_k$ be the matrices defined in Eqs. (14), (15) and (16). Let $w(0)$ be an arbitrary initial vector and consider the sequence $\{w(t)\}$ generated by the iteration

$$w(t+1) = (D_k + \bar{B}_k)^{-1}(\bar{B}_k - B_k)w(t) + (D_k + \bar{B}_k)^{-1}(-AH_k^{-1}\nabla f(x^k)), \tag{17}$$

for all $t \geq 0$. Then the spectral radius of the matrix $(D_k + \bar{B}_k)^{-1}(B_k - \bar{B}_k)$ is strictly bounded above by 1 and the sequence $\{w(t)\}$ converges as $t \to \infty$, and its limit is the solution to Eq. (10).

Proof. We split the matrix $AH_k^{-1}A'$ as

$$(AH_k^{-1}A') = (D_k + \bar{B}_k) + (B_k - \bar{B}_k) \tag{18}$$

and use the iterative scheme presented in Eqs. (12) and (13) to solve Eq. (10). For all k, both the real matrix H_k and its inverse, H_k^{-1}, are positive definite and diagonal. The matrix A has full row rank and is element-wise nonnegative. Therefore the product $AH_k^{-1}A'$ is real, symmetric, element-wise nonnegative and positive definite. We let

$$Q_k = (D_k + \bar{B}_k) - (B_k - \bar{B}_k) = D_k + 2\bar{B}_k - B_k \tag{19}$$

denote the difference matrix. By definition of $\bar{B}_k$ [cf. Eq. (16)], the matrix $2\bar{B}_k - B_k$ is diagonally dominant, with nonnegative diagonal entries. Moreover, due to strict positivity of the second derivatives of the logarithmic barrier functions, we have $(D_k)_{ii} > 0$ for all i. Therefore the matrix Q_k is strictly diagonally dominant. By Theorem 4.2, such matrices are positive definite. Therefore, by Theorem 4.1, the spectral radius of the matrix $(D_k + \bar{B}_k)^{-1}(B_k - \bar{B}_k)$ is strictly bounded above by 1. Hence the splitting scheme (18) guarantees the sequence $\{w(t)\}$ generated by iteration (17) to converge to the solution of Eq. (10). □

This provides an iterative scheme to compute the dual vector w^k at each primal iteration k using an iterative scheme. We will refer to the iterative scheme defined in Eq. (17) as the *dual iteration*.

There are many ways to split the matrix $AH_k^{-1}A'$. The particular one in Eq. (18) is chosen here due to two desirable features. First it guarantees that the difference matrix Q_k [cf. Eq. (19)]

is strictly diagonally dominant, and hence ensures convergence of the sequence $\{w(t)\}$. Second, with this splitting scheme, the matrix $D_k + \bar{B}_k$ is diagonal, which eliminates the need for global information when calculating its inverse.

We next rewrite iteration (17), analyze the information exchange required to implement it and develop a distributed computation procedure to calculate the dual vector. For notational convenience, we define the *price of the route* for source i, $\pi_i(t)$, as the sum of the dual variables associated with links used by source i at the t^{th} dual iteration, i.e., $\pi_i(t) = \sum_{l \in L(i)} w_l(t)$. Similarly, we define the *weighted price of the route* for source i, $\Pi_i(t)$, as the price of the route for source i weighted by the ith diagonal element of the inverse Hessian matrix, i.e., $\Pi_i(t) = (H_k^{-1})_{ii} \sum_{l \in L(i)} w_l(t)$.

Lemma 4.4. For each primal iteration k, the dual iteration (17) can be written as

$$\begin{aligned} w_l(t+1) =& \frac{1}{(H_k)^{-1}_{(S+l)(S+l)} + \sum_{i \in S(l)} \Pi_i(0)} \Big(\Big(\sum_{i \in S(l)} \Pi_i(0) - \sum_{i \in S(l)} (H_k)^{-1}_{ii} \Big) w_l(t) - \sum_{i \in S(l)} \Pi_i(t) \\ &+ \sum_{i \in S(l)} (H_k)^{-1}_{ii} w_l(t) - \sum_{i \in S(l)} (H_k^{-1})_{ii} \nabla_i f(x^k) - (H_k^{-1})_{(S+l)(S+l)} \nabla_{S+l} f(x^k) \Big), \end{aligned} \tag{20}$$

where $\Pi_i(0)$ is the weighted price of the route for source i when $w(0) = [1, 1 \ldots, 1]'$.

Proof. Recall the definition of matrix A, i.e., $A_{li} = 1$ for $i = 1, 2 \ldots S$ if source i uses link l, i.e., $i \in S(l)$, and $A_{li} = 0$ otherwise. Therefore, we can write the price of the route for source i as, $\pi_i(t) = \sum_{l=1}^{L} A_{li} w(t)_l = [A']^i w(t)$. Similarly, since the Hessian matrix H_k is diagonal, the weighted price can be written as

$$\Pi_i(t) = (H_k)^{-1}_{ii} [A']^i w(t) = [H_k^{-1} A']^i w(t). \tag{21}$$

On the other hand, since $A = [R \; I(L)]$, where R is the routing matrix, we have

$$\begin{aligned} (AH_k^{-1}A'w(t))_l &= \sum_{i=1}^{S} ([A]_i [H_k^{-1}A']^i w(t))_l + (H_k^{-1})_{(S+l)(S+l)} w_l(t) \\ &= \sum_{i=1}^{S} A_{li} ([H_k^{-1}A']^i w(t)) + (H_k^{-1})_{(S+l)(S+l)} w_l(t). \end{aligned}$$

Using the definition of the matrix A one more time, this implies

$$\begin{aligned} (AH_k^{-1}A'w(t))_l &= \sum_{i \in S(l)} [H_k^{-1}A']^i w(t) + (H_k^{-1})_{(S+l)(S+l)} w_l(t) \\ &= \sum_{i \in S(l)} \Pi_i(t) + (H_k^{-1})_{(S+l)(S+l)} w_l(t), \end{aligned} \tag{22}$$

where the last equality follows from Eq. (21).

Using Eq. (15), the above relation implies that $((B_k + D_k)w(t))_l = \sum_{i \in S(l)} \Pi_i(t) + (H_k^{-1})_{(S+l)(S+l)} w_l(t)$. We next rewrite $(\bar{B}_k)_{ll}$. Using the fact that $w(0) = [1, 1 \ldots, 1]'$, we have

$$(AH_k^{-1}A'w(0))_l = ((B_k + D_k)w(0))_l = \sum_{j=1}^{L} (B_k)_{lj} + (D_k)_{ll}.$$

Using the definition of $\bar{B}_k$ [cf. Eq. (16)], this implies

$$(\bar{B}_k)_{ll} = \sum_{j=1}^{L}(B_k)_{lj} = (AH_k^{-1}A'w(0))_l - (D_k)_{ll} = \sum_{i\in S(l)} \Pi_i(0) + (H_k^{-1})_{(S+l)(S+l)} - (D_k)_{ll}.$$

This calculation can further be simplified using

$$(D_k)_{ll} = (AH_k^{-1}A')_{ll} = \sum_{i\in S(l)} (H_k)_{ii}^{-1} + (H_k)^{-1}_{(S+l)(S+l)}, \tag{23}$$

[cf. Eq. (14)], yielding

$$(\bar{B}_k)_{ll} = \sum_{i\in S(l)} \Pi_i(0) - \sum_{i\in S(l)} (H_k)_{ii}^{-1}. \tag{24}$$

Following the same argument, the value $(B_k w(t))_l$ for all t can be written as

$$\begin{aligned}(B_k w(t))_l &= (AH_k^{-1}A'w(t))_l - (D_k w(t))_l \\ &= \sum_{i=1}^{S} \Pi_i(t) + (H_k^{-1})_{(S+l)(S+l)} w_l(t) - (D_k)_{ll} w_l(t) \\ &= \sum_{i=1}^{S} \Pi_i(t) - \sum_{i\in S(l)} (H_k)_{ii}^{-1} w_l(t),\end{aligned}$$

where the first equality follows from Eq. (16), the second equality follows from Eq. (22), and the last equality follows from Eq. (23).

Finally, we can write $(AH_k^{-1}\nabla f(x^k))_l$ as

$$(AH_k^{-1}\nabla f(x^k))_l = \sum_{i\in S(l)} (H_k^{-1})_{ii}\nabla_i f(x^k) + (H_k^{-1})_{(S+l)(S+l)}\nabla_{S+l} f(x^k).$$

Substituting the preceding into (17), we obtain the desired iteration (20). □

We next analyze the information exchange required to implement iteration (20) among sources and links in the network. We first observe the local information available to sources and links. Each source i knows the ith diagonal entry of the Hessian $(H_k)_{ii}$ and the ith component of the gradient $\nabla_i f(x^k)$. Similarly, each link l knows the $(S+l)$th diagonal entry of the Hessian $(H_k)_{S+l,S+l}$ and the $(S+l)$th component of the gradient $\nabla_{S+l} f(x^k)$. In addition to the locally available information, each link l, when executing iteration (20), needs to compute the terms:

$$\sum_{i\in S(l)} (H_k)_{ii}^{-1}, \qquad \sum_{i\in S(l)} (H_k^{-1})_{ii}\nabla_i f(x^k), \qquad \sum_{i\in S(l)} \Pi_i(0), \qquad \sum_{i\in S(l)} \Pi_i(t).$$

The first two terms can be computed by link l if each source sends its local information to the links along its route "once" in primal iteration k. Similarly, the third term can be computed by link l once for every k if the route price $\pi_i(0) = \sum_{l\in L(i)} 1$ (aggregated along the links of a route when link prices are all equal to 1) are sent by the destination to source i, which then evaluates and sends the weighted price $\Pi_i(0)$ to the links along its route. The fourth term can be computed with a similar feedback mechanism, however the computation of this term needs to be repeated for every dual iteration t.

The preceding information exchange suggests the following distributed implementation of (20) (at each primal iteration k) among the sources and the links, where each source or link is viewed as a processor, information available at source i can be passed to the links it traverses, i.e., $l \in L(i)$, and information about the links along a route can be aggregated and sent back to the corresponding source using a feedback mechanism:

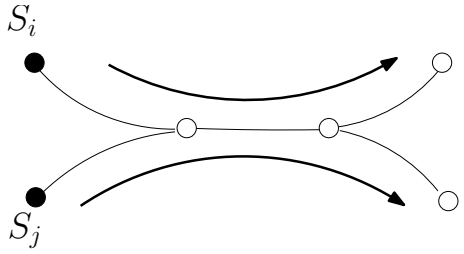


Figure 1: Direction of information flow for the steps 1.a, 1.c and 2.b, from sources to the links they use.

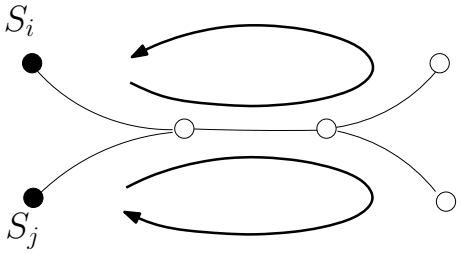


Figure 2: Direction of flow for the steps 1.b and 2.a, from links to the sources using them.

1. **Initialization.**

   1.a Each source i sends its local information $(H_k)_{ii}$ and $\nabla_i f(x^k)$ to the links along its route, $l \in L(i)$. Each link l computes $(H_k)^{-1}_{(S+l)(S+l)}$, $\sum_{i\in S(l)}(H_k)^{-1}_{ii}$, $(H_k^{-1})_{(S+l)(S+l)}\nabla_{S+l} f(x^k)$ and $\sum_{i\in S(l)}(H_k^{-1})_{ii}\nabla_i f(x^k)$.

   1.b Each link l starts with price $w_l(0) = 1$. The link prices $w_l(0)$ are aggregated along route i to compute $\pi(0) = \sum_{l\in L(i)} w_l(0)$ at the destination. This information is sent back to source i.

   1.c Each source computes the weighted price $\Pi_i(0) = (H_k^{-1})_{ii}\sum_{l\in L(i)} w_l(0)$ and sends it to the links along its route, $l \in L(i)$.

   1.d Each link l then initializes with arbitrary price $w_l(1)$.

2. **Dual Iteration.**

   2.a The link prices $w_l(t)$ are updated using (20) and aggregated along route i to compute $\pi(t)$ at the destination. This information is sent back to source i.

   2.b Each source computes the weighted price $\Pi_i(t)$ and sends it to the links along its route, $l \in L(i)$.

The direction of information flow can be seen in Figures 1 and 2. Note that the sources need to send their Hessian and gradient information once per primal iteration since these values do not change in the dual iterations. Moreover, this algorithm has comparable level of information exchange with the subgradient based algorithms applied to the NUM problem (2) (see [1], [16], [19], [20] for more details). In both types of algorithms, only the sum of prices of links along a route is fed back to the source, and the links update prices based on scalar information sent from sources using that link.

4.3 Distributed Computation of the Primal Newton Direction

Given the dual vector $w^k(t)$ obtained from the dual iteration (17) in finitely many steps, the primal Newton direction can be computed according to Eq. (9) as

$$(\Delta x^k)_i = -(H_k)^{-1}_{ii}(\nabla_i f(x^k) + (A' w^k(t))_i) = -(H_k)^{-1}_{ii}\nabla_i f(x^k) + \Pi_i(t), \tag{25}$$

where $\Pi_i(t)$ is the weighted price of the route for source i computed at termination of the dual iteration. Hence, the primal Newton direction can be computed using local information by each source. However, because the dual variable computation involves an iterative scheme, the exact value for w^k is not available. Therefore, the direction Δx^k computed using Eq. (25) may violate

the equality constraints in problems (4). To maintain feasibility of the generated primal vectors, the calculation of the *inexact Newton direction* at a primal vector x^k, which we denote by $\Delta\tilde{x}^k$, is separated into two stages.

In the first stage, the first S components of $\Delta\tilde{x}^k$, denoted by $\Delta\tilde{s}^k$, is computed via Eq. (25) using the dual variables obtained via the iterative scheme, i.e.,

$$\Delta\tilde{s}_i^k = -(H_k)_{ii}^{-1}(\nabla_i f(x^k) + [R']^i w^k(t). \tag{26}$$

In the second stage, the last L components of $\Delta\tilde{x}^k$ (corresponding to the slack variables, cf. Eq. (3)) are computed to ensure that the condition $A\Delta\tilde{x}^k = 0$ is satisfied, i.e.

$$\Delta\tilde{x}^k = \begin{pmatrix} \Delta\tilde{s}^k \\ -R\Delta\tilde{s}^k \end{pmatrix}, \tag{27}$$

which implies $\Delta\tilde{x}^k_{l+S} = -\sum_{i\in S(l)} \Delta\tilde{s}^k_i$ for each link l. This calculation at each link l involves computing the slack introduced by the components of $\Delta\tilde{s}^k$ corresponding to the sources using link l, and hence can be computed in a distributed way.

The algorithm presented generates the primal vectors as follows: Let x^0 be an initial strictly positive feasible primal vector (see Eq. (6) for one possible choice). For any $k \geq 0$, we have

$$x^{k+1} = x^k + d^k\Delta\tilde{x}^k, \tag{28}$$

where d^k is a positive stepsize and $\Delta\tilde{x}^k$ is the inexact Newton direction at primal vector x^k (obtained through an iterative dual variable computation scheme and a two-stage primal direction computation that maintains feasibility). We will refer to this algorithm as the *(distributed) inexact Newton method*.

The method is inexact due to the iterative computation of the dual vector w^k and the modification we use to maintain feasibility. As shown in [30], if the following bounds on the errors are satisfied, the objective function value generated by the inexact Newton algorithm (with selection of proper stepsize) converges quadratically in terms of primal iterations to an error neighborhood of the optimal value, where the neighborhood can be characterized explicitly using the parameters of the algorithm and the error bounds.

Assumption 2. Let $\{x^k\}$ denote the sequence of primal vectors generated by the distributed inexact Newton method. Let Δx^k and $\Delta\tilde{x}^k$ denote the exact and inexact Newton directions at x^k, and γ^k denote the error in the Newton direction computation, i.e.,

$$\Delta x^k = \Delta\tilde{x}^k + \gamma^k. \tag{29}$$

For all k, γ^k satisfies

$$|(\gamma^k)'\nabla^2 f(x^k)\gamma^k| \leq p^2(\Delta\tilde{x}^k)'\nabla^2 f(x^k)\Delta\tilde{x}^k + \epsilon. \tag{30}$$

for some positive scalars $p < 1$ and ϵ.

This assumption imposes a bound on the weighted norm of the Newton direction error γ^k as a function of the weighted norm of $\Delta\tilde{x}^k$ and a constant ϵ. Note that without the constant ϵ, we would require this error to vanish when x^k is close to the optimal solution, *i.e.*, when $\Delta\tilde{x}^k$ is small, which is impractical for implementation purposes. Given p and ϵ, one can use the distributed scheme presented in the following section to guarantee the preceding assumption is satisfied.

4.4 Distributed Error Checking

In this section, we present a distributed error checking method to determine when to terminate the dual computation procedure to meet the error tolerance level in Assumption 2 at a fixed primal iteration k. The method involves two stages: in the first stage, the links and sources execute a predetermined number of dual iterations. In the second stage, if the error tolerance level has yet not been satisfied, the links and sources implement dual iterations until some distributed termination criteria is met. For the rest of this section, we once again suppress the dependence of the dual vector on the primal iteration index k for notational convenience and we adopt the following assumption on the information available to each node and link.

Assumption 3. There exists a positive scalar $F < 1$ such that the spectral radius of the matrix $M = (D_k + \bar{B}_k)^{-1}(\bar{B}_k - B_k)$ satisfies $\rho(M) \leq F$. Each source and link knows the scalar F and the total number of sources and links in the graph, denoted by S and L respectively.

The value F can vary across different primal iterations. As observed in [30], the bound F coincides with a bound on a largest eigenvalue of a Laplacian matrix of a graph related to the network topology. These bounds can be obtained based on the known results from graph theory [10], [5]. In this assumption, we only require availability of some aggregate information, and hence the distributed nature of the algorithm is preserved. Before we introduce the details of the distributed error checking method, we establish a relation between $||w^* - w(t)||_\infty$ and $||w(t+1) - w(t)||_\infty$, which is essential in developing the method.

Lemma 4.5. Let the matrix M be $M = (D_k + \bar{B}_k)^{-1}(\bar{B}_k - B_k)$. Let $w(t)$ denote the dual variable generated by iteration (17), and w^* be the fixed point of the iteration. Let F and L be the positive scalar defined in Assumption 3. Then the following relation holds,

$$||w^* - w(t)||_\infty \leq \frac{\sqrt{L}}{1-F} ||w(t+1) - w(t)||_\infty . \tag{31}$$

Proof. Iteration (17) implies that the fixed point w^* satisfies the following relation,

$$w^* = Mw^* + (D_k + \bar{B}_k)^{-1}(-AH_k^{-1}\nabla f(x^k)),$$

and the iterates $w(t)$ satisfy,

$$w(t+1) = Mw(t) + (D_k + \bar{B}_k)^{-1}(-AH_k^{-1}\nabla f(x^k)).$$

By combining the above two relations, we obtain,

$$w(t+1) - w(t) = (I - M)(w^* - w(t)).$$

Hence, by the definition of matrix infinity norm, we have

$$||w^* - w(t)||_\infty \leq \left|\left|(1-M)^{-1}\right|\right|_\infty ||w(t+1) - w(t)||_\infty .$$

Using norm equivalence for finite dimensional Euclidean space and theories of linear algebra, we obtain $||(I-M)^{-1}||_\infty \leq \sqrt{L}\,||(I-M)^{-1}||_2 \leq \frac{\sqrt{L}}{1-||M||_2}$ [12], [2]. For the symmetric real matrix M we have $\rho(M) = ||M||_2$, and hence we obtain the desired relation. □

We next use the above lemma to develop two theorems, each of which corresponds to one stage of the distributed error checking method.

Theorem 4.6. Let $\{x^k\}$ be the primal sequence generated by the inexact Newton method (28) and H_k be the corresponding Hessian matrix at the k^{th} iteration. Let $w(t)$ be the inexact dual variable obtained after t dual iterations (17) and w^* be the exact solution to (10), i.e. the limit of the sequence $\{w(t)\}$ as $t \to \infty$. Let vectors Δx^k and $\Delta \tilde{x}^k$ be the exact and inexact Newton directions obtained using w^* and $w(t)$ [cf. Eqs. (26)-(27)], and vector γ^k be the error in the Newton direction computation at x^k , defined by $\gamma^k = \Delta x^k - \Delta \tilde{x}^k$. For some positive scalar p, let

$$\rho_i = \left| \frac{\sqrt{L}(H_k^{-1})_{ii}|L(i)|\,||w(t+1) - w(t)||_\infty}{(1-F)(H_k^{-1})_{ii}[R']^i w(t)} \right| \tag{32}$$

for each source i, and

$$\rho_l = \left| \frac{\sqrt{L}\sum_{i\in S(l)}(H_k^{-1})_{ii}|L(i)|\,||w(t+1) - w(t)||_\infty}{(1-F)\sum_{i\in S(l)}(H_k^{-1})_{ii}[R']^i w(t)} \right| \tag{33}$$

for each link l. Define a nonnegative scalar β^k as

$$\beta^k = \left(\max\left\{ \max_{i\in\mathcal{S}} \frac{\rho_i}{p}, \max_{l\in\mathcal{L}} \frac{\rho_l}{p} \right\} \right)^{-2}. \tag{34}$$

Then we have

$$\beta^k (\gamma^k)' H_k \gamma^k \leq p^2 (\Delta\tilde{x}^k)' H_k (\Delta\tilde{x}^k). \tag{35}$$

Proof. For notational convenience, we let matrix P_k denote the $S \times S$ principal submatrix of H_k, i.e. $(P_k)_{ii} = (H_k)_{ii}$ for $i \leq S$, vector $\Delta\tilde{s}$ in $\mathbb{R}^S$ denote the first S components of the vector $\Delta\tilde{x}^k$, vector $\Delta\tilde{y}_l$ in $\mathbb{R}^L$ denote the last L components of the vector $\Delta\tilde{x}^k$. Similarly, we denote by Δs and Δy the first S and last L components of the exact Newton direction Δx respectively. From Eq. (26), we have for each $i \in S$,

$$\begin{aligned}
\left| \frac{\Delta s_i - \Delta\tilde{s}_i}{\Delta\tilde{s}_i} \right| &= \left| \frac{(H_k^{-1})_{ii}[R']^i (w^* - w(t))}{(H_k^{-1})_{ii}[R']^i w(t)} \right| \\
&\leq \left| \frac{(H_k^{-1})_{ii}[R']^i e\,||w^* - w(t)||_\infty}{(H_k^{-1})_{ii}[R']^i w(t)} \right| \\
&\leq \left| \frac{\sqrt{L}(H_k^{-1})_{ii}|L(i)|\,||w(t+1) - w(t)||_\infty}{(1-F)(H_k^{-1})_{ii}[R']^i w(t)} \right| = \rho_i,
\end{aligned}$$

where the first inequality follows from the element-wise nonnegativity of matrices H_k and R, and the second inequality follows from relation (31).

Similarly for each link $l \in L$, by relations (26) and (27) we obtain

$$\begin{aligned}
\left| \frac{\Delta y_l - \Delta\tilde{y}_l}{\Delta\tilde{y}_l} \right| &= \left| \frac{[R]^l P_k^{-1} R'(w^* - w(t))}{[R]^l P_k^{-1} R' w(t)} \right| \\
&\leq \left| \frac{\sum_{i\in S(l)}(P_k^{-1})_{ii} R' e\,||w^* - w(t)||_\infty}{\sum_{i\in S(l)}(P_k^{-1})_{ii}[R']^i w(t)} \right| \\
&\leq \left| \frac{\sqrt{L}\sum_{i\in S(l)}(H_k^{-1})_{ii}|L(i)|\,||w(t+1) - w(t)||_\infty}{(1-F)\sum_{i\in S(l)}(H_k^{-1})_{ii}[R']^i w(t)} \right| = \rho_l,
\end{aligned}$$

where the first inequality follows from the structure of the matrix R and the element-wise nonnegativity of matrices H_k and R, and the second inequality follows from relation (31) and the definition for matrix P_k.

The definition for β^k [cf. Eq. (34)] implies that

$$\frac{p}{\sqrt{\beta^k}} = \max\left\{\max_{i\in\mathcal{S}} \rho_i, \max_{l\in\mathcal{L}} \rho_l\right\}.$$

Therefore the preceding relations imply that $\left|\frac{\Delta s_i - \Delta\tilde{s}_i}{\Delta\tilde{s}_i}\right| \leq \frac{p}{\sqrt{\beta^k}}$ and $\left|\frac{\Delta y_l - \Delta\tilde{y}_l}{\Delta\tilde{y}_l}\right| \leq \frac{p}{\sqrt{\beta^k}}$, i.e.,

$$\sqrt{\beta^k}\left|\gamma_i^k\right| \leq p|\Delta\tilde{x}_i|,$$

which implies the desired relation.

□

Theorem 4.7. Let $\{x^k\}$ be the primal sequence generated by the inexact Newton method (28) and H_k be the corresponding Hessian matrix at k^{th} iteration. Let $w(t)$ be the inexact dual variable obtained after t dual iterations (17) and w^* be the exact solution to (10), i.e. the limit of the sequence $\{w(t)\}$ as $t \to \infty$. Let vectors Δx^k and $\Delta\tilde{x}^k$ be the exact and inexact Newton directions obtained using w^* and $w(t)$ [cf. Eqs. (26)-(27)] respectively, and vector γ^k be the error in the Newton direction computation at x^k , defined by $\gamma^k = \Delta x^k - \Delta\tilde{x}^k$. For some scalar β and ϵ where $0 < \beta^k < 1$ and $\epsilon > 0$, let

$$h_i = \sqrt{\frac{\epsilon}{(1-\beta^k)(L+S)L}}\frac{1-F}{|L(i)|(H_k^{-\frac{1}{2}})_{ii}} \tag{36}$$

for each source i, and

$$h_l = \sqrt{\frac{\epsilon}{(1-\beta^k)(L+S)L}}\frac{1-F}{(H_k^{\frac{1}{2}})_{(S+l)(S+l)}\sum_{i\in S(L)}|L(i)|(H_k)_{ii}^{-1}} \tag{37}$$

for each link l. Define a nonnegative scalar h as

$$h = \left(\min\left\{\min_{i\in\mathcal{S}} h_i, \min_{l\in\mathcal{L}} h_l\right\}\right). \tag{38}$$

Then the condition

$$||w(t+1) - w(t)||_\infty \leq h \tag{39}$$

implies

$$(\gamma^k)' H_k \gamma^k \leq \frac{\epsilon}{1-\beta^k}, \tag{40}$$

for $\Delta\tilde{x}^k$ obtained according to (27) using $w(t)$.

Proof. We let matrix P_k denote the $S \times S$ principal submatrix of H_k, i.e. $(P_k)_{ii} = (H_k)_{ii}$ for $i \leq S$, for notational convenience. The definition of h [cf. Eq. (38)] and relation (39) implies

$$||w(t+1) - w(t)||_\infty \leq h_i, \in \mathcal{S},$$

and

$$||w(t+1) - w(t)||_\infty \leq h_l, \quad \text{for l } \in \mathcal{L},$$

Using relation (31) and the definition of h_i and h_l [cf. Eqs. (36) and (37)], the above two relations implies respectively that

$$||w^* - w(t)||_\infty \leq \sqrt{\frac{\epsilon}{(1-\beta^k)(L+S)}}\frac{1}{|L(i)|(H_k^{-\frac{1}{2}})_{ii}}, \quad \text{for i } \in \mathcal{S}, \tag{41}$$

and

$$||w^* - w(t)||_\infty \leq \sqrt{\frac{\epsilon}{(1-\beta^k)(L+S)}} \frac{1}{(H_k^{\frac{1}{2}})_{(S+l)(S+l)} \sum_{i \in S(L)} |L(i)| (H_k)_{ii}^{-1}}, \quad \text{for l } \in \mathcal{L}. \tag{42}$$

By using the element-wise nonnegativity of matrices H and A, we have for each source i,

$$\left| (H_k^{-\frac{1}{2}})_{ii} [R']^i (w^* - w(t)) \right| \leq (H_k^{-\frac{1}{2}})_{ii} [R']^i e \, ||w^* - w(t)||_\infty = |L(i)| (H_k^{-\frac{1}{2}})_{ii} \, ||w^* - w(t)||_\infty ,$$

where the last equality follows from the fact that $[R']^i e = |L(i)|$ for each source i.

The above inequality and relation (41) imply

$$\left| (H_k^{-\frac{1}{2}})_{ii} [R']^i (w^* - w(t)) \right| \leq \sqrt{\frac{\epsilon}{(1-\beta^k)(L+S)}}. \tag{43}$$

By the definition of matrices P_k and R, we have for each link l,

$$\begin{aligned} | \, (H_k^{\frac{1}{2}})_{(S+l)(S+l)} \, (\, RP_k^{-1} R'(w^* - w(t)) \,)^l \, | &\leq (H_k^{\frac{1}{2}})_{(S+l)(S+l)} [R]^l P_k^{-1} R' e \, ||w^* - w(t)||_\infty \\ &= (H_k^{\frac{1}{2}})_{(S+l)(S+l)} \sum_{i \in S(l)} |L(i)| (H_k^{-1})_{ii} \, ||w^* - w(t)||_\infty . \end{aligned}$$

When combined with relation (42), the preceding relation yields

$$\left| (H_k^{\frac{1}{2}})_{(S+l)(S+l)} \left(RP_k^{-1} R'(w^* - w(t)) \right)^l \right| \leq \sqrt{\frac{\epsilon}{(1-\beta^k)(L+S)}}. \tag{44}$$

From Eqs. (26)-(27) and the definition of γ, we have

$$\gamma_i^k = - \begin{pmatrix} P_k^{-1} R'(w^* - w(t)) \\ RP_k^{-1} R'(w^* - w(t)) \end{pmatrix},$$

which implies that

$$\begin{aligned} (\gamma^k)' H_k \gamma^k &= \sum_{i \in \mathcal{S}} \left((H_k^{-\frac{1}{2}})_{ii} [R']^i (w^* - w(t)) \right)^2 + \sum_{l \in \mathcal{L}} \left((H_k^{\frac{1}{2}})_{(S+l)(S+l)} \left(RP_k^{-1} R'(w^* - w(t)) \right)^l \right)^2 \\ &\leq \frac{\epsilon}{1-\beta^k}, \end{aligned}$$

where the inequality follows from (43), (44), which establishes the desired relation.

□

We develop the distributed error checking method based on the preceding two theorems:

- Stage 1: The links and sources implement T iterations of (17), where T is a predetermined globally known constant. The links and sources then use Theorem 4.6 with $t = T - 1$ and p as the desired relative error tolerance level defined in Assumption 2 to obtain a value β^k. If $\beta^k \geq 1$, then the dual iteration terminates.

- Stage 2: The links and sources use Theorem 4.7 with β^k obtained in the first stage and ϵ defined in Assumption 2 to obtain value h. Then they perform more iterations of the form (17) until the criterion (39) is satisfied.[9]

[9]The error tolerance level will terminate after finite number of iterations for any $h > 0$, due to the convergence of the sequence $w(t)$ established in Theorem 4.3 .

Stage 1 corresponds to checking the term $p^2(\Delta\tilde{x}^k)'H_k(\Delta\tilde{x}^k)$, while Stage 2 corresponds to the term ϵ in the error tolerance level. If the method terminates the dual iterations in Stage 1, then Theorem 4.6 suggests that Assumption 2 is satisfied for any $\epsilon > 0$; otherwise, by combining relations (35) and (40), we have

$$(\gamma^k)'H_k\gamma^k = (\beta^k + (1-\beta^k))(\gamma^k)'H_k\gamma^k \leq p^2(\Delta\tilde{x}^k)'H_k(\Delta\tilde{x}^k) + \epsilon,$$

which shows that the error tolerance level in Assumption 2 is satisfied.

To show that the above method can be implemented in a distributed way, we first rewrite the terms ρ_i, ρ_l, h_i and h_l and analyze the information required to compute them in a decentralized way. We use the definition of the weighted price of the route $\Pi_i(t)$ and obtain $\Pi_i(t) = (H_k^{-1})_{ii}\sum_{l\in L(i)} w(t) = (H_k^{-1})_{ii}[R']^i w(t)$ and $\Pi_i(0) = (H_k^{-1})_{ii}|L(i)|$, where $w_l(0) = 1$ for all links l. Therefore relations (32) and (33) can be rewritten as

$$\rho_i = \left| \frac{\sqrt{L}\Pi_i(0)\,||w(t+1) - w(t)||_\infty}{(1-F)\Pi_i(t)} \right|,$$

$$\rho_l = \left| \frac{\sqrt{L}\sum_{i\in S(l)}\Pi_i(0)\,||w(t+1) - w(t)||_\infty}{(1-F)\sum_{i\in S(l)}\Pi_i(t)} \right|.$$

Similarly, relations (36) and (37) can be transformed into

$$h_i = \sqrt{\frac{\epsilon}{(1-\beta^k)(L+S)L}}\frac{1-F}{\pi_i(0)(H_k^{-\frac{1}{2}})_{ii}},$$

$$h_l = \sqrt{\frac{\epsilon}{(1-\beta^k)(L+S)L}}\frac{1-F}{(H_k^{\frac{1}{2}})_{(S+l)(S+l)}\sum_{i\in S(L)}\Pi_i(0)}.$$

In our dual variable computation procedure, the values $\pi_i(0)$, $\Pi_i(0)$ and $\Pi_i(t)$ are readily available to all the links $l \in L(i)$ through the feedback mechanism described in Section 4.2. Each source and link knows its local Hessian, i.e., $(H_k)_{ii}$ for source i and $(H_k)_{(S+l)(S+l)}$ for link l. The value β^k is only used in Stage 2 and it is available from the previous stage. Therefore in the above four expressions, the only not immediately available information is $||w(t+1) - w(t)||_\infty$, which can be obtained using a maximum consensus algorithm.[10] Based on these four terms, the values of β^k and h can be obtained using once again maximum consensus and hence all the components necessary for the error checking method can be computed in a distributed way.

We observe that in the first T iterations, i.e., Stage 1, only two executions of maximum consensus algorithms is required, where one is used to compute $||w(t+1) - w(t)||_\infty$ and the other for β^k. On the other hand, even though the computation of the value h in Stage 2 needs only one execution of the maximum consensus algorithm, the term $||w(t+1) - w(t)||_\infty$ needs to be computed at each dual iteration t. Therefore the error checking in Stage 1 can be completed much more efficiently than in Stage 2. Hence, when we design values p and ϵ in Assumption 2, we should choose p to be relatively large whenever possible, which results in an error checking method that does not enter Stage 2 frequently, and is hence faster. Other than the distributed

[10] In a maximum consensus algorithm, each link/source starts with some state. In each update, the source sends its current value to the links on its route, and each link keeps the only the maximum value among all the values received (including the previous value it holds), each source then updates by aggregating maximum information along its route. Therefore after one round of algorithm, the sources sharing link(s) with the route that contains the maximum value now has the maximum value. Since the links form a connected graph, after at most S iterations, the entire network reaches a consensus on the maximum state value and the algorithm terminates.

error checking method presented here, in [31] we also proposed a distributed scheme to explicitly compute at each primal iteration a bound on the number of dual steps that can satisfy the error level.

4.5 Stepsize Rule

Based on the preceding two sections, we can compute the inexact Newton direction $\Delta\tilde{x}^k$ in a distributed way, the only other unknown term in the update equation (28) is the stepsize d^k. In this section, we describe a stepsize rule that can achieve local superlinear convergence rate (to an error neighborhood) for the primal iterations (see [30] for more details). Section 4.5.1 presents a distributed procedure to compute the stepsize. Section 4.5.2 shows that with this stepsize rule, the primal vectors x^k generated by the algorithm remain strictly positive for all k, hence ensuring that the Hessian matrix and therefore the (inexact) Newton direction are well-defined at all iterates (see Eq. (11)).

Our stepsize rule will be based on an inexact version of the Newton decrement. At a given primal vector x^k (with Hessian matrix H_k), we define the *exact Newton direction*, denoted by Δx^k, as the exact solution of the system of equations (8). The *exact Newton decrement* $\lambda(x^k)$ is defined as

$$\lambda(x^k) = \sqrt{(\Delta x^k)' H_k \Delta x^k}. \tag{45}$$

Similarly, the *inexact Newton decrement* $\tilde{\lambda}(x^k)$ is given by

$$\tilde{\lambda}(x^k) = \sqrt{(\Delta \tilde{x}^k)' H_k \Delta \tilde{x}^k}, \tag{46}$$

where $\Delta\tilde{x}^k$ is the inexact Newton direction at primal vector x^k. Note that both $\lambda(x^k)$ and $\tilde{\lambda}(x^k)$ are nonnegative and well-defined due to the fact that the matrix $H_k = \nabla^2 f(x^k)$ is positive definite.

Given the scalar $\tilde{\lambda}(x^k)$, at each iteration k, we choose the stepsize d^k as follows: Let V be some positive scalar with $0 < V < 0.267$. We have

$$d^k = \begin{cases} \frac{1}{\tilde{\lambda}(x^k)+1} & \text{if} \quad \tilde{\lambda}(x^k) \geq V \text{ for all previous } k, \\ 1 & \text{otherwise.} \end{cases} \tag{47}$$

The upper bound on V guarantees local quadratic convergence of our algorithm [30]. The fact that $V < 1$ also ensures starting with a strictly positive feasible solution, the primal vectors x^k generated by our algorithm remain strictly positive for all k (see Section 4.5.2).

4.5.1 Distributed Inexact Newton Decrement Computation

We first note that in Eq. (47), the only unknown term is the inexact Newton decrement $\tilde{\lambda}(x^k)$. In order to compute the value of $\tilde{\lambda}(x^k)$, we rewrite the inexact Newton decrement as $\tilde{\lambda}(x^k) = \sqrt{\sum_{i\in\mathcal{S}} (\Delta\tilde{x}_i^k)^2 (H_k)_{ii} + \sum_{l\in\mathcal{L}} (\Delta\tilde{x}_{l+S}^k)^2 (H_k)_{(l+S)(l+S)}}$, or equivalently,

$$\left(\tilde{\lambda}(x^k)\right)^2 = \sum_{i\in\mathcal{S}} (\Delta\tilde{x}_i^k)^2 (H_k)_{ii} + \sum_{l\in\mathcal{L}} (\Delta\tilde{x}_{l+S}^k)^2 (H_k)_{(l+S)(l+S)}. \tag{48}$$

In the sequel of this section, we develop a distributed summation procedure to compute this quantity by aggregating the local information available on sources and links. A key feature of this procedure is that it respects the simple information exchange mechanism used by first order methods applied to the NUM problem: information about the links along the routes is aggregated and sent back to the sources using a feedback mechanism. Over-counting is avoided

using a novel off-line construction, which forms an *(undirected) auxiliary graph* that contains information on sources sharing common links.

Given a network with source set $\mathcal{S} = \{1, 2, \ldots, S\}$ (each associated with a predetermined route) and link set $\mathcal{L} = \{1, 2, \ldots, L\}$, we define the set of nodes in the auxiliary graph as the set $\mathcal{S}$, i.e., each node corresponds to a source (or equivalently, a flow) in the original network. The edges are formed between sources that share common links according to the following iterative construction. In this construction, each source is equipped with a state (or color) and each link is equipped with a set (a subset of sources), which are updated using signals sent by the sources along their routes.

Auxiliary Graph Construction:

- Initialization: Each link l is associated with a set $\Theta_l = \emptyset$. One arbitrarily chosen source is marked as grey, and the rest are marked as white. The grey source sends a signal {label, i} to its route. Each link l receiving the signal, i.e., $l \in L(i)$, adds i to Θ_l.

- Iteration: In each iteration, first the sources update their states and send out signals according to step (A). Each link l then receives signals sent in step (A) from the sources $i \in S(l)$ and updates the set Θ_l according to step (B).

  (A) Each source i:

    (A.a) If it is white, it sums up $|\Theta_l|$ along its route, using the value $|\Theta_l|$ from the previous time.

      (A.a.1) If $\sum_{l \in L(i)} |\Theta_l| > 0$, then the source i is marked grey and it sends two signals {neighbor, i} and {label, i} to its route.

      (A.a.2) Else, i.e., $\sum_{l \in L(i)} |\Theta_l| = 0$, source i does nothing for this iteration.

    (A.a) Otherwise, i.e., it is grey, source i does nothing.

  (B) Each link l:

    (B.a) If $\Theta_l = \emptyset$:

      (B.a.1) If it experiences signal {label, i} passing through it, it adds i to Θ_l. When there are more than one such signals during the same iteration, only the smallest i is added. The signal keeps traversing the rest of its route.

      (B.a.2) Otherwise link l simply carries the signal(s) passing through it, if any, to the next link or node.

    (B.b) Else, i.e., $\Theta_l \neq \emptyset$:

      (B.b.1) If it experiences signal {neighbor, i} passing through it, an edge (i, j) with label L_l is added to the auxiliary graph for all $j \in \Theta_l$, and then i is added to the set Θ_l. If there are more than one such signals during the same iteration, the sources are added sequentially, and the resulting nodes in the set Θ_l form a clique in the auxiliary graph. Link l then stops the signal, i.e., it does not pass the signals to the next link or node.

      (B.b.2) Otherwise link l simply carries the signal(s) passing through it, if any, to the next link or node.

- Termination: Terminate after $S - 1$ number of iterations.

The auxiliary graph construction process for the sample network in Figure 3 is illustrated in Figure 4, where the left column reflects the color of the nodes in the original network and

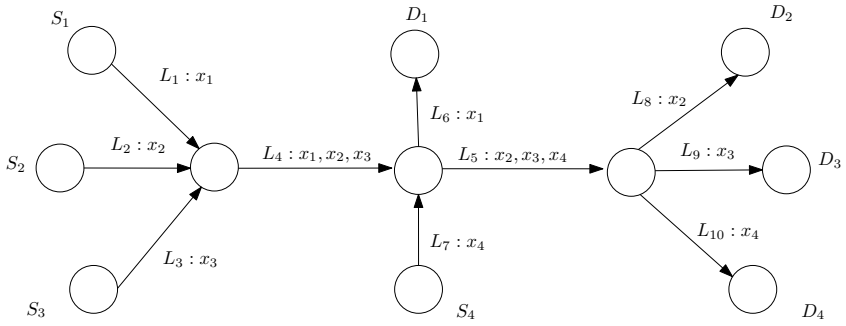


Figure 3: A sample network with four sources and ten links. Each link shows the flows (or sources) using that link. This example will be used to illustrate different parts of the distributed stepsize computation in this section.

the elements of the set Θ_l (labeled on each link l), while the right column corresponds to the auxiliary graph constructed after each iteration.[11]

We next investigate some properties of the auxiliary graph, which will be used in proving that our distributed summation procedure yields the corrects values.

Lemma 4.8. Consider a network and its auxiliary graph with sets $\{\Theta_l\}_{l\in\mathcal{L}}$. The following statements hold:

(1) For each link l, $\Theta_l \subset S(l)$.

(2) Source nodes i, j are connected in the auxiliary graph if and only if there exists a link l, such that $\{i, j\} \subset \Theta_l$.

(3) The auxiliary graph does not contain multiple edges, i.e., there exists at most one edge between any pair of nodes.

(4) The auxiliary graph is connected.

(5) For each link l, $\Theta_l \neq \emptyset$.

(6) There is no simple cycle in the auxiliary graph other than that formed by only the edges with the same label.

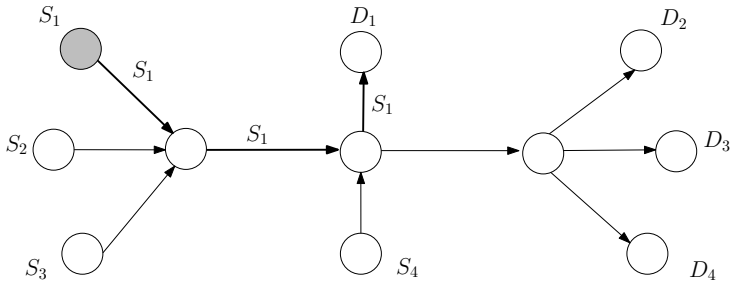
Proof. We prove the above statements in the order they are stated.

(1) Part (1) follows immediately from our auxiliary graph construction, because each source only sends signals to links on its own route and the links only update their set Θ_l when they experience some signals passing through them.

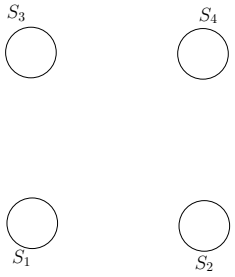
(2) In the auxiliary graph construction, a link is added to the auxiliary graph only in step (B.b.1), where part (2) clearly holds.

(3) From the first two parts, there is an edge between source nodes i, j, i.e., $\{i, j\} \subset \Theta_l$ for some l, only if i and j share link l in the original network. From the auxiliary graph construction, if sources i and j share link l then an edge with label L_l between i and j is formed at some iteration if and only if one of the following three cases holds:

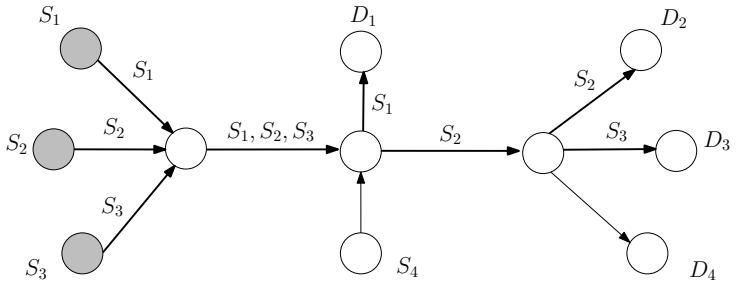
[11]Note that depending on construction, a network may have different auxiliary graphs associated with it. Any of these graphs can be used in the distributed summation procedure.



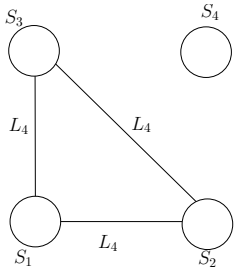
(a) State of the network $t = 0$



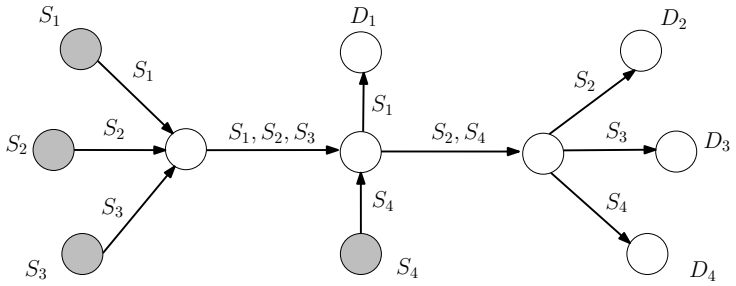
(b) State of the auxiliary graph $t = 0$



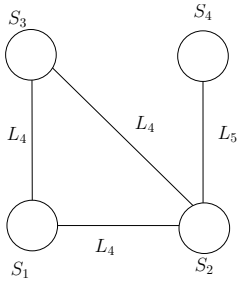
(c) State of the network $t = 1$



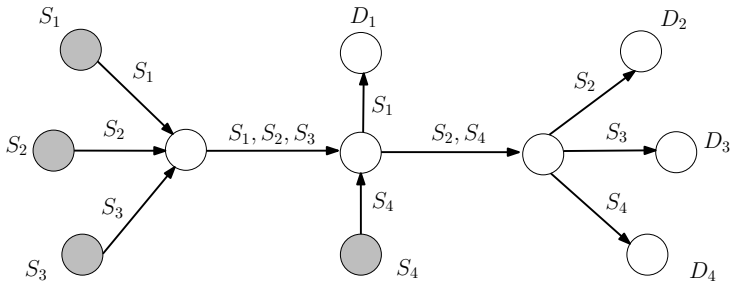
(d) State of the auxiliary graph $t = 1$



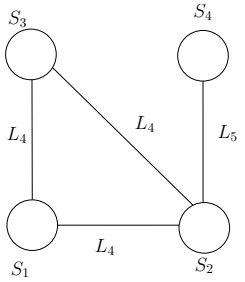
(e) State of the network $t = 2$



(f) State of the auxiliary graph $t = 2$



(g) State of the network $t = 3$



(h) State of the auxiliary graph $t = 3$

Figure 4: Steps of the construction of the auxiliary graph corresponding to the network in Figure 3. The elements of Θ_l are labeled on link l. A link is drawn bold in the original graph if $\Theta_l \neq \emptyset$.

I In the beginning of the previous iteration $\Theta_l = \emptyset$ and sources i, j are both white. During the previous iteration, source i becomes grey and sends out the signal {label,i} to link l, hence $\Theta_l = \{i\}$. In the current iteration, source j with $\sum_{m \in L(j)} |\Theta_m| \geq |\Theta_l| > 0$ becomes grey and sends out signal {neighbor, j} to link l;

II The symmetric case of I, where first source j becomes grey and one iteration later source i becomes grey.

III In the beginning of the previous iteration $\Theta_l = \emptyset$ and sources i, j are both white. During the previous iteration, some other source t with $l \in L(t)$ becomes grey and sends out the signal {label,t} to link l, hence $\Theta_l = \{t\}$. In the current iteration, both source i and j with $\sum_{m \in L(i)} |\Theta_m| \geq |\Theta_l| > 0$ and $\sum_{m \in L(j)} |\Theta_m| \geq |\Theta_l| > 0$ become grey and send out signals {neighbor, i} and {neighbor, j} to link l.

Hence if an edge connecting nodes i and j exists in the auxiliary graph, then in the beginning of the iteration when the edge is formed at least one of the nodes is white, and by the end of the iteration both nodes are colored grey and stay grey. Therefore the edges between i and j in the auxiliary graph can only be formed during exactly one iteration.

We next show that only one such edge can be formed in one iteration. The first two cases are symmetric, and without loss of generality we only consider cases I and III. In both of these cases, an edge between i and j is formed with label L_l only if link l receives the signal {neighbor, j} and $\Theta_l \neq \emptyset$. In step (B.b.1) of the auxiliary graph construction, the first link with $\Theta_l \neq \emptyset$ stops the signal from passing to the rest of its route, hence at most one edge between i and j can be generated. Hence part (3) holds.

(4) By using a similar analysis as above, it is straightforward to see that if at one iteration source i from the original network becomes grey, then in the next iteration all the sources which share link with i become grey and are connected to i in the auxiliary graph. By induction, we conclude that all the nodes in the auxiliary graph corresponding to sources colored grey in the original network are connected to the source node marked grey in the initialization step, and hence these nodes form a connected component.

We next show that all nodes are colored grey when the auxiliary graph construction procedure terminates. We first argue that at least one node is marked grey from white at each iteration before all nodes are marked grey. Assume the contrary is true, that is at some iteration no more nodes are marked grey and there exists a set of white nodes S^*. This implies that the nodes in S^* do not share any links with the nodes in $\mathcal{S} \backslash S^*$ and thus there is no path from any source in the set $\mathcal{S} \backslash S^*$ to any source in S^* using the links (including the feedback mechanisms) in the original network. However, this contradicts the fact that all links form a strongly connected graph. Therefore after $S - 1$ iterations all nodes in the original graph are colored grey and therefore we have the desired statement hold.

(5) Analysis for part (3) suggests that all the connected nodes in the auxiliary graph are colored grey. In view of the part (4), all the sources are colored grey when the auxiliary graph construction procedure terminates. Step (B.a.1) implies that a link has $\Theta_l = \emptyset$ if all sources $i \in S(l)$ are white. Since each link is used by at least one source, and all sources are grey, part (5) holds.

(6) We prove part (6) by showing the auxiliary graph, when the cycles formed by the edges of the same label are removed, is acyclic. For each link l, let i_l^* denote the first element added to the set Θ_l in the auxiliary graph construction process, which is uniquely defined for each link l by Step (B.a.1). In the set $\mathcal{S}$ for each link l, we define an equivalence class

by $i \sim j$ if $\{i,j\} \subset \Theta_l \backslash \{i_l^*\}$, which implies if and only if i and j are connected in the auxiliary graph and $i \sim j$, this link is formed by scenario III as defined above in the proof of part (3).

The nodes in each equivalence class are connected by edges with the same label, which form the undesired cycles. We remove these cycles by merging each equivalence class into one representative node, which inherits all the edges going between the nodes in the equivalence class and $\mathcal{S}\backslash\Theta_l$ in the auxiliary graph, and is connected to i_l^* via one edge. Note the resulting graph is connected, since the auxiliary graph is by part (4) and all the remaining edges are generated under scenarios I and II as defined in the proof of part (3).

We now show that the resulting graph contains no cycle. From cases I and II, it follows immediately that an edge is generated when one more source becomes grey. Therefore if number of noes is N, we have $N-1$ edges. In a connected graph, this implies we have a tree, i.e. acyclic, and hence part (6) holds.

□

We denote the set of links inducing edges in the auxiliary graph as $L^* = \{l \mid |\Theta_l| > 1\}$ and for each source i the set of links which induce edges in the auxiliary graph as $L^*(i) = \{l \mid i \in \Theta_l, l \in L^*\} \subset L(i)$ for notational convenience. Each link can identify if it is in L^* by the cardinality of the set Θ_l. Each source i can obtain $|L^*(i)|$ along the links on its route. The auxiliary graph remains the same throughout the distributed inexact Newton algorithm and only depends on the structure of the network (independent of the utility functions and link capacities), therefore given a network, the above construction only needs to be preformed once prior to execution of the distributed Newton algorithm.

We next present a distributed procedure to compute the sum in Eq. (48) and show that the sets Θ_l constructed using the above procedure avoids over-counting and enables computation of the correct values.[12]

Distributed Summation Procedure:

- Initialization: Each link l initializes to $z_l(0) = 0$. Each source i computes $y_i^* = (\Delta\tilde{x}_i^k)^2 (H_k)_{ii}$ and each link l computes $z_l^* = \frac{1}{|S(l)|}(\Delta\tilde{x}_{l+S}^k)^2 (H_k)_{(l+S)(l+S)}$. Each source i aggregates the sum
$$y_i(0) = y_i^* + \sum_{l \in L(i)} z_l^* \tag{49}$$
along its route.

- Iteration for $t = 1, 2, \ldots, S$. The following 3 steps are completed in the order they are presented.

  a. Each source i sends its current value $y_i(t)$ to its route.

  b. Each link l uses the $y_i(t)$ received and computes
  $$z_l(t) = \sum_{i \in \Theta_l} y_i(t-1) - (|\Theta_l| - 1)\, z_l(t-1). \tag{50}$$

  c. Each source i aggregates information along its route from the links $l \in L^*(i)$ and computes
  $$y_i(t) = \sum_{l \in L^*(i)} z_l(t) - (|L^*(i)| - 1)\, y_i(t-1). \tag{51}$$

[12]Note that the execution of the procedure only uses the sets Θ_l, L^*, and $L^*(i)$. We will use the structure of the auxiliary graph in proving the correctness of the procedure.

- Termination: Terminate after S number of iterations.

By the diagonal structure of the Hessian matrix H_k, the scalars $(\Delta\tilde{x}_i^k)^2(H_k)_{ii}$ and $(\Delta\tilde{x}_{l+S}^k)^2(H_k)_{(l+S)(l+S)}$ are available to the corresponding source i and link l respectively, hence z_l^* and y_i^* can be computed using local information. In the above process, each source only uses aggregate information along its route $L^*(i) \subset L(i)$ and each link l only uses information from sources $i \in \Theta_l \subset S(l)$. The evolution of the distributed summation procedure for the sample network in Figure 3 is shown in Figures 5 and 6.

We next establish two lemmas, which quantifies the expansion of the t-hop neighborhood in the auxiliary graph for the links and sources. This will be key in showing that the aforementioned summation procedure yields the correct values at the sources and the links. For each source i, we use the notation $\mathcal{N}_i(t)$ to denote the set of nodes that are connected to node i by a path of length at most t in the auxiliary graph. Note that $\mathcal{N}_i(0) = \{i\}$. We say that node i is *t-hops away* from node j is the length of the shortest path between nodes i and j is t.

Lemma 4.9. Consider a network and its auxiliary graph with sets $\{\Theta_l\}_{l\in\mathcal{L}}$. For any link l and all $t \geq 1$, we have,

$$\mathcal{N}_i(t) \cap \mathcal{N}_j(t) = \cup_{m\in\Theta_l}\mathcal{N}_m(t-1) \qquad \text{for } i,j \in \Theta_l \text{ with } i \neq j. \tag{52}$$

Proof. Since the source nodes $i, j \in \Theta_l$, by part (2) of Lemma 4.8, they are 1-hop away from all other nodes in Θ_l. Hence if a source node n is in $\mathcal{N}_m(t-1)$ for $m \in \Theta_l$, then n is at most t-hops away from i or j. This yields

$$\cup_{m\in\Theta_l}\mathcal{N}_m(t-1) \subset \mathcal{N}_i(t) \cap \mathcal{N}_j(t). \tag{53}$$

On the other hand, if $n \in \mathcal{N}_i(t) \cap \mathcal{N}_j(t)$, then we have either $n \in \mathcal{N}_i(t-1)$ and hence $n \in \cup_{m\in\Theta_l}\mathcal{N}_m(t-1)$ or

$$n \in (\mathcal{N}_i(t) \backslash \mathcal{N}_i(t-1)) \cap \mathcal{N}_j(t).$$

Let $P(a,b)$ denote an ordered set of nodes on the path between nodes a and b including b but not a for notational convenience. Then the above relation implies there exists a path with $|P(i,n)| = t$ and $|P(j,n)| \leq t$. Let $n^* \in P(i,n) \cap P(j,n)$ and $P(j,n^*) \cap P(j,n^*) = \emptyset$. The node n^* exists, because the two paths both end at n. If $n^* \notin \Theta_l$, then we have a cycle of $\{P(i,n^*), P(n^*,j), P(j,i)\}$, which includes an edge with label L_l between i and j and other edges. In view of part (6) of Lemma 4.8, this leads to a contradiction. Therefore we obtain $n^* \in \Theta_l$, implying $P(i,n) = \{P(i,n^*), P(n^*,n)\}$. Since i is connected to all nodes in Θ_l, $|P(i,n^*)| = 1$ and hence $|P(n^*,n)| = t-1$, which implies $n \in \mathcal{N}_{n^*}(t-1) \subset \cup_{m\in\Theta_l}\mathcal{N}_m(t-1)$. Therefore the above analysis yields

$$\mathcal{N}_i(t) \cap \mathcal{N}_j(t) \subset \cup_{m\in\Theta_l}\mathcal{N}_m(t-1).$$

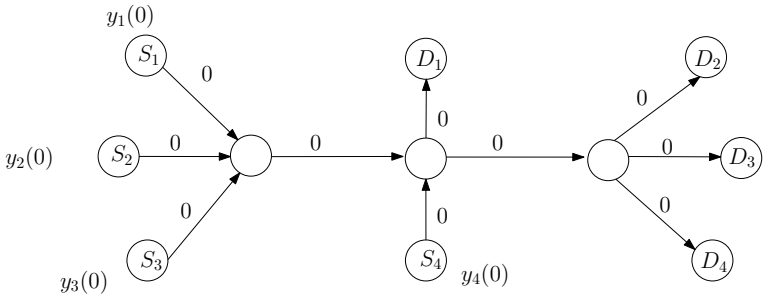
With relation (53), this establishes the desired equality. □

Lemma 4.10. Consider a network and its auxiliary graph with sets $\{\Theta_l\}_{l\in\mathcal{L}}$. For any source i, and all $t \geq 1$, we have,

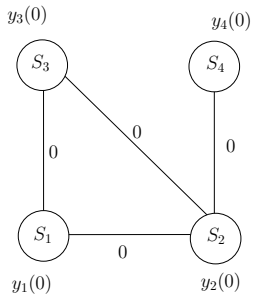
$$(\cup_{j\in\Theta_l}\mathcal{N}_j(t)) \cap (\cup_{j\in\Theta_m}\mathcal{N}_j(t)) = \mathcal{N}_i(t) \qquad \text{for } l,m \in L^*(i) \text{ with } l \neq m. \tag{54}$$

Proof. Since $l, m \in L^*(i)$, we have $i \in \Theta_l$ and $i \in \Theta_m$, this yields,

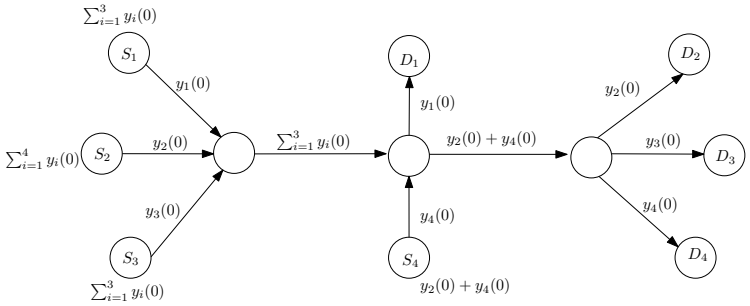
$$\mathcal{N}_i(t) \subset (\cup_{j\in\Theta_l}\mathcal{N}_j(t)) \cap (\cup_{j\in\Theta_m}\mathcal{N}_j(t)).$$



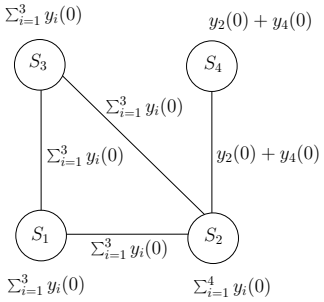
(a) State of the network $t = 0$



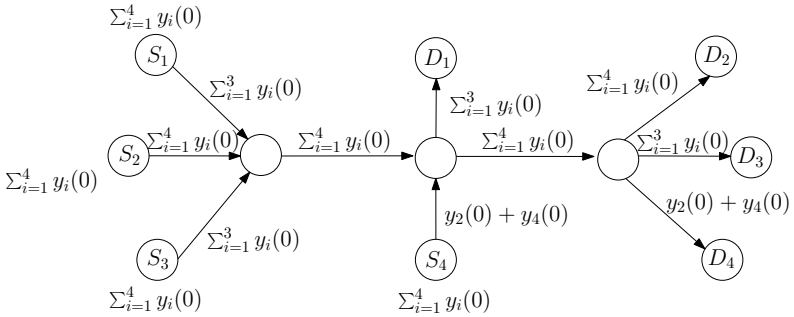
(b) State of auxiliary graph $t = 0$



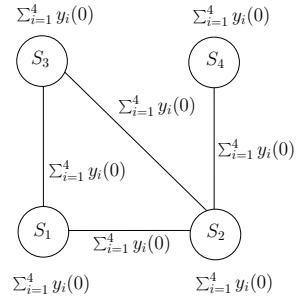
(c) State of the network $t = 1$



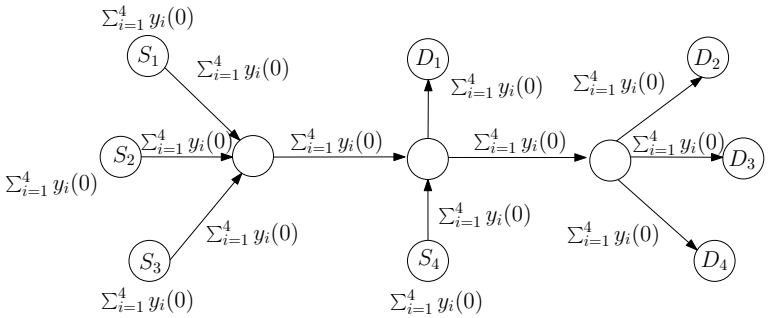
(d) State of auxiliary graph $t = 1$



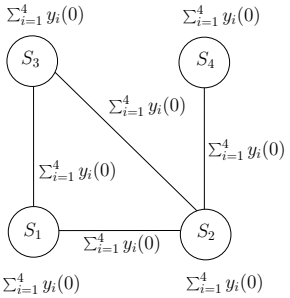
(e) State of the network $t = 2$



(f) State of auxiliary graph $t = 2$

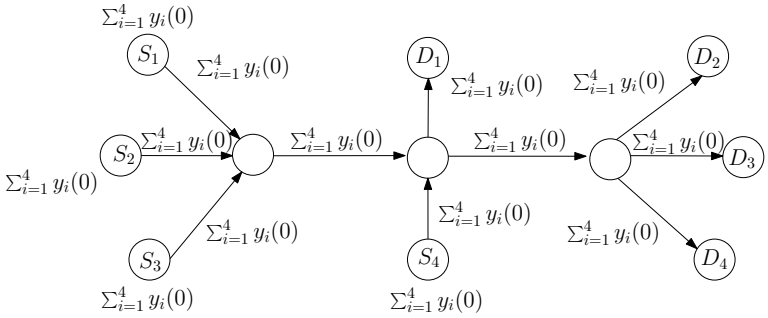


(g) State of the network $t = 3$

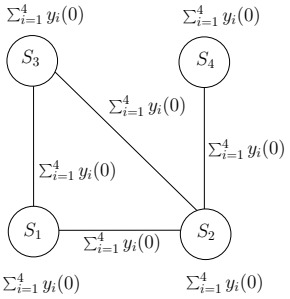


(h) State of auxiliary graph $t = 3$

Figure 5: Evolution of distributed summation process, where $\rho_i = y_i(0)$ and destination node is indicated using a dot with the same color as its corresponding source.



(a) State of the network $t = 4$



(b) State of auxiliary graph $t = 4$

Figure 6: Evolution of distributed summation process continued, where $\rho_i = y_i(0)$ and destination node is indicated using a dot with the same color as its corresponding source.

On the other hand, assume there exists a node n with $n \in (\cup_{j\in\Theta_l}\mathcal{N}_j(t)) \cap (\cup_{j\in\Theta_m}\mathcal{N}_j(t))$, and $n \notin \mathcal{N}_i(t)$. Then there exists a node $p \in \Theta_l$ with $p \neq i$ and $n \in \mathcal{N}_p(t)$. Similarly there exists a node $q \in \Theta_m$ with $q \neq i$ and $n \in \mathcal{N}_q(t)$. Let $P(a, b)$ denote an ordered set nodes on the path between nodes a and b including b but not a for notational convenience. Let $n^* \in P(p, n) \cap P(q, n)$ and $P(p, n^*) \cap P(q, n^*) = \emptyset$. The node n^* exists, because the two paths both end at n. Since nodes i, p are connected via an edge with label L_l and i, q are connected via an edge with label L_m, we have a cycle of $\{P(i, p), P(p, n), P(n, q), P(q, i)\}$, which contradicts part (6) in Lemma 4.8 and we have

$$(\cup_{j\in\Theta_l}\mathcal{N}_j(t)) \cap (\cup_{j\in\Theta_m}\mathcal{N}_j(t)) \subset \mathcal{N}_i(t).$$

The preceding two relations establish the desired equivalence. □

Equipped with the preceding lemma, we can now show that upon termination of the summation procedure, each source i and link l have $y_i(S) = z_l(S-1) = (\tilde{\lambda}(x^k))^2$ [cf. Eq. (48)].

Theorem 4.11. Consider a network and its auxiliary graph with sets $\{\Theta_l\}_{l\in\mathcal{L}}$. Let Ω denote the set of all subsets of S and define the function $\sigma : \Omega \to \mathbb{R}$ as

$$\sigma(K) = \sum_{l\in\cup_{i\in K}L(i)} z_l^* \sum_{i\in K} I_{\{l\in L(i)\}} + \sum_{i\in K} y_i^*,$$

where $y_i^* = (\Delta\tilde{x}_i^k)^2(H_k)_{ii}$, $z_l^* = \frac{1}{|S(l)|}(\Delta\tilde{x}_{l+S}^k)^2(H_k)_{(l+S)(l+S)}$ and $I_{\{l\in L(i)\}}$ is the indicator function for the event $\{l \in L(i)\}$. Let $y_i(t)$ and $z_l(t)$ be the iterates generated by the distributed summation procedure described above. Then for all $t \in \{1, \ldots, S\}$, the value $z_l(t)$ at each link satisfies

$$z_l(t) = \sigma(\cup_{i\in\Theta_l}\mathcal{N}_i(t-1)), \tag{55}$$

and the value $y_i(t)$ at each source node satisfies

$$y_i(t) = \sigma(\mathcal{N}_i(t)). \tag{56}$$

Proof. We use induction to prove the theorem.
Base case: $t = 1$**.**
Since $z_l(0) = 0$ for all links, Eq. (50) for $t = 1$ is

$$z_l(1) = \sum_{i\in\Theta_l} y_i(0) = \sum_{i\in\Theta_l}(y_i^* + \sum_{l\in L(i)} z_l^*) = \sigma(\Theta_l),$$

where we use the definition of $y(0)$ [cf. Eq. (49)] and the function $\sigma(\cdot)$. Since $\mathcal{N}_i(0) = i$, the above relation implies Eq. (55) holds.

For source i, from update relation (51), we have

$$y_i(1) = \sum_{l \in L^*(i)} \sigma(\Theta_l) - (|L^*(i)| - 1)\, y_i(0).$$

Lemma 4.10 and inclusion-exclusion principle imply

$$\sum_{l \in L^*(i)} \sigma(\Theta_l) = \sigma(\cup_{l \in L^*(i)} \Theta_l) + (|L^*(i)| - 1)\, \sigma(i).$$

Since $y_i(0) = \sigma(i)$ based on the definition of $y_i(0)$ [cf. Eq. (49)], by rearranging the preceding two relations, we obtain

$$y_i(1) = \sigma(\cup_{l \in L^*(i)} \Theta_l) = \sigma(\mathcal{N}_i(1)),$$

which shows Eq. (56) holds for $t = 1$.

Inductive step for $t = T \geq 2$.

Assume for $t = T - 1$, Eqs. (56) and (55) hold, we first show that Eq. (55) hold. When $t = T$, by update equation (50), we obtain for link l

$$\begin{aligned} z_l(T) &= \sum_{i \in \Theta_l} y_i(T-1) - (|\Theta_l| - 1)\, z_l(T-1) \\ &= \sum_{i \in \Theta_l} \sigma(\mathcal{N}_i(T-1)) - (|\Theta_l| - 1)\, z_l(T-1), \end{aligned}$$

where the second equality follows from Eq. (56) for $t = T - 1$.

If $|\Theta_l| = 1$, then we have $z_l(T) = \sigma(\mathcal{N}_i(T-1))$, for $i \in \Theta_l$, therefore Eq. (55) is satisfied.

For $|\Theta_l| > 1$, using Lemma 4.9 for $t = T$ and by inclusion-exclusion principle, we obtain

$$\sum_{i \in \Theta_l} \sigma(\mathcal{N}_i(T-1)) = \sigma\left(\cup_{i \in \Theta_l} \mathcal{N}_i(T-1)\right) + (|\Theta_l| - 1)\, \sigma(\cup_{m \in \Theta_l} \mathcal{N}_m(T-2)).$$

Eq. (55) for $t = T-1$ yields $z_l(T-1) = \sigma(\cup_{m \in \Theta_l} \mathcal{N}_m(T-2))$. By using this fact and rearranging the preceding two relations, we have Eq. (55) holds for $t = T$, i.e.,

$$z_l(T) = \sigma\left(\cup_{i \in \Theta_l} \mathcal{N}_i(T-1)\right).$$

We next establish Eq. (56). From update equation (51), using the preceding relation, we have

$$\begin{aligned} y_i(T) &= \sum_{l \in L^*(i)} z_l(T) - (|L^*(i)| - 1)\, y_i(T-1) \\ &= \sum_{l \in L^*(i)} \sigma\left(\cup_{i \in \Theta_l} \mathcal{N}_i(T-1)\right) - (|L^*(i)| - 1)\, y_i(T-1). \end{aligned}$$

Lemma 4.10 and inclusion-exclusion principle imply

$$\sum_{l \in L^*(i)} \sigma\left(\cup_{i \in \Theta_l} \mathcal{N}_i(T-1)\right) = \sigma(\cup_{l \in L^*(i)} \cup_{i \in \Theta_l} \mathcal{N}_i(T-1)) + (|L^*(i)| - 1)\, \sigma(\mathcal{N}_i(T-1)).$$

By definition of $\mathcal{N}_i(\cdot)$, we have $\cup_{l \in L^*(i)} \cup_{i \in \Theta_l} \mathcal{N}_i(T-1) = \mathcal{N}_i(T)$. By using Eq. (56) for $t = T-1$, i.e., $y_i(T-1) = \sigma(\mathcal{N}_i(T-1))$ and rearranging the above two equations, we obtain

$$y(T) = \sigma(\mathcal{N}_i(T)),$$

which completes the inductive step. □

Using definition of the function $\sigma(\cdot)$, we have $(\tilde{\lambda}(x^k))^2 = \sigma(\mathcal{S})$. By the above theorem, we conclude that after S iterations,

$$y_i(S) = \sigma(\mathcal{N}_i(S)) = \sigma(\mathcal{S}) = (\tilde{\lambda}(x^k))^2.$$

By observing that $\cup_{i\in\Theta_l}\mathcal{N}_i(S-1) = \mathcal{S}$, we also have

$$z_i(S-1) = \sigma\left(\cup_{i\in\Theta_l}\mathcal{N}_i(S-1)\right) = \sigma(\mathcal{S})) = (\tilde{\lambda}(x^k))^2,$$

where we used part (5) of Lemma 4.8. This shows that the value $\tilde{\lambda}(x^k)^2$ is available to all sources and links after $S-1$ iterations.

Note that the number S is an upper bound on the number of iterations required in the distributed summation process to obtain the correct value at the links and sources in the original graph. If the value of the diameter of the auxiliary graph (or an upper bound on it) is known, then the process would terminate in number of steps equal to this value plus 1. For instance, when all the sources share one common link, then the auxiliary graph is a complete graph, and only 2 iterations is required. On the other hand, when the auxiliary graph is a line, the summation procedure would take S iterations.

We finally contrast our distributed summation procedure with spanning tree computations, which were used widely in 1970s and 1980s for performing information exchange among different processors in network flow problems. In spanning tree based approaches, information from all processors is passed along the edges of a spanning tree, and stored at and broadcast by a designated central root node (see [18] and [8]). In contrast, our summation procedure involves (scalar) information aggregated along the routes and fed back independently to different sources, which is a more natural exchange mechanism in an environment with decentralized sources. Moreover, processors in the system (i.e., sources and links) do not need to maintain predecessor/successor information (as required by spanning tree methods). The only network-related information is the sets θ_l for $l \in \mathcal{L}$ kept at the individual links and obtained from the auxiliary graph, which is itself constructed using the feedback mechanism described above.

4.5.2 Strict Feasibility of Primal Iterates

We next establish that starting with a strictly positive primal vector x^0, by using the stepsize given in (47), all the primal iterates generated by the inexact Newton algorithm remain strictly positivity. This is necessary to ensure that the algorithm is well defined. By using the slack variable definitions, this also implies the source rates generated by the inexact Newton algorithm (i.e., the first S components of x^k) do not violate capacity constraints throughout the algorithm.

Theorem 4.12. Given a strictly positive feasible primal vector x^0, let $\{x^k\}$ be the sequence generated by the inexact distributed Newton method (28) for problem (4) with $\mu \geq 1$. Assume that the stepsize d^k is selected according to Eq. (47) for some positive scalar V with $0 < V < 0.267$. Then, the primal vector x^k is strictly positive, i.e., $x^k > 0$, for all k.

Proof. We will prove this claim by induction. The base case of $x^0 > 0$ holds by the assumption of the theorem. Since the U_i are strictly concave [cf. Assumption 1], for any x^k, we have $-\frac{\partial^2 U_i}{\partial x_i^2}(x_i^k) \geq 0$. Given the form of the Hessian matrix [cf. Eq. (11)], this implies $(H_k)_{ii} \geq \frac{\mu}{(x_i^k)^2}$ for all i, and therefore

$$\tilde{\lambda}(x^k) = \left(\sum_{i=1}^{S+L}(\Delta\tilde{x}_i^k)^2(H_k)_{ii}\right)^{\frac{1}{2}} \geq \left(\sum_{i=1}^{S+L}\mu\left(\frac{\Delta\tilde{x}_i^k}{x_i^k}\right)^2\right)^{\frac{1}{2}} \geq \max_i\left|\frac{\sqrt{\mu}\Delta\tilde{x}_i^k}{x_i^k}\right|,$$

where the last inequality follows from the nonnegativity of the terms $\mu\left(\frac{\Delta\tilde{x}_i^k}{x_i^k}\right)^2$. By taking the reciprocal on both sides, the above relation implies

$$\frac{1}{\tilde{\lambda}(x^k)} \leq \frac{1}{\max_i \left|\frac{\sqrt{\mu}\Delta\tilde{x}_i^k}{x_i^k}\right|} = \frac{1}{\sqrt{\mu}}\min_i \left|\frac{x_i^k}{\Delta\tilde{x}_i^k}\right| \leq \min_i \left|\frac{x_i^k}{\Delta\tilde{x}_i^k}\right|, \tag{57}$$

where the last inequality follows from the fact that $\mu \geq 1$.

We show the inductive step by considering two cases.

- Case i: $\tilde{\lambda}(x^k) \geq V$
  The stepsize choice d^k [cf. Eq. (47)] satisfies

  $$d^k = 1/(1+\tilde{\lambda}(x^k)) < 1/\tilde{\lambda}(x^k).$$

  Using Eq. (57), this implies $d^k < \min_i \left|\frac{x_i^k}{\Delta\tilde{x}_i^k}\right|$. Hence if $x^k > 0$, then $x^{k+1} = x^k + d^k\Delta\tilde{x}^k > 0$.

- Case ii: $\tilde{\lambda}(x^k) < V$
  Since $0 < V < 0.267$, we have $\tilde{\lambda}(x^k) < V \leq 1$. Hence we obtain

  $$d^k = 1 < \frac{1}{\tilde{\lambda}(x^k)} \leq \min_i \left|\frac{x_i^k}{\Delta\tilde{x}_i^k}\right|,$$

  where the last inequality follows from Eq. (57). Once again, if $x^k > 0$, then $x^{k+1} = x^k + d^k\Delta\tilde{x}^k > 0$.

In both cases we have $x^{k+1} = x^k + d^k\Delta\tilde{x}^k > 0$, which completes the induction proof. □

Hence the distributed inexact Newton update is well defined throughout the algorithm and as we show in [30], the algorithm achieves superlinear convergence rate in terms of primal iterations to an error neighborhood with sufficiently small error parameters p and ϵ.

5 Simulation Results

Our simulation results demonstrate that the decentralized Newton method significantly outperforms the existing methods in terms of number of iterations. For our distributed inexact Newton method, we used the following error tolerance levels: $p = 10^{-3}$, $\epsilon = 10^{-4}$ [cf. Assumption 2], and when $\tilde{\lambda}(x^{\bar{k}}) < V = 0.12$ we switch stepsize choice to be $d^k = 1$ for all $k \geq \bar{k}$. With these error tolerance levels, all the conditions necessary for achieving local quadratic rate of convergence can be satisfied. We implemented the distributed inexact Newton method twice with different barrier coefficients; it is shown in [30] that a proper selection of the barrier coefficient (together with a scaling of objective function) guarantees the resulting objective function value to be within 1% of the optimal value of problem (2). For a comprehensive comparison, we count both the primal and dual iterations implemented through distributed error checking method described in 4.4.[13] In particular, in what follows, the number of iterations of our method refers to the sum of dual iterations at each of the generated primal iterate. In the simulation results, we compare our distributed inexact Newton method performance against both the subgradient method used in [19] and the Newton-type diagonal scaling dual method developed in [1]. Both of these methods were implemented using a constant stepsize that can guarantee convergence as shown in [19] and [1].

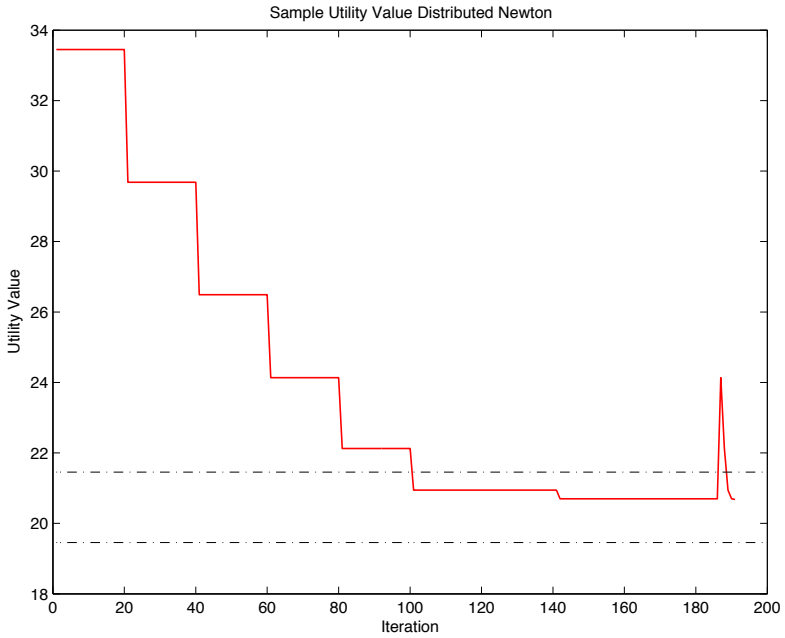


Figure 7: One sample objective function value of distributed inexact Newton method against number of iterations. The dotted black lines denote $\pm 5\%$ interval of the optimal objective function value.

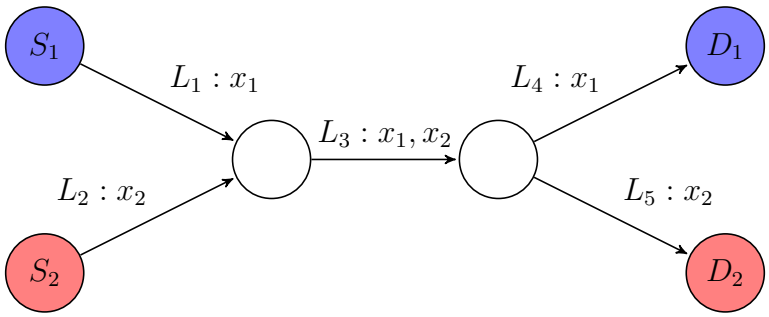


Figure 8: A sample network. Each source-destination pair is displayed with the same color. We use x_i to denote the flow corresponding to the i^{th} source-destination pair and L_i to denote the i^{th} link.

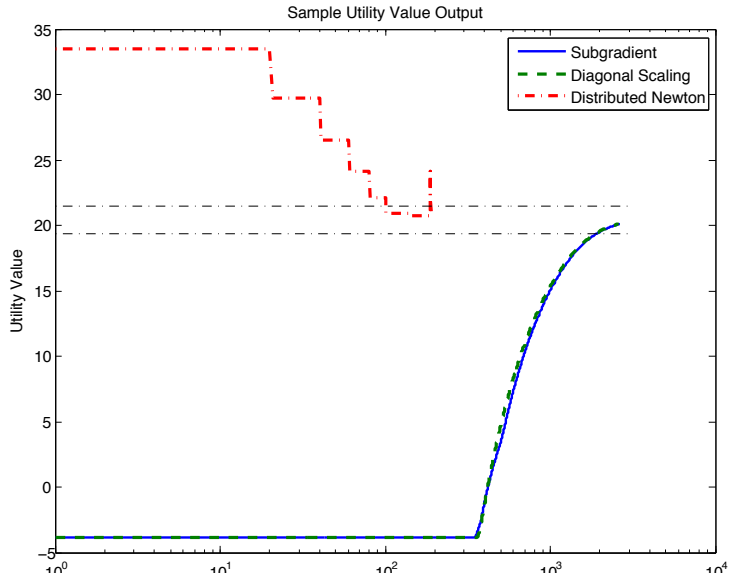


Figure 9: One sample objective function value of all three methods against log scaled iteration count. The dotted black lines denote ±5% interval of the optimal objective function value.

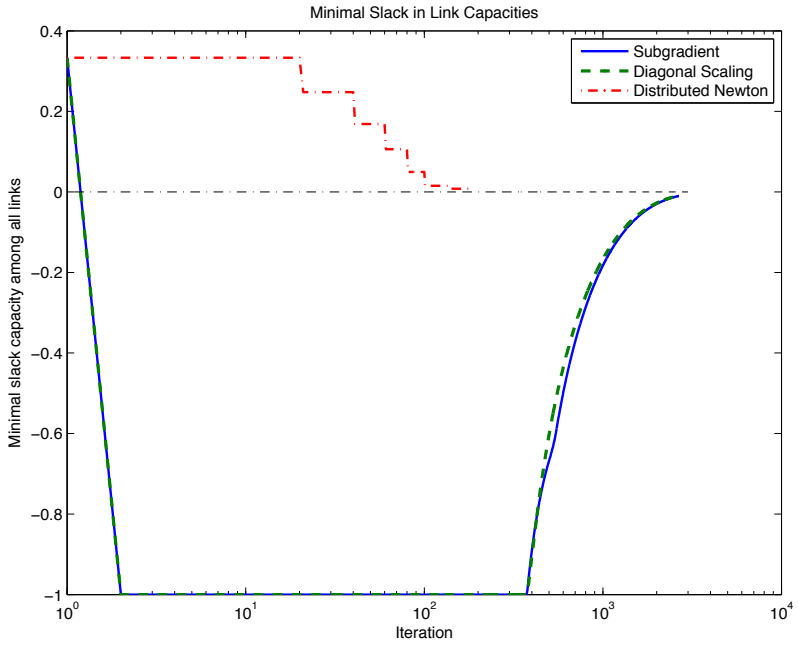


Figure 10: Sample minimal slack in link capacity of all three methods against log scaled iteration count. Negative slack means violating capacity constraint. The dotted black line denotes 0.

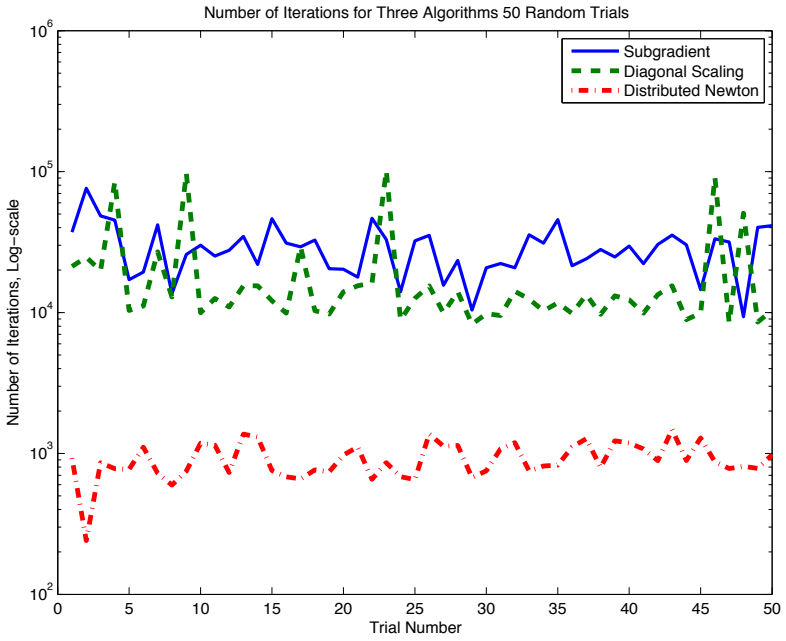


Figure 11: Log scaled iteration count for the 3 methods implemented over 50 randomly generated networks.

Figure 7 presents a sample evolution of the objective function value when the distributed inexact Newton method is applied to a simple network shown in Figure 8. The horizontal line segments correspond to the dual iterations, where the primal vector stays constant, and each jump in the figure is a primal Newton update. The spike close to the end is a result of rescaling and using a new barrier coefficient in the second round of the distributed inexact Newton algorithm (see [30] for more details regarding implementing two inexact Newton algorithms to obtain a good approximation of the optimal function value). The black dotted lines indicate $\pm 5\%$ interval around the optimal objective function value.

The other two algorithms were implemented for the same problem, and the objective function values are plotted in Figure 9, with logarithmic scaled iteration count on the x-axis. We use black dotted lines to indicate $\pm 5\%$ interval around the optimal objective function value. While the subgradient and diagonal scaling methods have similar convergence behavior, the distributed inexact Newton method significantly outperforms the two.

One of the important features of the distributed inexact Newton method is that, unlike the other two algorithms, the generated primal iterates satisfy the link capacity constraint throughout the algorithm. This observation is confirmed by Figure 10, where the minimal slacks in links are shown for all three algorithms. The black dotted line is the zero line and a negative slack means violating the capacity constraint. The slacks that our distributed inexact Newton method yields always stays above the zero line, while the other two only becomes feasible in the end.

To test the performances of the methods over general networks, we generated 50 random networks, with number of links $L = 15$ and number of sources $S = 8$. Each routing matrix

[13]In these simulations we did not include the number of steps required to compute the stepsize (distributed summation with finite termination) and to implement distributed error checking (maximum consensus) to allow the possibilities that other methods can be used to compute these. Note that the number of iterations required by both of these computation is upper bounded by the number of sources, which is a small constant (5 and 8 for example) in our simulations.

consists of $L \times R$ Bernoulli random variables.[14] All three methods are implemented over the 50 networks. We record the number of iterations upon termination for all 3 methods, and results are shown in Figure 11 on a log scale. The mean number of iterations to convergence from the 50 trials is 924 for distributed inexact Newton method, 20286 for Newton-type diagonal scaling and 29315 for subgradient method.

6 Conclusions

This paper develops a distributed inexact Newton-type second order algorithm for Network Utility Maximization problems, which can achieve superlinear convergence rate (in primal iterates) to some error neighborhood. We show that the computation of the dual variables can be implemented in a decentralized manner using a matrix splitting scheme. We also presented a stepsize rule based on the inexact Newton decrement and a distributed scheme for its computation. The implementation of both the matrix splitting and stepsize computation schemes use an information exchange mechanism similar to that involved in first order methods applied to this problem. Simulation results also indicate significant improvement over traditional distributed algorithms for NUM problems. Possible future directions include investigation of the network effects on the speed of convergence and analysis of the convergence properties for a fixed finite truncation of dual iterations.

References

[1] S. Athuraliya and S. Low. Optimization flow control with Newton-like algorithm. *Journal of Telecommunication Systems*, 15:345–358, 2000.

[2] A. Berman and R. J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. Academic Press, New York, 1979.

[3] D. P. Bertsekas and E. M. Gafni. Projected Newton Methods and Optimization of Multicommodity Flows. *IEEE Transactions on Automatic Control*, 28(12), 1983.

[4] D. P. Bertsekas, A. Nedic, and A. Ozdaglar. *Convex Analysis and Optimization*. Athena Scientific, Cambridge, MA, 2003.

[5] N. Biggs. *Algebraic Graph Theory*. Cambridge University Press, second edition, 1993.

[6] V. D. Blondel, J. M. Hendrickx, A. Olshevsky, and J. N. Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. *Proceedings of IEEE CDC*, 2005.

[7] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[8] G. H. Bradley, G. G. Brown, and G. W. Graves. Design and implementation of large scale primal transshipment algorithms. *Management Science*, 24(1):1–34, 1977.

[9] M. Chiang, S. H. Low, A. R. Calderbank, and J.C. Doyle. Layering as optimization decomposition: a mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, 2007.

[14] When there exists a source that does not use any links or a link that is not used by any sources, we discard the routing matrix and generate another one.

[10] F. R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics)*. No. 92, American Mathematical Society, 1997.

[11] R. Cottle, J. Pang, and R. Stone. *The Linear Complementarity Problem*. Academic Press, 1992.

[12] R. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, New York, 1985.

[13] A. Jadbabaie, J. Lin, and S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[14] A. Jadbabaie, A. Ozdaglar, and M. Zargham. A Distributed Newton method for network optimization. *Proc. of CDC*, 2009.

[15] F. Jarre. Interior-point methods for convex programming. *Applied Mathematics and Optimization*, 26:287–311, 1992.

[16] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[17] F. P. Kelly, A. K. Maulloo, and D. K. Tan. Rate control for communication networks: shadow prices, proportional fairness, and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[18] J. G. Klincewicz. A Newton Method for Convex Separable Network Flow Problems. *Networks*, 13:427–442, 1983.

[19] S. H. Low and D. E. Lapsley. Optimization flow control, I: basic algorithm and convergence. *IEEE/ACM Transaction on Networking*, 7(6):861–874, 1999.

[20] A. Nedic and A. Ozdaglar. *Convex Optimization in Signal Processing and Communications*, chapter Cooperative distributed multi-agent optimization. Eds., Eldar, Y. and Palomar, D., Cambridge University Press, 2008.

[21] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48 –61, Jan 2009.

[22] A. Nedic and A. Ozdaglar. Convergence rate for consensus with delays. *Journal of Global Optimization*, 47(3):437–456, 2010.

[23] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, 2001.

[24] R. Olfati-Saber and R. M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, 2004.

[25] A. Olshevsky and J. N. Tsitsiklis. Convergence speed in distributed consensus and averaging. *SIAM Journal on Control and Optimization*, 48(1):33–35, 2009.

[26] R. Srikant. *The Mathematics of Internet Congestion Control (Systems and Control: Foundations and Applications)*. Birkhäuser Boston, 2004.

[27] J. N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation*. PhD thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1984.

[28] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9):803–812, 1986.

[29] R. Varga. *Gershgorin and His Circles*. Springer Series in Computational Mathematics, 2004.

[30] E. Wei, A. Ozdaglar, and A. Jadbabaie. A distributed Newton method for Network Utility Maximization II, Convergence. *LIDS Report 2870*, 2011.

[31] E. Wei, M. Zargham, A. Ozdaglar, and A. Jadbabaie. On dual convergence of the distributed Newton method for Network Utility Maximization. *LIDS Report 2868*, 2011.