An Efficient Distributed Algorithm for Resource Allocation in Large-Scale Coupled Systems

Di Niu
Department of Electrical and Computer Engineering
University of Alberta
dniu@ualberta.ca

Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
bli@eecg.toronto.edu

***Abstract*—In modern large-scale systems, fast distributed resource allocation and utility maximization are becoming increasingly important. Traditional solutions to such problems rely on primal/dual decomposition and gradient methods, whose convergence is sensitive to the choice of the stepsize and may not be sufficient to satisfy the requirement of large-scale real-time applications. We propose a new iterative approach to distributed resource allocation in coupled systems. Without complicating message-passing, the new approach is robust to parameter choices and expedites convergence by exploiting problem structures. We theoretically analyze the asynchronous algorithm convergence conditions, and empirically evaluate its benefits in a case of cloud network resource reservation based on real-world data.**

I. Introduction

Resource allocation in distributed systems is often modeled as utility maximization problems, which are solved using primal/dual decomposition and gradient/subgradient methods [1]. Though these problems arise in diverse applications, two general trends are visible.

First, due to the explosion in scale of modern systems, it is increasingly important to develop fast and robust distributed solutions to these problems that could meet the requirements of large-scale real-time applications. *Second*, an increasing level of coupling is observed in distributed systems. For example, in a cost-effective resource allocation problem, each user, when consuming a certain amount of resources, will enjoy a corresponding utility, while all the users may together incur a coupled cost at the resource provider due to multiplexing. The goal is to maximize the utility of all participants, including the resource provider, that is to maximize the sum of user utilities minus the coupled provider cost. As another example, when multiple users download movies from the same server, each user has a utility as a function of the bytes it wishes to download, but suffers from a delay penalty due to the congestion incurred by all these downloads [2]. A further example is DSL spectrum management of copper wires in a cable binder, where a user's utility is a function of the signal-to-interference ratios that depend on the transmit powers of other users [3].

In this paper, we propose a new distributed approach to resource allocation problems with coupled objectives, with a goal towards fast and robust convergence in large-scale systems. Let us illustrate the basic idea of the new approach with the simple example of maximizing the sum of user utilities minus a coupled provider cost. The algorithm proceeds in the following iterative steps: given all the resource allocation requirements from users, the provider computes a price vector as a weighted average of the marginal cost under current resource requirements and the previous price; each user then computes its resource requirement under the new price by maximizing its surplus, that is, its utility minus the price.

The new approach distinguishes itself from traditional gradient methods in three aspects. *First*, while the performance of gradient methods is sensitive to stepsize selection, the parameter in our approach is a weight between 0 and 1, which is easier to set and achieves robustness. *Second*, our approach can be understood as a novel distributed version of the nonlinear Jacobi algorithm [4], and only passes the gradient information of certain utilities, e.g., the cost function, instead of all the utilities. By exploiting the structural difference between utility functions, our approach can achieve much faster convergence in certain scenarios. *Third*, in certain engineering applications, unlike gradient methods, our approach satisfies a *contraction* assumption [4], and thus converges even if executed asynchronously among network elements, further increasing convergence speed. While enjoying these advantages, the new approach only feeds back gradient information, and has the same economic interpretation as pricing.

We provide asynchronous convergence conditions of the new approach and compare its convergence speed with that of gradient methods both analytically and through experiments based on real-world trace data. Furthermore, we show how to generalize the basic idea above, with the aid of duality theory, to solve a general class of problems with additive coupled objective functions, which is referred to as *consensus optimization* [5]. Consensus optimization is a general form for posing and solving distributed optimization problems and has many applications in statistical and machine learning. Therefore, the proposed approach is also an efficient decentralized solution to such machine learning problems that involve high-dimensional data and a large number of training samples.

The roadmap of this paper is outlined as follows. Sec. II formulates a basic resource allocation problem with coupling and depicts a pricing framework to explain distributed algorithms throughout the paper. In Sec. III, we describe traditional methods and propose our new algorithm based on pricing. In Sec. IV, we rigorously analyze the algorithm

convergence conditions, especially, in an asynchronous sense. In Sec. V, through an example of quadratic programming, we draw insights on the performance of different algorithms. In Sec. VI, we generalize the basic algorithm to solve distributed optimization problems with general additive coupled objectives. In Sec. VII, we demonstrate the benefits of the new approach through a numerical case study of bandwidth reservation in a cloud computing system based on real-world traces. In Sec. VIII, we present related work. We conclude the paper in Sec. IX.

II. Pricing in a Basic Coupled System

We consider a basic resource allocation problem of the form

$$\max_{x} \sum_{i=1}^{n} U_i(x_i) - C(x) \quad (1)$$
$$\text{s.t.} \quad x_i \in [a_i, b_i], \quad i = 1, \ldots, n,$$

for some real numbers a_i, b_i, where $x = (x_1, \ldots, x_n)$ represents resource allocation, each U_i is a strictly concave and monotonically increasing utility function, and $C(x)$ is a strictly convex cost function that is monotonically increasing in each x_i. Assume C and U_i are twice continuously differentiable. In Sec. VI, we will see that based on duality theory, problems with general coupled objectives can be transformed into (1).

Problem (1) often arises in networking systems, where each user i is associated with an utility $U_i(x_i)$ by using x_i units of some resource, while the users incur a total cost $C(x)$ at the resource provider, which may be coupled among x_i's in an arbitrary way depending on the technology used. The goal of the resource provider is to make resource allocation decision x wisely, so that the social welfare $\sum_i U_i(x_i) - C(x)$ is maximized. Apparently, granting the maximum resources to users will maximize the sum of their utilities, but will also incur a high cost $C(x)$. As a convex optimization problem, (1) has a unique solution $x^* = (x_1^*, \ldots, x_n^*)$. However, it is undesirable to solve (1) in a centralized way using conventional algorithms such as interior point methods [6], because the utility U_i is not known to the resource provider or other users, and the cost function C is unknown to the users.

Distributed solutions to problem (1) are feasible via pricing. The basic idea is that the provider should charge each user i a price $p_i x_i$ for using resource x_i, where the unit price p_i can be heterogeneous among users. Given a price vector $p = (p_1, \ldots, p_n)$, problem (1) is equivalent to

$$\max_{x \in \prod_i [a_i, b_i]} \sum_{i=1}^{n} \big(U_i(x_i) - p_i x_i\big) + \big(p^{\mathsf{T}} x - C(x)\big), \quad (2)$$

where $U_i(x_i) - p_i x_i$ is the *surplus* of user i for using resource x_i, and $p^{\mathsf{T}} x - C(x)$ is the *profit* of the provider.

A pricing-based iterative solution to problem (2) can be interpreted as follows. The provider sets the price vector p, under which each user will maximize its surplus and choose to use resource

$$x_i(p_i) := \arg\max_{x_i \in [a_i, b_i]} U_i(x_i) - p_i x_i, \quad \forall i. \quad (3)$$

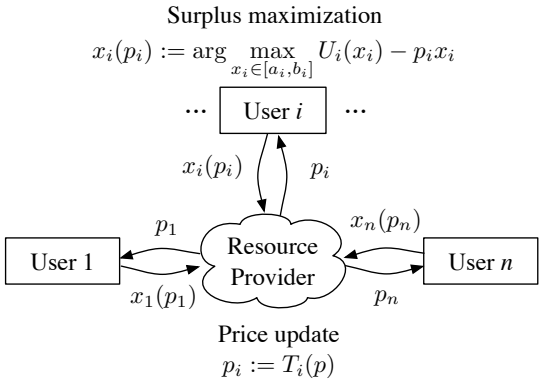

Fig. 1. The iterative process of price updating. The price update for each user i can be performed asynchronously.

Based on all the returned x_i, the provider aims to update the price vector p iteratively, such that $x(p) = (x_1(p_1), \ldots, x_n(p_n))$ produced by the surplus-maximizing users will eventually converge to the optimal resource allocation x^*.

Such a process is illustrated in Fig. 1. Note that the returned $x_1, ..., x_n$ can be used in the updates of all p_i. However, the price updating does not have to be synchronous for all the users. In other words, there could be heterogeneous message-passing delays between the provider and different users, and it is allowed that p_i and x_i for a particular user i have been updated for any number of iterations, before p_j ($j \neq i$) for other users are updated.

We assume each network element can only use its local information and the available feedback. Specifically, we make the following *locality* assumptions on price updates:

Assumption 1: When the resource provider sets the price, the price update for each user i is a function of the current price p and all the returned user resource requirements $x_1(p_1), \ldots, x_n(p_n)$ given by (3). Furthermore, the provider has full knowledge of the cost function C but no knowledge of any user's utility function U_i.

The price update rule can be rewritten as a function

$$p_i := T_i(p), \quad (4)$$

with $T = (T_1, \ldots, T_n)$. The above locality assumptions on the structure of T enforce price updating under the minimum message-passing overhead. Clearly, when message-passing is limited to only prices p and resource allocations $x(p)$, any algorithm that requires information beyond gradients, e.g., Newton methods, can hardly be decentralized.

III. Algorithms

Our main objective is to design an effective price updating rule T, so that $x(p(t))$ can converge to the optimal resource allocation x^* within a small number of iterations. Before presenting our new algorithm, let us review an existing popular method, based on Lagrangian dual decomposition and the gradient algorithm. We will also point out the connection between our new algorithm and the nonlinear Jacobi algorithm.

A. Dual Decomposition and Gradient Methods

We now briefly describe Lagrangian dual decomposition and interpret it as a price updating algorithm introduced in Sec. II. Problem (1) is equivalent to

$$\begin{aligned} &\max_{x,y} \textstyle\sum_{i=1}^n U_i(x_i) - C(y) \\ &\text{s.t.} \quad x = y, \quad x_i \in [a_i, b_i], \quad i = 1, \dots, n. \end{aligned} \tag{5}$$

It is easy to check that the Lagrangian dual problem of (5) is

$$\min_p q(p) = \sum_{i=1}^n \max_{x_i \in [a_i, b_i]} \{U_i(x_i) - p_i x_i\} + \max_y \{p^\mathsf{T} y - C(y)\}.$$

Since problem (5) is convex, the duality gap is zero: if p^* solves the dual problem, then $x_i^* = \arg\max_{x_i \in [a_i, b_i]} U_i(x_i) - p_i^* x_i$ solves the original problem.

Thus, we only need to solve the dual problem. It is known [1], [4], [6] that the gradient of $q(p)$ is

$$\nabla q(p) = y(p) - x(p), \tag{6}$$

where $y(p) = (y_1(p), \dots, y_n(p))$ is given by

$$y(p) = \arg\max_y p^\mathsf{T} y - C(y), \tag{7}$$

and $x(p) = (x_1(p_1), \dots, x_n(p_n))$ is defined as follows:

$$x_i(p_i) = \arg\max_{x_i \in [a_i, b_i]} U_i(x_i) - p_i x_i. \tag{8}$$

Applying a gradient algorithm to the dual problem yields the following price updating rule:

$$p_i := T_i(p) = p_i - \gamma\big(y_i(p) - x_i(p_i)\big), \tag{9}$$

where γ is a sufficiently small positive *stepsize*.

Obviously, (9) can be interpreted as the following price updating procedure: given a price vector p set by the resource provider, each user returns $x_i(p_i)$ to the provider by maximizing its own surplus. The provider then computes $y(p)$ locally by maximizing $p^\mathsf{T} y - C(y)$ and updates p according to (9). Apparently, the price update here only makes use of the current price p, the returned $x(p)$, and the local cost function C at the provider, complying with the locality assumptions on price updating.

B. The New Algorithm

We propose a new type of algorithms for solving problem (1), which satisfies the locality assumptions on T in Sec. II.

Algorithm 1: Assume the resource provider updates the price p. The price update rule is, for all i,

$$p_i := T_i(p) = (1-\gamma)p_i + \gamma \nabla_i C\big(x_1(p_1), \dots, x_n(p_n)\big), \tag{10}$$

where $\gamma \in (0, 1]$ is a *relaxation parameter*, and

$$x_i(p_i) := \arg\max_{x_i \in [a_i, b_i]} U_i(x_i) - p_i x_i \tag{11}$$

is the resource requirement collected from each user i under the current price p.

Instead of relying on dual decomposition, Algorithm 1 actually observes the optimality conditions of (1). For ease of explanation, let us ignore the constraint $x_i \in [a_i, b_i]$ for now. By first-order conditions, if x^* solves (1), we have

$$U_i'(x_i^*) = \nabla_i C(x^*), \quad \forall i. \tag{12}$$

Let us consider Algorithm 1 when $\gamma = 1$. Assume the iteration (10) has a fixed point p^*. Then at p^*, we have

$$U_i'\big(x_i(p_i^*)\big) = p_i^* = \nabla_i C\big(x_1(p_1^*), \dots, x_n(p_n^*)\big), \quad \forall i, \tag{13}$$

where the first equality is due to the first-order condition of (11). Thus, $(x_1(p_1^*), \dots, x_n(p_n^*))$ satisfies the conditions (12) and is an optimal solution x^*, as our problem is convex.

The following proposition shows that as long as Algorithm 1 converges, it will solve the optimization problem (1).

Proposition 1: Suppose that Algorithm 1 has a fixed point p^* and an associated x^*. Then, x^* solves (1).

Proof: Please refer to Appendix A for the proof. ■

In Algorithm 1, surplus maximization is performed only at users, whereas in gradient methods (9), such maximizations must be performed at both users and the provider to obtain $x(p)$ and $y(p)$. First, this greatly reduces the computational complexity when n is large. In particular, Algorithm 1 can scale to a very large n without having to solve convex optimization at the provider side. More importantly, we will see that the new algorithm can better exploit the structures of U_i and C to achieve a faster convergence rate. We will also see that it is easier to choose the parameter γ in the new algorithm, as it does not have to be very small.

C. Relationship to the Nonlinear Jacobi Algorithm

All the algorithms above have *synchronous* and *asynchronous* versions, depending on whether updates of p_i and $x_i(p)$ are performed for all users i synchronously or asynchronously. For example, in an asynchronous Algorithm 1, p_i and $x_i(p_i)$ can be updated for an user i for any finite number of iterations, before other $x_j(p_j)$ for $j \neq i$ are updated.

In the following, we show that the nonlinear Jacobi algorithm [4] is a limiting case of the asynchronous version of Algorithm 1, while Algorithm 1 can be understood as a novel decentralization of the nonlinear Jacobi algorithm. The nonlinear Jacobi algorithm consists of iteratively optimizing in a parallel fashion with respect to each variable x_i while keeping the rest of the variables x_j $(j \neq i)$ fixed. Formally, for problem (1), it is defined as

$$x_i := \arg\max_{x_i \in [a_i, b_i]} U_i(x_i) - C(x_1, \dots, x_i, \dots, x_n) + \sum_{j \neq i} U_j(x_j). \tag{14}$$

And the above update is to be conducted for all i in parallel. Note that the above nonlinear Jacobi algorithm *cannot* be applied to our problem, because the resource provider does not know any user utility U_i, and the users do not know the cost function C. Neither can the nonlinear Jacobi algorithm be interpreted as a pricing-based algorithm.

However, the nonlinear Jacobi algorithm is a limiting case of Algorithm 1 executed *asynchronously*. If for each user i, we perform (10) and (11) of Algorithm 1 for an infinite number of iterations before updating the other variables x_j (for all $j \neq i$) in equation (10) for this user, then the x_i obtained this way is the same as the one given by (14), and thus Algorithm 1 becomes the nonlinear Jacobi algorithm. In other words, the Algorithm 1 proposed here, with a convenient interpretation as pricing, is essentially a novel distributed version of the nonlinear Jacobi algorithm.

IV. Convergence Conditions

We analyze the convergence of the algorithms above, and in particular, aim to find sufficient conditions under which a certain algorithm is a *contraction mapping* [4]. Although more general convergence conditions may be found for an algorithm when executed synchronously, contraction mapping can ensure its convergence even if executed completely asynchronously [4]. Note that an asynchronous algorithm is much better than its synchronous counterpart, since message-passing between network elements usually incurs heterogeneous delays in reality. Based on the derived conditions, we will compare the algorithm robustness to the choice of γ and their convergence speeds in Sec. V, using a simple example of quadratic programming.

A. Contraction Mapping

As some preliminaries, let us introduce contraction mapping and its properties. Many iterative algorithms can be written as

$$x(t+1) := T(x(t)), \quad t = 0, 1, \ldots, \tag{15}$$

where $x(t) \in X \subset \Re^n$, and $T : X \mapsto X$ is a mapping from X into itself. The mapping T is called a *contraction* if

$$\|T(x) - T(y)\| \leq \alpha \|x - y\|, \quad \forall x, y \in X, \tag{16}$$

where $\|\cdot\|$ is some norm, and the constant $\alpha \in [0, 1)$ is called the *modulus* of T. Furthermore, the mapping T is called a *pseudo-contraction* if T has a fixed point $x^* \in X$ (such that $x^* = T(x^*)$) and

$$\|T(x) - x^*\| \leq \alpha \|x - x^*\|, \quad \forall x \in X. \tag{17}$$

The following theorem establishes the geometric convergence of both contractions and pseudo-contractions:

Theorem 1: (Geometric Convergence) Suppose that $X \subset \Re^n$ and the mapping $T : X \mapsto X$ is a contraction, or a pseudo-contraction with a fixed point $x^* \in X$. Suppose the modulus of T is $\alpha \in [0, 1)$. Then, T has a *unique* fixed point x^* and the sequence $\{x(t)\}$ generated by $x(t+1) := T\big(x(t)\big)$ satisfies

$$\|x(t) - x^*\| \leq \alpha^t \|x(0) - x^*\|, \quad \forall t \geq 0, \tag{18}$$

for every choice of the initial vector $x(0) \in X$. In particular, $\{x(t)\}$ converges to x^* geometrically.

To simplify notations, we may use $c_i(x)$ to stand for $\nabla_i C(x)$ and $u_i(x_i)$ for $U_i'(x_i)$. We denote $\partial_{x_j x_i} C(x) = \nabla_j c_i(x)$ the second order partial derivatives of C.

B. Convergence of Algorithm 1 when $\gamma = 1$

We analyze the convergence of Algorithm 1 under different choices of γ. First, suppose $\gamma = 1$. We rewrite Algorithm 1 in another form amenable to analysis. Denote $[x_i]_i^+$ the *projection* of $x_i \in \Re$ onto the interval $[a_i, b_i]$, i.e.,

$$[x_i]_i^+ = \arg\min_{z \in [a_i, b_i]} |z - x_i|, \quad i = 1, \ldots, n. \tag{19}$$

It is easy to check that (11) is equivalent to

$$x_i(p_i) := [\arg\max_{x_i} U_i(x_i) - p_i x_i]_i^+ = [u_i^{-1}(p_i)]_i^+ \tag{20}$$

Therefore, letting $\gamma = 1$ and substituting (10) into (20) yields

$$x_i := T_i(x) = [u_i^{-1}(c_i(x))]_i^+, \quad i = 1, \ldots, n, \tag{21}$$

which is an equivalent iteration to Algorithm 1 when $\gamma = 1$. The following result gives a sufficient condition for the convergence of Algorithm 1 with $\gamma = 1$.

Proposition 2: Suppose $\gamma = 1$. If, for all i, we have

$$\textstyle\sum_{j=1}^n |\partial_{x_j x_i} C(x)| < \min_{z_i} |U_i''(z_i)|, \ \forall x \in \prod_i [a_i, b_i], \tag{22}$$

then $T = (T_1, \ldots, T_n)$ given by (21) is a contraction and the sequence $\{x(p(t))\}$ generated by Algorithm 1 converges geometrically to the optimal solution x^* to (1), given any initial price vector $p(0)$.

Proof: Please refer to Appendix B for the proof. ■

C. Convergence of Algorithm 1 with Over-relaxation

Next, we consider the convergence condition when $\gamma < 1$, which is much harder to analyze in general. In many scenarios, the optimal solution x^* is in the interior of the box $\prod_i [a_i, b_i]$, i.e., $a_i < x_i^* < b_i$ for all i. If the $x(p(t))$ generated by (10) and (11) in each iteration t is always in the interior of $\prod_i [a_i, b_i]$, we can rewrite (11) as $x_i(p_i) := u_i^{-1}(p_i)$, and Algorithm 1 is equivalent to the following iteration:

$$x_i := T_i(x) = u_i^{-1}\big((1-\gamma)u_i(x_i) + \gamma c_i(x)\big), \ \forall i. \tag{23}$$

The following result gives a convergence condition of Algorithm 1 when $\gamma < 1$.

Proposition 3: Suppose $a_i < x_i^* < b_i$ for all i. Set $x_i(p_i(0))$ to be a_i for all i, or b_i for all i. If we have

$$\begin{cases} 0 < \gamma \leq \big(1 + \frac{\partial_{x_i x_i} C(x)}{|U_i''(x_i)|}\big)^{-1}, \\ \sum_{j \neq i} \partial_{x_j x_i} C(x) \leq 0, \end{cases} \tag{24}$$

for all x between $x(p(0))$ and x^*, for all i, then $T = (T_1, \ldots, T_2)$ given by (23) is a pseudo-contraction between $x(p(0))$ and x^*, and the sequence $\{x(p(t))\}$ generated by Algorithm 1 converges to x^* geometrically.

Proof: Please refer to Appendix B for the proof. ■

In many engineering problems, users may have *a priori* knowledge of (a_i, b_i) that contains its optimal x_i^*, which justifies the assumption that x^* resides in the interior of $\prod_i [a_i, b_i]$. Furthermore, setting $x_i(p_i(0))$ to a_i (or b_i) is easy: the resource provider only needs to set a very high (or very low) $p_i(0)$ to push the user resource requirement $x_i(p_i)$ to its corresponding boundary value. Conditions (24) essentially ensure that $x(p(t))$ increases (or decreases) from a (or b) to x^* monotonically.

D. Convergence of Gradient Methods for Dual Decomposition

There are many results on the convergence of gradient algorithms. For example, for a sufficiently small constant stepsize γ, the algorithm is guaranteed to converge to the optimal value [6]. For comparison, here we derive a sufficient condition under which the gradient algorithm applied to the dual problem, i.e., iteration (9), is a contraction for the unconstrained version of problem (1).

Proposition 4: Suppose that problem (1) has no constraints and that the Hessian matrix of $C(x)$ is $H(x) = [\partial_{x_j x_i} C(x)]_{n\times n}$. Denote the inverse of $H(x)$ as $P(x) = [H(x)]^{-1} = [P_{ij}(x)]_{n\times n}$.

If we have the following conditions for all x and i:

$$\begin{cases} 0 < \gamma \le \left(P_{ii}(x) + \frac{1}{|U_i''(x_i)|}\right)^{-1}, \\ \sum_{j\neq i} |P_{ij}(x)| < P_{ii}(x) + \frac{1}{|U_i''(x_i)|}, \end{cases} \tag{25}$$

then $T = (T_1, \ldots, T_n)$ given by (9) is a contraction and the sequence $\{x(p(t))\}$ converges geometrically to x^*.

Proof: Please refer to Appendix B for the proof. ∎

When the constraints $a_i \le x_i \le b_i$ are considered, the contraction property is hard to analyze in general, since in this case, $f_i(p) = y_i(p) - [u_i^{-1}(p_i)]_i^+$ involves projection and is not continuously differentiable.

As a final note, when an update rule T is a contraction or pseudo-contraction with respect to the maximum norm $\|\cdot\|_\infty$, *asynchronous* convergence of the corresponding algorithm can be established using Proposition 2.1 in [4] (pp. 431). In other words, if the sufficient conditions in this section hold, the corresponding algorithms also converge *asynchronously*.

V. Quadratic Programming: Convergence Performance Comparisons

We compare the convergence conditions and speeds of different algorithms using an example of quadratic programming. A quadratic cost C and utilities U_i simplify the conditions derived in Sec. IV, allowing us to draw insights on the algorithm convergence performance.

Suppose $U_i(x_i) = \frac{1}{2}(\beta_i x_i^2 + \rho_i x_i + \tau_i)$ and $C(x) = \frac{1}{2} x^{\mathsf{T}} A x$, where $\beta_i < 0$ for all i, and $A = [a_{ij}]_{n\times n}$ is an $n \times n$ positive semi-definite symmetric matrix. Then the unconstrained version of problem (1) becomes

$$\max_x \ \frac{1}{2}\left(\sum_{i=1}^n (\beta_i x_i^2 + \rho_i x_i + \tau_i) - x^{\mathsf{T}} A x\right). \tag{26}$$

It is easy to check that $U_i''(x_i) = \beta_i$ and $\partial_{x_j x_i} C(x) = a_{ij}$, for all x. Furthermore, the Hessian matrix of $C(x)$ is $H(x) = A$, and the inverse of $H(x)$ is $P(x) = [P_{ij}(x)]_{n\times n} = A^{-1}$. If we denote A^{-1} as $[a'_{ij}]_{n\times n}$, then $P_{ij}(x) = a'_{ij}$, for all x.

According to Proposition 2 and Proposition 3, Algorithm 1 will converge to x^*, if

$$\begin{cases} \gamma = 1, \\ \sum_{j=1}^n |a_{ij}| < |\beta_i|, \ \forall i, \end{cases} \tag{27}$$

or if

$$\begin{cases} 0 < \gamma \le (1 + \frac{a_{ii}}{|\beta_i|})^{-1}, \\ \sum_{j\neq i} a_{ij} \le 0, \end{cases} \quad \forall i. \tag{28}$$

According to Proposition 4, the gradient algorithm applied to the dual decomposition will converge to x^*, if

$$\begin{cases} 0 < \gamma \le (a'_{ii} + \frac{1}{|\beta_i|})^{-1}, \\ \sum_{j\neq i} |a'_{ij}| < a'_{ii} + \frac{1}{|\beta_i|}, \end{cases} \quad \forall i. \tag{29}$$

Let us now analyze the algorithm convergence speeds. Since the convergence speed of a contraction is governed by its modulus α, we derive $\alpha_1(\gamma)$ and $\alpha_D(\gamma)$ for Algorithm 1 and dual decomposition, respectively, under different choices of the parameter γ.

If $\gamma = 1$, from the proof of Proposition 2, we obtain

$$\alpha_1(1) = \max_i \sum_{j=1}^n \frac{|a_{ij}|}{|\beta_i|}.$$

If $\gamma < 1$, for dual decomposition, we have

$$\begin{aligned} \alpha_D(\gamma) &= \max_i |1 - \gamma(\frac{1}{|\beta_i|} + a'_{ii})| + \gamma \textstyle\sum_{j\neq i} |a'_{ij}| \\ &\ge \max_i \frac{\sum_{j\neq i} |a'_{ij}|}{(a'_{ii} + 1/|\beta_i|)}, \end{aligned} \tag{30}$$

where the equality is achieved when a different γ_i is allowed for each user i and $\gamma_i = (a'_{ii} + 1/|\beta_i|)^{-1}$. The right hand side of the above inequality represents the best convergence speed of dual decomposition with the gradient algorithm.

A. Discussions

Let us compare the convergence speeds of different algorithms. Note that the smaller the modulus α is, the faster the corresponding contraction converges. Algorithm 1 may potentially achieve much faster convergence if $|\beta_i|$ is big relative to $|a_{ij}|$, which is often the case in communication and networking systems, where the service cost C is low due to multiplexing or due to low unit cost for resources. On the contrary, if $|\beta_i|$ is small relative to $|a_{ij}|$, the gradient algorithm for the dual problem will converge faster. Therefore, when gradient methods for dual decomposition fail to be a contraction (and thus a very small stepsize γ must be chosen), we may resort to Algorithm 1 to achieve fast and asynchronous convergence.

Another obvious strength of the new algorithm is its robustness and simplicity in parameter selection. Under certain circumstances (e.g., when $\sum_{j=1}^n |a_{ij}| < |\beta_i|, \ \forall i$), choosing $\gamma = 1$ suffices to achieve fast convergence for the new algorithm.

Even if $\gamma < 1$, it is also much easier to set γ in the new algorithm than in dual decomposition with the gradient algorithm. As has been analyzed, for dual decomposition to achieve its best convergence speed (given it is a contraction), γ should be set around $(a'_{ii} + 1/|\beta_i|)^{-1}$. This requires one to have good estimates about the absolute values of a'_{ii}, $|\beta_i|$; otherwise, γ usually needs to be set to a small value to guarantee convergence, which, however, leads to slow convergence.

In contrast, for Algorithm 1 to achieve its best convergence speed (given it is a pseudo-contraction), γ should be set to around $(1 + a_{ii}/|\beta_i|)^{-1}$, which only depends on the relative

values of a_{ii} and $|\beta_i|$ and is thus easier to estimate. For example, for Algorithm 1, to choose a $\gamma < 1$ that satisfies condition (28) is easy: if $a_{ii} \leq |\beta_i|$, for all i, which is often the case when the service cost is low, we can simply set γ to 0.5.

VI. Handling General Coupled Objectives

The proposed algorithm does not only solve (1). We now show how to use Algorithm 1 to approach a general form of optimization problems with coupled objective functions, referred to as *consensus optimization* [5]:

$$\begin{aligned} &\min_x \textstyle\sum_{i=0}^m F_i(x) \\ &\text{s.t.} \quad x \in P_i, \quad i = 0, 1, ..., m, \end{aligned} \tag{31}$$

where $F_i : \Re^n \mapsto \Re$ are strictly convex functions, and P_i are bounded polyhedral subsets of $\Re^n$. We assume at least F_0 depends on all n coordinates of the vector x and $P_0 \subset P_i$.

The consensus optimization problem (31) not only models distributed resource allocation with arbitrary coupling among utility functions, but also finds wide applications in machine learning problems [5]. For example, in model fitting, the vector x represents the set of parameters in a model and F_i represents the loss function associated with the ith training sample. The learning goal is to find the model parameters x that minimizes the sum of loss functions for all training samples. Note that (1) is a special case of (31) in that each U_i only depends on the ith coordinate of x, while C plays the role of F_0.

We show that based on duality theory, consensus optimization (31) can be converted into an equivalent form of (1). Introducing auxiliary variables $x_i \in \Re^n$, (31) is equivalent to

$$\begin{aligned} &\min_x F_0(x) + \textstyle\sum_{i=1}^m F_i(x_i) \\ &\text{s.t.} \quad x \in P_0, \quad x_i = x, \quad x_i \in P_i, \quad i = 1, ..., m. \end{aligned} \tag{32}$$

It is easy to check that the dual problem of (32) is

$$\max_p q(p) = q_0(\textstyle\sum_{i=1}^m p_i) + \sum_{i=1}^m q_i(p_i) \tag{33}$$

where $p_i \in \Re^n$ are the Lagrangian multipliers, and

$$q_0(\textstyle\sum_{i=1}^m p_i) = \min_{x \in P_0} \{F_0(x) - (\sum_{i=1}^m p_i)^\mathsf{T} x\}, \tag{34}$$

$$q_i(p_i) = \min_{x_i \in P_i} \{F_i(x_i) + p_i^\mathsf{T} x_i\}. \tag{35}$$

The dual function $q(p)$ is continuously differentiable and

$$\frac{\partial q(p)}{\partial p_i} = \frac{\partial q_0(p)}{\partial p_i} + \frac{\partial q_i(p_i)}{\partial p_i} = -x(p) + x_i(p_i), \tag{36}$$

where $x(p)$ and $x_i(p_i)$ are the unique minimizing vectors in (34) and (35), respectively. A traditional gradient method would update p in the direction of its gradient $\nabla q(p)$.

Alternatively, we can apply Algorithm 1 to the dual problem (33) with p as the variable by setting $U_i(p_i) = q_i(p_i)$ and $C(p) = -q_0(p)$, leading to the alternating iterations for all i:

$$\tilde{x}_i := (1-\gamma)\tilde{x}_i + \gamma \nabla_i C(p) = (1-\gamma)\tilde{x}_i + \gamma x(p) \tag{37}$$

$$p_i := \arg\max_{p_i} q_i(p_i) - p_i^\mathsf{T} \tilde{x}_i, \tag{38}$$

where $\tilde{x}_i$ is the price set for F_i. We can further simplify the update (38) for p_i. We assume $P_0 \subset P_i$ for all i. Since $x(p) \in P_0$, from (37), we have $\tilde{x}_i \in P_0 \subset P_i$. Clearly, p_i is the solution to the system of equations $\frac{\partial q_i(p_i)}{\partial p_i} - \tilde{x}_i = 0$, or equivalently, $x_i(p_i) - \tilde{x}_i = 0$. When $x_i(p_i) = \tilde{x}_i$ is in the interior of P_i, from the first-order conditions of (35), we have

$$p_i = -\frac{\partial F_i(\tilde{x}_i)}{\partial x_i}. \tag{39}$$

Therefore, Algorithm 1, when applied to consensus optimization (31), is fully described by the iterations (37) and (39), where p_i are variables, $\tilde{x}_i$ are prices, F_0 plays the role of a "resource provider" and each F_i is a "user". Each "user" updates its variable p_i in a distributed fashion based on the price $\tilde{x}_i$ computed and sent by the "provider" until convergence.

Note that $P_0 \subset P_i$ for all i is a condition to use Algorithm 1, since otherwise, $\tilde{x}_i$ may reside outside of P_i and the gradient of $q_i(p_i) - p_i^\mathsf{T}\tilde{x}_i$, which is $x_i(p_i) - \tilde{x}_i$, cannot be zero. In this case, (38) is not well defined. When $P_0 \subset P_i$, we can apply Algorithm 1 to the unconstrained dual problem (33), which is equivalent to (31).

VII. Cloud Network Reservation: Trace-Driven Simulations

We study a typical large-scale real-time cloud resource reservation problem, first presented in [7]. In this study, we show that contraction assumptions for gradient algorithms do not often hold in reality, which causes slow convergence and non-trivial challenges for stepsize selection.

Suppose that n video companies have random bandwidth demands $D_1, \ldots, D_n$ from end users. As the *tenants* of a cloud provider, these video companies want to reserve the egress network bandwidth to guarantee smooth video delivery to their end users. We assume that the reservation of egress network bandwidth from cloud data centers has already been enabled by the detailed engineering techniques proposed in [8]–[10].

When guaranteeing x_i portion of its demand D_i, video company i enjoys a utility $U_i(x_i, D_i)$, e.g.,

$$U_i(x_i, D_i) = \alpha_i x_i D_i - \mathrm{e}^{B_i(D_i - x_i D_i)}, \tag{40}$$

which represents a linear utility gain from usage $x_i D_i$ and a convex increasing utility loss $\mathrm{e}^{B_i(1-x_i)D_i}$ for the demand $(1-x_i)D_i$ from end users that are not guaranteed. The cloud performs multiplexing to save bandwidth reservation cost: given a service violation probability ϵ, it needs to reserve a total bandwidth of K such that $\Pr(\sum_i x_i D_i > K) = \epsilon$. The problem is to decide the guaranteed portions $x = (x_1, \ldots, x_n)$ to maximize the expected *social welfare*, i.e.,

$$\max_x \mathrm{SW}(x) = \textstyle\sum_{i=1}^n \mathbf{E}_{D_i}[U_i(x_i, D_i)] - \beta K(x), \tag{41}$$

$$\text{s.t.} \quad x_i \in [0, 1], \quad i = 1, \ldots, n, \tag{42}$$

where β is the cost of reserving a unit amount of bandwidth.

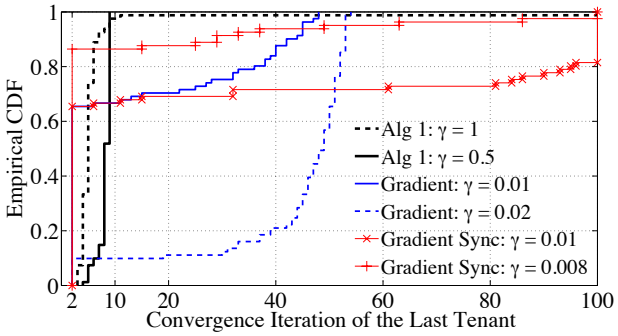

Fig. 2. CDF of convergence iterations in all 81 10-minute periods.

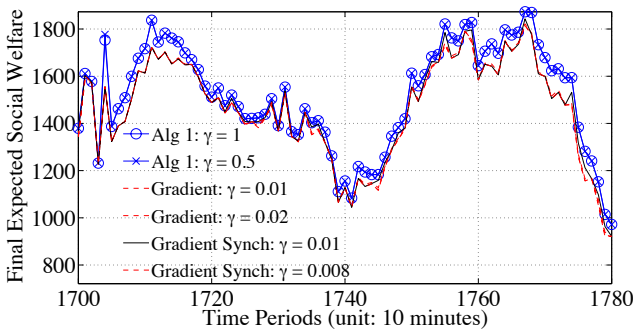

Fig. 3. Final objective values $\mathrm{SW}(x^*)$ produced by different algorithms.

Suppose $D_1, \ldots, D_n$ are Gaussian with predictable means $\mu = (\mu_1, \ldots, \mu_n)$, standard deviations $\sigma = (\sigma_1, \ldots, \sigma_n)$, and the covariance matrix $\Sigma = [\sigma_{ij}]_{n\times n}$. Then We have

$$\mathbf{E}[U_i(x_i, D_i)] = \alpha_i x_i \mu_i - \mathrm{e}^{B_i(1-x_i)\mu_i + \frac{1}{2}B_i^2(1-x_i)^2\sigma_i^2}, \quad (43)$$

and under Gaussian demands, the bandwidth reservation K is

$$K(x) = \mu^{\mathsf{T}} x + \theta(\epsilon)\sqrt{x^{\mathsf{T}}\Sigma x}, \quad (44)$$

where $\theta(\epsilon) = F^{-1}(1-\epsilon)$ is a constant, F being the CDF of normal distribution $\mathcal{N}(0,1)$ [7]. Now it is clear that (41) is a convex problem and can be solved through pricing, i.e., the cloud can charge user i a price $p_i x_i$ to control the user demand x_i towards expected social welfare maximization.

Our study is based on a video demand dataset from a real-world commercial VoD system [7]. Please refer to [7] for the data description. The data used here contains the bandwidth demands of $n = 468$ video channels (each we suppose to be a tenant of the cloud), measured every 10 minutes, over a 810-minute period during 2008 Olympics. Assume the demand statistics μ, Σ of tenants remain the same within each 10-minute period. We can estimate μ, Σ for the upcoming 10 minutes based on historical demands using the time-series forecasting techniques presented in [7], [11]. Once μ, Σ are obtained, we solve problem (41) for an optimal allocation x in this period. After the 10-minute period has past, the system proceeds to estimate μ, Σ for the next 10 minutes and computes a new x from (41). Since the cloud does not know U_i and the tenants do not know K, (41) must be solved distributively with the minimum message-passing. Furthermore, (41) must be solved in at most a few seconds in order not to delay the entire resource allocation process performed every 10 minutes. We set $\epsilon = 0.01$, $\alpha_i = 1$ $B_i = 0.5$, $\beta = 0.5$.

We apply different algorithms to (41) for the 81 consecutive 10-minute periods. To select algorithms, for the first period, we find that the condition (24) is satisfied, and (22) is satisfied for most but not all i. We may thus tentatively believe Algorithm 1 is contracting when $\gamma = 0.5$ and probably contracting when $\gamma = 1$. However, condition (25) is not satisfied, as the Hessian of $K(x)$ is small due to multiplexing and its inverse is big. Thus, the gradient method for dual decomposition is not a contraction. As a benchmark algorithm, the gradient method for the dual problem, although not contracting, can still converge theoretically when stepsize γ is sufficiently small.

The initial price $p(1)$ is set to the optimal price vector $\beta(\mu + \theta(\epsilon)\sigma)$ without multiplexing, which is easy to compute due to no coupling in the cost function [7]. Since the value of $\mathrm{SW}(x)$ is unknown to the cloud, we let an algorithm stop at iteration t if either $\|x(t) - x(t-1)\|_\infty < 10^{-2}$ or $t = 100$. Since Algorithm 1 is a contraction, which even converges asynchronously, we adopt an asynchronous stop criterion: if $|x_i(t) - x_i(t-1)| < 10^{-2}$, then tenant i stops updating x_i. In contrast, as the gradient method is not a contraction, we evaluate both synchronous and asynchronous stop criteria when it is used. Note that we do not assume heterogeneous message-passing delays, which may otherwise further strengthen the benefit of our asynchronous algorithm.

Fig. 2 plots the CDF of convergence iterations needed in all 81 experiments, one for each 10-minute period. Algorithm 1 always converges in 10 iterations, when $\gamma = 0.5$. If $\gamma = 1$, the convergence rate can be further speeded up to 5 iterations on average, although it does not converge in one single period, due to violation of the contraction assumption. In general, gradient methods need much longer time to converge, and also face a dilemma in selecting the stepsize γ. Since algorithms will stop when x is changing by less than 10^{-2}, we cannot set too small a γ, in which case, the algorithm will stop when $t = 2$ due to too small a change in $p(t)$. Fig. 2 shows that gradient methods with all considered γ produce erroneous outputs in many cases, as they stop at $t = 2$. Increasing γ reduces errors, but will lead to even longer convergence time. Fig. 2 suggests that unlike Algorithm 1, no constant stepsize γ for gradient methods can achieve correct outputs and fast convergence at the same time for all 81 experiments.

A further check into Fig. 3 shows that the final objective $\mathrm{SW}(x^*)$ produced by Algorithm 1 is also higher than those of the gradient methods. Gradient methods with asynchronous stop rules, although converged, may output wrong answers because they are not contractions and in theory, should not be applied with asynchronous stop rules. Gradient methods with synchronous stopping achieve slightly higher objective values if not stopping at $t = 2$. But since they do not converge within 100 iterations, they are still inferior to Algorithm 1.

Let us zoom into the first 10-minute period and see how

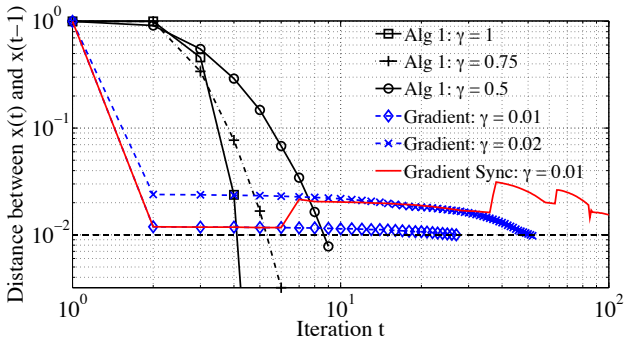

Fig. 4. The evolution of $\|x(t) - x(t-1)\|_\infty$ in the first experiment.

convergence happens for this experiment. From Fig. 4, we see that Algorithm 1 has an increasing convergence speed as t increases. In contrast, gradient methods perform well in the first two iterations, but its convergence slows down as the gradient becomes smaller. It is confirmed that gradient methods are extremely sensitive to γ, the choice of which is a non-trivial technical challenge.

Finally, the analysis above does not take into account the additional benefit of Algorithm 1 regarding the computational complexity at each node. Both algorithms have the same complexity at each user, which is a 1-dimensional surplus maximization. However, in Algorithm 1, the provider only needs to compute a cost gradient ∇C straightforwardly, whereas in gradient methods, the provider needs to solve a big 468-dimensional profit maximization problem for $y(p)$.

VIII. Related Work

Network resource allocation is often modeled as Network Utility Maximization (NUM) problems [1], [3], [12]. A number of primal/dual decomposition methods have been proposed to solve such problems in a distributed way. Most NUM problems in the literature are concerned with uncoupled utilities where the local variables of one element do not affect the utilities of other elements, which does not apply to systems with competition or cooperation, where utilities are often coupled.

Traditional distributed solutions to NUM with coupled objectives seek decomposition from Lagrangian dual problems with auxiliary variables, which are then solved with gradient/subgradient methods. This approach has a simple economic interpretation of *consistency pricing* [3], [13]. However, gradient methods are sensitive to the choice of stepsizes, leading to slow and unstable convergence. Furthermore, existing faster numerical convex problem solvers, e.g., *Newton methods* [4], can hardly be applied in a distributed way, as they involve a large amount of message-passing beyond gradient information, which is hard to implement and justify physically in reality.

The *nonlinear Jacobi algorithm* [4] solves convex problems by optimizing the objective function for each variable in parallel while holding other variables unchanged, and converges under certain contraction conditions. However, it cannot be applied to our problem, as no network element has global information of all the utilities. However, the Jacobi algorithm is a limiting case of Algorithm 1 executed asynchronously. If for each i, we perform (10) and (11) for an infinite number of iterations before updating x_j ($j \neq i$), then Algorithm 1 becomes the nonlinear Jacobi algorithm. In other words, Algorithm 1 is essentially a distributed version of the Jacobi algorithm. When Algorithm 1 is a contraction, it converges asynchronously and thus its asynchronous limiting case, the non-linear Jacobi algorithm, also converges.

Pricing in distributed systems has also been approached with game theory [14], [15]. However, this paper is concerned with achieving fast and robust convergence in large-scale systems, while incentive issues are not our focus.

IX. Concluding Remarks

In this paper, we propose a new algorithm for large-scale distributed resource allocation with coupled objectives, and analyze the asynchronous algorithm convergence conditions. Trace-driven simulations show that the proposed method can speed up convergence by 5 times over gradient methods in a real-time cloud network reservation problem with better robustness to parameter changes. It remains open to explore the applicability of the new algorithm to problems with coupled constraints.

References

[1] 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*, vol. 95, no. 1, pp. 255–312, January 2007.

[2] J. K. MacKie-Mason and H. R. Varian, "Pricing Congestible Network Resources," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1141–1149, 1995.

[3] D. P. Palomar and M. Chiang, "A Tutorial on Decomposition Methods for Network Utility Maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, August 2006.

[4] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[5] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[6] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Athena Scientific, 2003.

[7] D. Niu, C. Feng, and B. Li, "Pricing Cloud Bandwidth Reservations under Demand Uncertainty," in *Proc. of ACM SIGMETRICS*, 2012.

[8] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: a Data Center Network Virtualization Architecture with Bandwidth Guarantees," in *Proc. of ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2010.

[9] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *Proc. of SIGCOMM'11*, Toronto, ON, Canada, 2011.

[10] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The Only Constant is Change: Incorporating Time-Varying Network Reservations in Data Centers," in *Proc. of ACM SIGCOMM*, 2012.

[11] D. Niu, Z. Liu, B. Li, and S. Zhao, "Demand Forecast and Performance Prediction in Peer-Assisted On-Demand Streaming Systems," in *Proc. of IEEE INFOCOM Mini-Conference*, 2011.

[12] X. Lin, N. B. Shroff, and R. Srikant, "A Tutorial on Cross-Layer Optimization in Wireless Networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1452–1463, August 2006.

[13] C. W. Tan, D. P. Palomar, and M. Chiang, "Distributed Optimization of Coupled Systems With Applications to Network Utility Maximization," in *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing*, Toulouse, France, 2006.

[14] F. P. Kelly, "Charging and Rate Control for Elastic Traffic," *European Transactions on Telecommunications*, vol. 8, pp. 33–37, 1997.
[15] R. Johari and J. N. Tsitsiklis, "Efficiency Loss in a Network Resource Allocation Game," *Mathematics of Operations Research*, vol. 29, no. 3, pp. 407–435, August 2004.

APPENDIX A
PROOFS OF SEC. III

Proof of Proposition 1: One approach is to show the fixed point x^* satisfies the KKT conditions of (1). Here, we present a simpler proof based on the following lemma, the proof of which can be found in [4] (pp. 210, Proposition 3.1):

Lemma 1: Suppose F is convex on a convex set X. Then, $x \in X$ minimizes F over X, if and only if $(y-x)^{\mathsf{T}}\nabla F(x) \geq 0$ for every $y \in X$.

Suppose Algorithm 1 has a fixed point p^* and an associated x^*. Then we have

$$\begin{cases} p_i^* = \nabla_i C(x^*), \quad \forall i, \\ x_i^* = \arg\max_{x_i \in [a_i,b_i]} U_i(x_i) - p_i^* x_i, \quad \forall i. \end{cases} \tag{45}$$

Since x_i^* maximizes $U_i(x_i) - p_i^* x_i$ on $[a_i, b_i]$, by Lemma 1,

$$(y_i - x_i^*)(U_i'(x_i^*) - p_i^*) \leq 0, \quad \forall y_i \in [a_i, b_i], \ \forall i.$$

Since $p_i^* = \nabla_i C(x^*)$, we have

$$\textstyle\sum_i (y_i - x_i^*)(U_i'(x_i^*) - \nabla_i C(x^*)) \leq 0, \ \forall y \in \prod_i [a_i, b_i], \tag{46}$$

which, by Lemma 1, means x^* solves (1). ∎

APPENDIX B
PROOFS OF SEC. IV

Proof of Proposition 2: For each i, define a function $g_i : [0,1] \mapsto \Re$ by

$$g_i(t) = u_i^{-1}\big(c_i(z(t))\big) = u_i^{-1}\big(c_i(tx + (1-t)y)\big). \tag{47}$$

Note that g_i is continuously differentiable. We have

$$\begin{aligned} |T_i(x) - T_i(y)| &= |[u_i^{-1}(c_i(x))]_i^+ - [u_i^{-1}(c_i(y))]_i^+| \\ &\leq |g_i(1) - g_i(0)| = \left| \int_0^1 \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \mathrm{d}t \right| \\ &\leq \int_0^1 \left| \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \right| \mathrm{d}t \leq \max_{t \in [0,1]} \left| \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \right|, \end{aligned}$$

where the first inequality is because $|[x_i]_i^+ - [y_i]_i^+| \leq |x_i - y_i|$ for all $x_i, y_i \in \Re$. Furthermore, the chain rule yields

$$\begin{aligned} \left| \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \right| &= \left| \textstyle\sum_{j=1}^n \nabla_j u_i^{-1} \circ c_i(tx + (1-t)y) \cdot (x_j - y_j) \right| \\ &\leq \left| (u_i^{-1})'(c_i(z(t))) \right| \cdot \textstyle\sum_{j=1}^n \left| \nabla_j c_i(z(t)) \right| \cdot |x_j - y_j|, \end{aligned}$$

where the function $u_i^{-1} \circ c_i(x)$ is equivalent to $u_i^{-1}(c_i(x))$. If condition (22) holds, we have for all $x \in \prod_i [a_i, b_i]$,

$$\begin{aligned} \textstyle\sum_{j=1}^n |\nabla_j c_i(x)| &< \min_{z_i} |u_i'(z_i)| \leq |u_i'\big(u_i^{-1}(c_i(x))\big)| \\ &= 1/|(u_i^{-1})'(c_i(x))| \end{aligned} \tag{48}$$

Therefore, there exists a positive $\alpha < 1$ such that

$$\left| \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \right| \leq \alpha \max_j |x_j - y_j| = \alpha \|x - y\|_\infty, \ \forall t \in [0,1], \ \forall i.$$

Therefore, we have shown that

$$\|T(x) - T(y)\|_\infty \leq \alpha \|x - y\|_\infty, \quad \forall x, y \in \textstyle\prod_i [a_i, b_i],$$

and T is a contraction with modulus α with respect to the maximum norm in $\prod_i [a_i, b_i]$. By Theorem 1, Algorithm 1 converges geometrically to x^*. ∎

Proof of Proposition 3: For each i, define a function $g_i : [0,1] \mapsto \Re$ by

$$g_i(t) = u_i^{-1}\big((1-\gamma)u_i(z_i(t)) + \gamma c_i(z(t))\big), \tag{49}$$

where $z(t) = tx + (1-t)x^*$.

Now suppose $x < x^*$. Let us show that $x_i < T_i(x) \leq x_i^*$ for all i. We have

$$T_i(x) - T_i(x^*) = g_i(1) - g_i(0) = \int_0^1 \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} \mathrm{d}t, \tag{50}$$

where $\mathrm{d}g_i(t)/\mathrm{d}t$ is given by

$$\begin{aligned} \frac{\mathrm{d}g_i(t)}{\mathrm{d}t} &= (u_i^{-1})'\big((1-\gamma)u_i(z_i(t)) + \gamma c_i(z(t))\big) \cdot \big((1-\gamma) \\ &\quad u_i'(z_i(t))(x_i - x_i^*) + \gamma \textstyle\sum_j \nabla_j c_i(z(t))(x_j - x_j^*)\big) \\ &= \big(u_i'(T_i(z(t)))\big)^{-1} \cdot \big(((1-\gamma)u_i'(z_i(t)) + \gamma \nabla_i c_i(z(t))) \\ &\quad (x_i - x_i^*) + \gamma \textstyle\sum_{j \neq i} \nabla_j c_i(z(t))(x_j - x_j^*)\big) \end{aligned} \tag{51}$$

Because U_i is strictly concave, $u_i'(x) < 0$ for all x. Hence, under the condition (24), if $x < x^*$, we have $T_i(x) \leq x_i^*$ for all i, from which we immediately have

$$u_i\big(T_i(x)\big) = (1-\gamma)u_i(x_i) + \gamma c_i(x) \geq u_i(x_i^*) = c_i(x^*) > c_i(x),$$

which leads to $u_i(x_i) > c_i(x)$ and thus

$$u_i(x_i) > (1-\gamma)u_i(x_i) + \gamma c_i(x), \quad \forall \gamma > 0. \tag{52}$$

Applying $u_i^{-1}(\cdot)$ to both sides yields $x_i < T_i(x)$ for all i. Therefore, there exists an $\alpha \in [0,1)$, which depends on γ, u_i, c_i and x^*, such that

$$|T_i(x) - x_i^*| \leq \alpha |x_i - x_i^*|, \quad \forall x_i \in [a_i, x_i^*], \ \forall i, \tag{53}$$

or equivalently, $\|T(x) - x^*\|_\infty \leq \alpha \|x - x^*\|_\infty$, for all x between $a = (a_1, \ldots, a_n)$ and x^*. In other words, we have shown that T is a pseudo-contraction in $\prod_i [a_i, x_i^*]$.

Similarly, if $x > x^*$, we can show that $x_i > T_i(x) \geq x_i^*$ for all i, and T is a pseudo-contraction in $\prod_i [x_i^*, b_i]$. Thus, Algorithm 1 converges geometrically. ∎

Proof Sketch of Proposition 4: Note that $T_i(p)$ in (9) can be written in the form $T_i(p) = p_i - \gamma f_i(p)$, where $f_i(p) := y_i(p) - x_i(p)$, with f_i being continuously differentiable. By Proposition 1.11 of [4] (pp. 194), $T_i(p)$ is a contraction if

$$\begin{cases} 0 < \gamma \leq 1/\nabla_i f_i(p), \quad \forall p, \ \forall i, \\ \sum_{j \neq i} |\nabla_j f_i(p)| < \nabla_i f_i(p), \quad \forall p, \ \forall i. \end{cases} \tag{54}$$

We can derive $\nabla_j f_i(p)$ from the Hessian matrix of $C(x)$ and $U(x)$. In particular, we have

$$\begin{aligned} \nabla_i f_i(p) &= \nabla_i y_i(p) - \nabla_i x_i(p) = P_{ii}(x(p)) - \big(U_i''(x_i(p))\big)^{-1} \\ \nabla_j f_i(p) &= \nabla_j y_i(p) - \nabla_j x_i(p) = P_{ij}(x(p)), \quad \forall j \neq i. \end{aligned}$$

Substituting the above into (54) and utilizing the fact $U_i''(x_i) < 0$ will prove the proposition. ∎