Network Optimization with Dynamic Demands and Link Prices

Stacy Patterson, Mike P. Wittie, Kevin Almeroth, and Bassam Bamieh

***Abstract*— We present *Overlapping Cluster Decomposition* (OCD), a novel distributed algorithm for network optimization targeted for networks with dynamic demands and link prices. OCD uses a dual decomposition of the global problem into local optimization problems in each node's neighborhood. The local solutions are then reconciled to find the global optimal solution. While OCD is a descent method and thus may converge slowly in a static network, we show that OCD can more rapidly adapt to changing network conditions than previously proposed first-order and Newton-like network optimization algorithms. Therefore, OCD yields better solutions over time than previously proposed methods at a comparable communication cost.**

I. Introduction

Network optimization involves determining an allocation of resources, most commonly an allocation of data flow to links, that is optimal with respect to some objective function. In this work, we address the single commodity network optimization problem. Namely, there is a single information flow, e.g. a streaming video conference, and different nodes have different supply and demand rates for this flow. Each link has an convex cost function, also called a *price*, and the objective is to assign flow to links so as to minimize the total cost while meeting the supply and demand requirements.

We consider a scenario where link prices and supply and demand rates can change over time, and the magnitudes and timings of these changes are not known a priori. Each node knows the link prices and demands for some local neighborhood for the current time period, or *epoch*, only. Nodes can communicate with neighbors within some small, fixed radius, and each node controls the flow on its outgoing links. The goal is for all nodes to minimize the total cost of the flow allocation over time using only local communication. To achieve this goal, nodes collaborate to converge to the global optimum for the current network conditions, incorporating changes in demands and link prices as soon as they are detected.

We propose a novel distributed algorithm for network optimization called *Overlapping Cluster Decomposition* (OCD) that is targeted for this dynamic setting. OCD enables rapid reallocation of resources in response to changing flow demands and link prices. OCD prioritizes fast convergence of local optimization problems in each node's neighborhood, or *cluster*, that are then reconciled towards the global optimum. As we show in our evaluations, OCD can more readily adapt to fluctuations in demands and link prices than previously proposed distributed optimization algorithms at a comparable communication cost.

Previous works have proposed gradient type methods for both primal and dual versions of the network optimization problem in static networks (e.g. see [1], [2]). While gradient-based approaches are simple to implement in a distributed manner, they can result in slow convergence to the optimal solution [3]. More recently, several distributed Newton-like algorithms for network optimization and the related problem of network utility maximization have been proposed. These works also consider a static network setting. Athuraliya and Low propose a method to approximate the Newton direction using a zeroth order Taylor approximation of the inverse of the Hessian [4]. The recent work by Zargham et al. gives a more general method, Accelerated Dual Descent (ADD), that is based on the h^{th} order Taylor approximation [5]. Wei et al. propose a distributed consensus algorithm to compute the Newton direction in each iteration of the descent algorithm [6], [7]. A similar method based on gossiping and consensus techniques was proposed by Modiano et al. [8]. Finally, Bickson et al. propose a method that uses consensus rounds to compute the Newton step using the Gaussian belief propagation [9].

Several works have addressed the problem of network utility maximization with delivery contracts in dynamic networks. The delivery contracts are specified as constraints over time. Therefore, unlike in our setting, the problem cannot be decoupled into a sequence of optimization problems. Trichakis et al. [10] present an algorithm finds the optimal allocation when information about network changes is known a priori, and they give a heuristic for the setting when this information is not known. They use a first-order method based on dual decomposition. The recent work by Wei et al. [11] presents a faster converging, Newton-based method that solves the same problem when the dynamics are known a priori. This full information assumption is in stark contrast to our assumption that nodes have no knowledge about the network dynamics.

The remainder of this work is organized as follows. In Section II, we formally define the network optimization problem and provide background on previously proposed solutions for static networks. In Section III, we present the Overlapping

This work was funded in part by the Arlene & Arnold Goldstein Center at the Technion Autonomous Systems Program and a Technion fellowship.

S. Patterson is with the Department of Electrical Engineering, Technion - Israel Institute of Technology, Haifa 32000, Israel, stacyp@ee.technion.ac.il

M. Wittie is with the Department of Computer Science, Montana State University, Bozeman, MT 59717, USA, mwittie@cs.montana.edu

K. Almeroth is with the Department of Computer Science, University of California, Santa Barbara, Santa Barbara, CA 93106, USA, almeroth@cs.ucsb.edu

B. Bamieh is with the Department of Mechanical Engineering, University of California, Santa Barbara, Santa Barbara, CA 93106, USA, bamieh@engineering.ucsb.edu

Cluster Decomposition method. Section IV gives some of our evaluation results, highlighting the performance of OCD in dynamic networks. Finally, we conclude in Section V.

II. Problem Formulation and Background

We model the network by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices (also called nodes), with $|\mathcal{V}| = N$ and $\mathcal{E}$ is the set of links, with $|\mathcal{E}| = M$. The network connectivity is captured by an $N \times M$ matrix $A = [a_{ij}]$, where

$$a_{ij} = \begin{cases} 1 & \text{if edge } j \text{ leaves node } i \\ -1 & \text{if edge } j \text{ enters node } i \\ 0 & \text{otherwise.} \end{cases}$$

We divide time into epochs, $t = 1, 2, \ldots$. In each epoch t, a node can be a *source* that injects flow into the network, a *sink* that absorbs flow from the network, or a *peer* that simply allows flow to pass through it. The information about the flow demands is specified by the *demand* vector $b^t \in \mathbb{R}^{\mathbb{N}}$, where each component b_i^t gives the rate of injection or absorption at node i for epoch t. For source nodes, $b_i^t > 0$, for sink nodes, $b_i^t < 0$, and for peers $b_i^t = 0$. To ensure that the network optimization problem is feasible, we require that $\sum_{v \in V} b_i^t = 0$ for all t, meaning that, at all times, all flow injected into the network is also absorbed. In each epoch, every link $e \in \mathcal{E}$ has a strictly convex, twice-differentiable cost function $\phi_e^t(x_e) : \mathbb{R} \to \mathbb{R}$, where x_e denotes the flow along link e. At any time, each node knows only the its own value of b_i^t and the cost functions of its adjacent edges for that epoch. It does not know the duration of the epochs nor its future demand or link cost functions.

A. The Network Optimization Problem

The objective of network optimization is, for each epoch t, to find an allocation of rates to the links that satisfies the injection and demand rates while minimizing the total cost. The optimal allocation for an epoch t is the solution to the following optimization problem,

$$\text{minimize} \qquad f(x) = \sum_{e \in \mathcal{E}} \phi_e^t(x_e) \tag{1}$$

$$\text{subject to} \qquad Ax = b^t \tag{2}$$

The constraints (2) are flow conservation constraints; they require that the flow allocation satisfies the injection and absorption specifications given by the demand vector and that the net flow at each node is 0. We note that the constraints in (2) are not linearly independent. To remedy this, we can simply remove any single equation. For the remainder of this paper, we assume that A and b^t have been adjusted in this respect.

The problem (1-2) involves minimization of a strictly convex objective function subject to linear constraints. Therefore, it has a unique solution, and this solution can be easily obtained in a centralized setting using standard software. However, we wish to solve this problem in-network in a distributed fashion. Ideally, the distributed algorithm should converge to the optimal solution quickly and with as little communication as possible. In the remainder of this section, we describe the dual formulation of the network optimization problem and overview the two classes of previously proposed network optimization algorithms. As these algorithms focus on solving the optimization problem within a single epoch, we omit the superscript t from our notation for clarity.

B. Dual Descent Algorithms

A well-known technique for solving a constrained optimization problem is to instead solve the unconstrained dual problem. We first define the Lagrangian function for (1 - 2),

$$\mathcal{L}(x, \lambda) := \sum_{e \in \mathcal{E}} \phi_e(x_e) + \lambda^T (Ax - b).$$

$\lambda \in \mathbb{R}^N$ are the Lagrange multipliers, one for each constraint in (2). We then define the dual function,

$$\begin{aligned} q(\lambda) &:= \inf_{x \in \mathbb{R}^M} \mathcal{L}(x, \lambda) \\ &= \inf_{x \in \mathbb{R}^{\mathbb{M}}} \left(\sum_{e \in E} \phi_e(x_e) + \lambda^T (Ax) \right) - \lambda^T b \\ &= \sum_{e \in E} \inf_{x \in \mathbb{R}^M} \left(\phi_e(x_e) + \lambda^T (Ax) \right) - \lambda^T b. \end{aligned} \tag{3}$$

Let $\hat{x}(\lambda)$ be the vector that optimizes (3) for a given λ. The components of $x(\lambda)$ are simply,

$$\hat{x}(\lambda)_e = (\phi_e')^{-1}(\lambda_i - \lambda_j), \tag{4}$$

where e corresponds to the edge from node i to node j. Note that each $x(\lambda)_e$ depends only on the i^{th} and j^{th} components of λ. Therefore, its value can be computed based on local information, namely the Lagrange multipliers of the nodes adjacent to edge e.

Finally, the dual problem can be written as an unconstrained convex optimization problem,

$$\text{maximize} \quad q(\lambda). \tag{5}$$

The primal problem in (1-2) is a convex optimization problem. Therefore, the solution to the dual problem (5) has zero duality gap [12], and the flow rates obtained in (4) for the optimal solution to (5) are optimal for the primal problem.

Unconstrained optimization problems of the form of (5) can be solved iteratively using a descent method. Starting with an arbitrary $\lambda(0)$, subsequent values are generated as follows,

$$\lambda(k+1) = \lambda(k) + \alpha(k) d(k), \tag{6}$$

where $\alpha(k)$ is the *step size* and $d(k)$ is the *step direction*. Recent works have proposed distributed descent methods for the dual of the network optimization problem. These methods fall into two main categories, gradient methods and Newton-like methods

1) The Gradient Method: In the gradient method, the step direction is given by the gradient of the objective function at the current value of $\lambda(k)$,

$$d(k) = \nabla q\left(\lambda(k)\right) = Ax(k) - b,$$

where

$$\hat{x}(k) := \arg\min_{x \in \mathbb{R}^N} \mathcal{L}\left(x, \lambda(k)\right).$$

The i^{th} component of $d(k)$ is given by

$$d_i(k) = \sum_{e=(i,j)} \hat{x}_e(k) - \sum_{e=(j,i)} \hat{x}_e(k) - b_i. \tag{7}$$

In the distributed implementation of this gradient descent algorithm, each node i is responsible for its component of $d(k)$. As shown in (7), a node can update its component in a local fashion, using only the components of $\hat{x}(k)$ corresponding to its immediate neighbors. Despite their simplicity and low control messaging overhead in each convergence round, the subgradient methods are known to suffer from slow convergence [3].

2) Newton's Method: An alternative descent method with faster convergence than the gradient method is Newton's method. In Newton's method, the step direction is the direction which minimizes a quadratic approximation of the objective at the last iteration [12]. For the dual problem (5), the Newton step direction in iteration k is

$$d(k) = -\left(\nabla^2 q\left(\lambda(k)\right)\right)^{-1} \nabla q\left(\lambda(k)\right),$$

where $\nabla^2 q(\lambda(k))$ is the Hessian matrix of the dual function q at $\lambda(k)$.

To compute the exact Newton step direction, every node would require knowledge of all dual variables. Several methods (with similar communication complexity) have been proposed to approximate the Newton step direction using only local information [4], [5], [7], [6], [8]. As a representative example, we consider the recently proposed Accelerated Dual Descent method (ADD), which has been shown to converge faster than consensus-based approximate Newton's methods [5]. In ADD, in every iteration, each node computes its step direction using the h^{th} order Taylor approximation of the inverse of the Hessian. This computation requires the node to communicate with all nodes within an $(h+1)$-hop radius.

While the approximate-Newton's methods offer a marked improvement in convergence rate over gradient methods, in Section IV, we show that they do not adjust quickly to network changes. In contrast, our Overlapping Cluster Decomposition method converges more slowly than ADD and other approximate Newton methods when the network is static but is able to maintain near optimal cost when the network changes. This benefit is achieved without any increase in communication cost. We describe the OCD method in the next section.

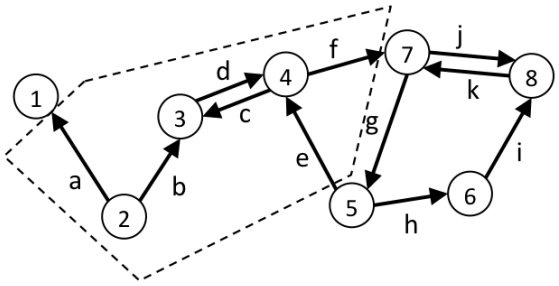


Fig. 1: A sample network. The 2-cluster for node 3 is indicated by the dashed lines. It contains nodes 2, 3, and 4 and edges a, b, c, d, e, and f. Note that nodes 1, 5, and 7 and edge g are not included in the cluster.

III. Overlapping Cluster Decomposition Method

The motivating intuition behind OCDis that a change in a network, for example the cost function of a link, affects resource allocation in a small neighborhood of that link to a greater degree than the network as a whole. Therefore, if each node solves a local optimization problem over its neighborhood, it can maintain a global cost that is close to optimal by making adjustments only within its neighborhood. We describe the decomposition of the global optimization problem into local optimization problems below.

A. Overlapping Cluster Dual Decomposition

In order to formally state the local optimization problems, we must first define what is meant by local information. We define the h-cluster of node v as the subgraph $\mathcal{C}^v := (\mathcal{V}^v, \mathcal{E}^v)$, where $\mathcal{V}^v$ is the set of nodes that are strictly less than h hops away from v, including v, and $\mathcal{E}^v$ is the set of links adjacent to the nodes in $\mathcal{V}^v$, irrespective of link direction. An example is shown in Figure 1, where the 2-cluster for node 3 is indicated by the dashed lines. We define A^v to be the matrix formed from the rows and columns of A corresponding to the nodes and links in $\mathcal{C}^v$. b^v is the vector formed from the components of b corresponding to the nodes in $\mathcal{C}^v$. A possible local optimization problem is then,

$$\text{minimize} \sum_{e \in \mathcal{E}^v} \phi_e(y_e^v) \quad \text{subject to } A^v y^v = b^v. \tag{8}$$

Here, y^v is the flow vector for the cluster of node v, each component corresponding to the flow along an edge in $\mathcal{C}^v$.

If nodes solve independent optimization problems like the one above, they may arrive at different values for the flow along edges common to multiple clusters. Therefore, it is necessary to reconcile these local solutions so that nodes agree on the flow rate for each edge. We can express this requirement through the following constraints,

$$y_e^v = x_e \quad \text{for } v \in \mathcal{V}, \text{for } e \in \mathcal{E}^v, \tag{9}$$

where y_e^v denotes the component of v's flow rate vector corresponding to edge e. These constraints are the *coupling constraints*.

With some minor adjustments, we can combine (8) and (9) to form an optimization problem equivalent to the original optimization problem (1 - 2). We call this form the Overlapping Cluster Decomposition. We then apply the framework for dual decomposition described by Samar et al. for convex optimization problems with a separable objective and coupling constraints [13] to derive a distributed algorithm that solves the optimization problem.

We first rewrite the coupling constraints (9) as follows,

$$E^v x = y^v \quad \text{for } v \in \mathcal{V},$$

where each $E^v = [E^v_{ij}]$ is the $|\mathcal{V}^v| \times M$ matrix with entries given by,

$$E^v_{ij} = \begin{cases} 1 & \text{if constraint } y^v_i = x_j \text{ exists} \\ 0 & \text{otherwise.} \end{cases}$$

The matrix E is formed by stacking the E^v matrices, one on top of the next. The vector y is formed from the y^v vectors in the same manner. The constraint (9) is equivalent to

$$Ex = y.$$

The network optimization problem can then be expressed in the following, decomposable form,

$$\text{minimize} \quad \sum_{v\in\mathcal{V}} \sum_{e\in\mathcal{E}^v} \frac{1}{\#_e} \phi_e(y^v_e) \tag{10}$$

$$\text{subject to} \quad A^v y^v = b^v \quad \text{for all } v \in \mathcal{V} \tag{11}$$

$$Ex = y \tag{12}$$

$$y^v_e \geq 0 \quad \text{for } v \in \mathcal{V},\ e \in \mathcal{E}^v. \tag{13}$$

Here, $\#_e$ is the number of clusters in which edge e appears. We assume that each node knows the network topology a priori and therefore can independently determine $\#_e$ for each edge in its cluster.

The objective in (10) is equivalent to the objective (1). The constraints in (11) and (12) together are equivalent to the constraint in (2). The constraints (13) add an additional requirement that all flow rates are non-negative, thus respecting the directionality of the links. These constraints are optional, and we note that they do not appear in the formulation of the problem and algorithm in [5]. Even with the addition of the non-negativity constraints, the problem is still a convex optimization problem and therefore has a unique solution. It is also straightforward to include link capacity constraints that bound the minimum and maximum flow that can be assigned to each link, though we omit this extension for brevity.

The next step is to form the partial Lagrangian with Lagrange multipliers for the coupling constraints,

$$\begin{aligned} L(x,y,\gamma) &= \sum_{v\in\mathcal{V}} \sum_{e\in\mathcal{E}^v} \frac{1}{\#_e} \phi_e(y^v_e) + \gamma^T (Ex - y) \\ &= \sum_{v\in\mathcal{V}} \left(\sum_{e\in\mathcal{E}^v} \frac{1}{\#_e} \phi_e(y^v_e) - (\gamma^v)^T y^v \right) \\ &\quad + \gamma^T E x. \end{aligned}$$

Each γ^v is the sub-vector of γ corresponding to node v; each component of γ^v, denoted γ^v_e, corresponds to an edge in node v's cluster.

To find the dual problem, we minimize over x and y separately. The minimum over x occurs where $E^T\gamma = 0$. To minimize over y, we define the subproblems $q^v(\gamma^v)$, one per node,

$$\text{minimize} \quad \sum_{e\in\mathcal{E}_v} \frac{1}{\#_e} \phi_e(y^v_e) - (\gamma^v)^T y^v \tag{14}$$

$$\text{subject to} \quad A^v y^v = b^v \tag{15}$$

$$y^v_e \geq 0 \quad \text{for } e \in \mathcal{E}^v. \tag{16}$$

Each subproblem is a convex optimization problem, and each node can solve its subproblem locally using any fast convex optimization method.

Finally, we arrive at the dual problem,

$$\text{maximize} \quad q(\gamma) = \sum_{v\in\mathcal{V}} q^v(\gamma^v) \tag{17}$$

$$\text{subject to} \quad E^T\gamma = 0 \tag{18}$$

In OCD, the dual problem (17 - 18), is solved using a distributed projected subgradient method. Each node v communicates with other nodes in overlapping clusters to learn the pricing functions ϕ_e for $e \in \mathcal{E}^v$. The Lagrange multipliers are initialized to $\gamma(0) = 0$. In each iteration k, each node solves its optimization problem $q^v(\gamma(k))$ over its cluster and communicates with nodes in overlapping clusters to update γ, and then the process is repeated. In a static network, with the proper choice of step-size, the algorithm is guaranteed to converge to the optimal solution. The solution to the dual problem in OCD has zero duality gap (see [12]). Thus, as γ converges to the optimum, y also converges to the optimal resource allocation.

In the remainder of this section, we explain the details of the projected subgradient algorithm and describe how it can be performed with local communication.

B. Distributed Projected Subgradient Algorithm

In the projected subgradient method, γ is updated according to the following equation,

$$\gamma(k+1) = \mathcal{P}\left(\gamma(k) + \alpha(k) d(k)\right).$$

As in the descent methods presented in Section II, $\alpha(k)$ is the step size. The step direction $d(k)$ can be any subgradient of q at $\gamma(k)$. We use $-\hat{y}$, the vector that optimizes the subproblems q^v. $\mathcal{P}(\cdot)$ is the Euclidean projection operator,

$$\mathcal{P}(z) = \left(I - E(E^TE)^{-1}E^T\right) z.$$

Therefore, the update step is

$$\gamma(k+1) = \gamma(k) + \alpha(k) \left(I - E(E^TE)^{-1}E^T\right) \hat{y}. \tag{19}$$

The operator $(E^TE)^{-1}E^T$ computes average over the $\hat{y}^v$ vectors, one average for each edge. Each component of the step direction is thus equivalent to the difference between $\hat{y}^v_e$ and the average of $\hat{y}^w_e$ for all nodes w with $e \in \mathcal{E}^w$.

Algorithm 1: OCD with Distributed Projected Subgradient Method, as performed at each node v.

init: $k := 0, \quad \gamma^v(0) := 0$
while *true* **do**
  Solve $q^v(\gamma^v(k))$. ;
  $\hat{y}^v := \arg\min q^v(\gamma^v(k))$;
  foreach $e \in \mathcal{E}^v$ **do**
    // Choose sending rate for this round. ;
    if v *is sender for link* e **then** $x_e := \hat{y}_e^v$;
    // Update dual variables. ;
    $z_e :=$ average($\hat{y}_e^w$), over all w with $e \in \mathcal{E}^w$;
    $\gamma_e^v(k+1) := \gamma_e^v(k) + \alpha(k)(-\hat{y}_e^v + z_e)$;
  end
  $k := k + 1$;
end

The distributed implementation of (19) is given in Algorithm 1. Each node v maintains a vector of Lagrange multipliers γ^v, each one corresponding to an edge in the cluster of v. In each iteration, the nodes independently solve the optimization subproblems q^v. While each edge belongs to multiple clusters, we must assign one flow rate to the physical link. We use the value that is computed at the sending node for that link. Nodes communicate to find the averages of their optimal $\hat{y}$ for each edge. Each node then updates its γ^v using these averages. To find the averages, each node much communicate with nodes in overlapping clusters. For cluster radius h, OCD requires that, in each iteration, each node communicate with all nodes within a radius of $2h-1$ hops.

In general, first order methods like OCD converge more slowly than Newton-type methods. However, our cluster-based decomposition makes it possible for local adjustments to be made within a single round. Therefore, as the network changes, OCD can quickly reallocate resources to achieve near-optimal cost. In subsequent rounds, if the network remains static, the OCD algorithm converges from this near optimal solution to the optimal flow rate assignment.

C. Accommodating Link and Demand Changes

When the link cost functions or demands change for a links and nodes in a node's cluster, the node learns the new cost functions in the first iteration after the change when it communicates with nodes in overlapping clusters. It continues to execute Algorithm 1 using these updated cost functions and demands and the same values for the Lagrange multipliers γ^v as in the previous iteration. If the cost functions and demands remain fixed for "long enough", these Lagrange multipliers will converge to optimal values, thus yielding the optimal allocation for the current network conditions.

Note: The optimization methods described in Sections II and III are descent methods for the dual problem. Therefore, until an optimal solution is achieved, the sending rates will violate the flow conservation constraints (2). If the total injection rate of a flow allocation exceeds the total absorption rate, the excess flow will be queued in buffers at nodes in the network, and buffers that continue to grow result in increasing end-to-end delays for packet deliveries between the sources and sinks. In practice, this can be dealt with by artificially limiting the network capacity and solving the network optimization problem in the limited network. Thus, the overallocation in the artificial network does not lead to overallocation in the actual network. However, this artificial limitation decreases the overall capacity of the network, and so, it is desirable that a network optimization algorithm achieves a solution that satisfies the conservations constraints as quickly as possible without artificially limiting the link capacities.

IV. Numerical Results

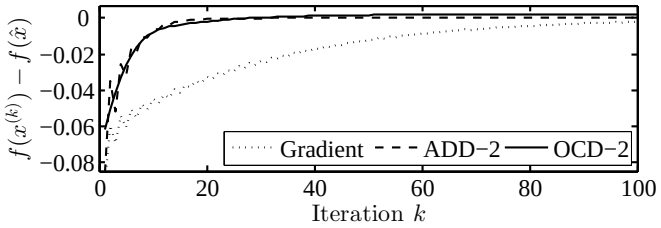
Our goal is to evaluate OCD's effectiveness at solving the minimum cost forwarding problem in dynamic networks. We demonstrate that OCD's emphasis on rapid local convergence leads to faster adjustments of global resource allocation than those achieved by the gradient and approximate Newton methods. These results show that methods like OCD based on fast local convergence and global reconciliation are an attractive approach to resource management in dynamic network settings.

We have implemented the gradient, ADD, and OCD methods using Matlab and `CVX` [14]. The convergence rate of the optimization method depends, in part, on the step size along their descent directions. Gradient and subgradient methods, including OCD, converge to within a neighborhood of the optimum if the step size is sufficiently small [1]. For both the gradient method and ADD, we use the optimal step size in each iteration. Note that, in both cases, computation of this optimal step size would require global information, and so a distributed implementation would typically rely on an approximated step size. One would therefore expect that, in practice, the convergence behavior of ADD and the gradient method would be worse than shown in our evaluations. We use a constant step size of two for the OCD method.

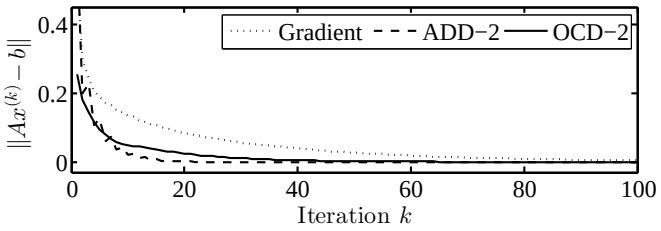
We present results of evaluations on two types of networks. The first is a 40 node unit disk graph, which is generated by placing the nodes uniformly at random in a unit square and connecting every pair of nodes within a Cartesian distance of 0.25 of each other with a bidirectional link. The second is a 40 node Barabási-Albert (BA) scale free graph [15], again with bidirectional links. For both network types, we restrict our experiments to graphs that are strongly connected and have diameter of twelve.

In each simulation, six nodes are selected at random to be sources and six are selected to be sinks. Sink and source demands are chosen uniformly from the interval $[0.5, 1.5]$ ($[-1.5, -0.5]$ for sinks), normalized so as to ensure the problem is feasible. The cost function of each link e is of the form $k_e x_e^2$, where k_e is chosen uniformly at random from the interval [0.5,1.5].

For each setting, we give results from a single network example. We have repeated these experiments on a large

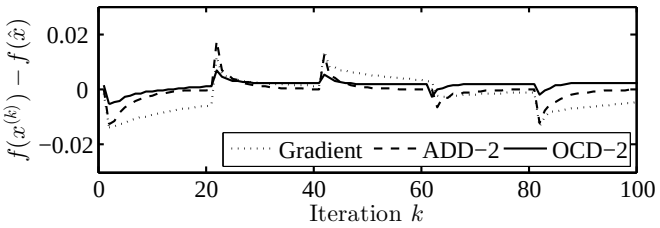


(a) Evolution of deviation of the total cost of flow allocation $f(x(k))$ from the optimal cost $f(\hat{x})$.

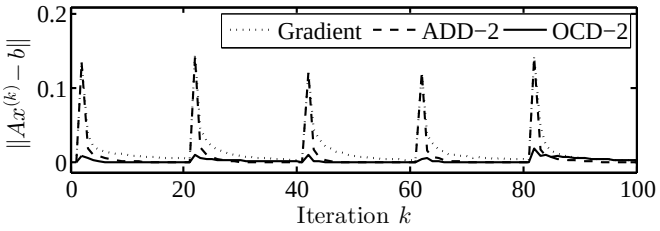


(b) Evolution of constraints violation.

Fig. 2: Comparison of network optimization algorithms in a 40 node, unit disk graph with fixed demands and link cost functions.



(a) Evolution of deviation of the total cost of flow allocation $f(x(k))$ from the optimal cost $f(\hat{x})$.



(b) Evolution of constraints violation. For the gradient method, the total of the constraints violation $\|Ax(k) - b\|$ over 100 iterations is 1.7905. For ADD, it is 1.0191, and for OCD it is 0.2120.

Fig. 3: Comparison of network optimization algorithms in a 40 node, unit disk graph with fixed demands, where link prices change in every 20 iterations.

number of network instances, with different link cost functions and demand distributions and have observed similar results.

A. Performance in Static Networks

We first present a comparison of the gradient, ADD, and OCD algorithms in a network with static link prices and demands. For ADD, we set $h = 2$, denoted ADD-2. This configuration means that, for each iteration, nodes must communicate with all other nodes within a three-hop neighborhood in each iteration. We compare ADD-2 with OCD-2, which also requires communication within a three-hop neighborhood. The gradient method only requires communication with one-hop neighbors. We initialize λ and γ to 0.

The results for the unit disk graph are given in Figure 2. Similar results were observed for the BA graph. Figure 2a shows the total cost of the solution generated by each algorithm for each iteration as the algorithms converge to the optimum, and Figure 2b shows the evolution of the violation of the flow constraints. Since these methods solve the dual problem, they begin from an infeasible solution, with a total allocation cost that is lower than optimal. As the methods progress, the constraints violations are reduced and the total cost is increased. As expected, the Newton-based ADD method demonstrates the fastest convergence behavior. We note that even though OCD is a gradient-based scheme, it outperforms the standard gradient method. In addition, by optimizing within clusters, OCD achieves significantly better results in the first few iterations than both ADD and the gradient-based method.

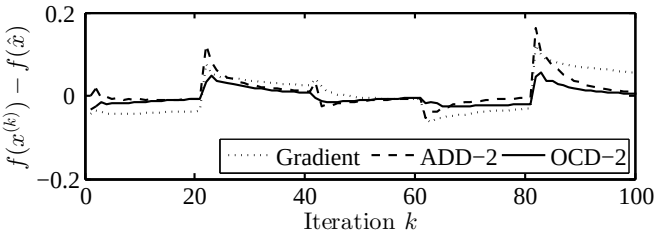
B. Performance with Dynamic Link Costs

We compare the abilities of the gradient method, ADD-2 and OCD-2 to adapt to changing link cost functions. We first let the algorithms converge to the optimal solution for the initial network configuration. We then alter the constant k_e in each link cost function every 20 iterations by choosing a new value uniformly at random from the interval $[0.5, 1.5]$.

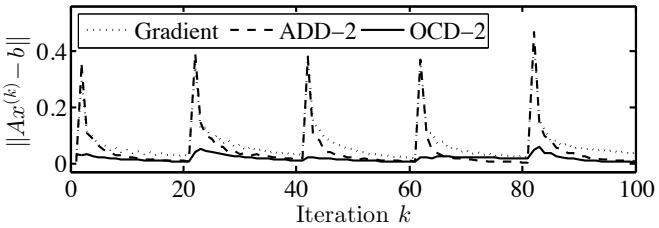
In Figure 3, we show the results for the unit disk graph and in Figure 4 we show results for the BA graph. Each figure shows 100 iterations after the convergence to the initial solution. Both the gradient method and ADD are significantly disturbed by link cost perturbations as shown by the large spikes in the deviation from the optimal cost (Figures 3a and 4a) and in flow constraints violation (Figures 3b and 4b). OCD is less affected by the changes in link costs. In OCD the decomposition enables the nodes to adjust to changes within their clusters within a single iteration. Therefore, OCD achieves a great reduction in the level of constraints violation almost immediately, while the gradient method and ADD require multiple iterations to a achieve similar reduction. Thus the overall performance of OCD is better than that of ADD in spite of OCD's slower convergence rate.

C. Performance with Dynamic Demands

We now present results on the performance of the algorithms in networks with changing source and sink demands. We run the algorithms in a 40 node BA graph with a random initial selection of six sources and six sinks until all solutions converge to the optimal. We then change the locations and magnitudes of the sources and sinks every 20 iterations. The results for 100 iterations after the initial convergence are shown in Figure 5.

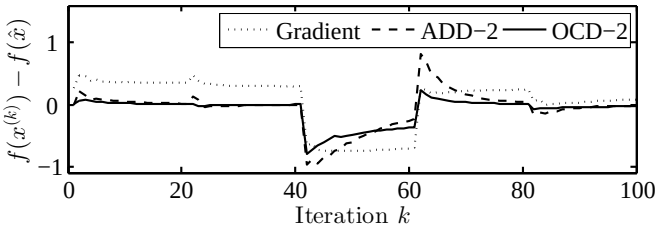


(a) Evolution of deviation of the total cost of flow allocation $f(x(k))$ from the optimal cost $f(\hat{x})$.

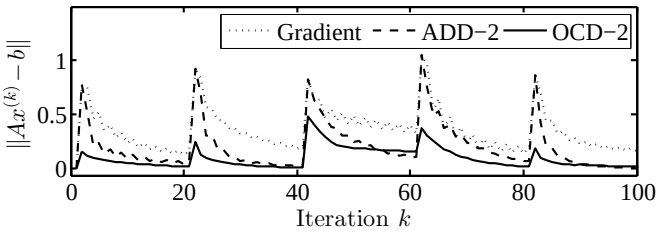


(b) Evolution of constraints violation. For the gradient method, the total of the constraints violation $\|Ax(k) - b\|$ over 100 iterations is 7.0228. For ADD, it is 4.5193, and for OCD it is 2.0954.

Fig. 4: Comparison of network optimization algorithms in a 40 node, BA random graph with fixed demands, where link prices change in every 20 iterations.



(a) Evolution of deviation of the total cost of flow allocation from the optimal cost.



(b) For the gradient method, the total of the constraints violation $\|Ax(k) - b\|$ over 100 iterations is 38.6967. For ADD, it is 26.2122, and for OCD it is 13.7524.

Fig. 5: Comparison of network optimization algorithms in a 40 node, BA graph with fixed link prices where source and sink locations and magnitudes change in every 20 iterations.

For all three methods, the deviation from the optimal cost and the constraints violation are significantly larger than for the dynamic link prices setting. The gradient method shows a large deviation from the optimal cost resource allocation as it changes over time (Figure 5a). ADD produces a better cost allocation, but it is still outperformed by our OCD method. Also as before, there is a spike in the constraints violation for all three methods when demands change (Figure 5b). The spike for OCD is smallest and shows that OCD arrives at a solution that is closer to feasible than both the gradient method and ADD in just a few iterations. We have observed similar trends for the setting where source and sink locations remain fixed and only the demand magnitudes change, but in this setting, the observed cost deviations and constraints violations are slightly smaller.

D. Effect of Communication Range

Finally, we explore the effect of different communication ranges on the convergence behavior of ADD and OCD. Table I shows results for the ADD and OCD methods in a 40 node BA graph where the algorithms are configured to communicate with the same number of nodes per round. We first let the network converge from its initial configuration until the total constraint violation $\|Ax(k) - b\|$ is less than 0.001. We then change the link cost functions change every 10 iterations (as described in Section IV-A). The second column gives the radius of the communication neighborhood for each algorithm. Columns three and four show the total flow constraints violation and total difference of the forwarding cost from the optimum over 200 iterations after the initial convergence. The table shows that the total flow constraints violation is significantly lower for OCD for all the listed communication ranges. OCD also has a smaller value for the sum of the absolute value of the deviation of the cost from the optimal over time for all but the smallest communication range. We have observed similar trends with different graph configurations and in the dynamic demand setting. We note that a smaller cost deviation total does not necessarily mean that the total cost is lower. In addition, if the cost for an iteration is lower than the optimum, the allocation for that iteration is an overallocation, and excess flow is being stored in buffers in the network. As stated in Section III, the increasing buffer size is detrimental to network capacity. Therefore, it is important that both the deviation from the optimal cost and the magnitude of the constraints violation be small.

TABLE I: Effect of communication distance on constraints violation and allocation cost for a 40 node BA graph where link prices change every 10 iterations. Columns three and four give the totals over 200 iterations.

	Comm. Radius	Total $\|Ax(k) - b\|$	Total $\|f(x(k)) - f(\hat{x})\|$
OCD-2	3	3.24	0.56
ADD-2	3	9.90	0.26
OCD-3	5	1.92	0.58
ADD-4	5	10.30	0.60
OCD-4	7	0.80	0.25
ADD-6	7	9.24	1.74
OCD-5	9	0.04	0.02
ADD-8	9	7.82	0.08

The results is this section demonstrate that OCD is more effective than both the gradient method and ADD in networks with changing link prices and source and sink demands.

The fast re-convergence of OCD in the initial iterations after a change in link prices or demands leads to lower aggregate allocation cost and constraints violation, in spite of the lower convergence rate observed in the static network scenario. This is an important observation because it shows that the emphasis on local convergence and approximate global reconciliation has the potential to achieve better global performance in dynamic networks than methods that approximate the global descent direction from local information.

V. Conclusion

In this paper we have presented OCD, a novel method for network optimization. Instead of solving the global optimization problem through local exchanges, OCD uses a dual decomposition of the global problem into local optimization problems in each node's neighborhood. As shown in our evaluations, in networks where link prices and supply and demand fluctuate, OCD can adapt more rapidly than previously proposed distributed optimization algorithms at a comparable communication cost. In future work, we plan to explore the extension of our decomposition approach to the multi-commodity network optimization problem and to settings where some properties of the network dynamics are known in advance.

References

[1] X. Lin and N. B. Shroff, "Joint rate control and scheduling in multihop wireless networks," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, 2004, pp. 1484–1489.

[2] A. Eryilmaz and R. Srikant, "Joint congestion control, routing, and MAC for stability and fairness in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1514–1524, August 2006.

[3] A. Nedić and A. Ozdaglar, "Approximate primal solutions and rate analysis for dual subgradient methods," *SIAM Journal on Optimization*, vol. 19, no. 4, pp. 1757–1780, February 2009.

[4] S. Athuraliya and S. Low, "Optimization flow control with newton-like algorithm," *Journal of Telecommunication Systems*, vol. 15, pp. 345–358, 2000.

[5] M. Zargham, A. O. Alejandro Ribeiro and, and A. Jadbabaie, "Accelerated dual descent for network optimization," in *Proceedings of the American Control Conference*, 2011, pp. 2663–2668.

[6] E. Wei, A. E. Ozdaglar, and A. Jadbabaie, "A distributed newton method for network utility maximization," in *Proceedings of the 49th IEEE Conference on Decision and Control*, 2010, pp. 1816–1821.

[7] ——, "A distributed newton method for network utility maximization," LIDS, Tech. Rep. 2832, 2010.

[8] E. Modiano, D. Shah, and G. Zussman, "Maximizing throughput in wireless networks via gossiping," *SIGMETRICS Performance Evaluation Review*, vol. 34, no. 1, pp. 27–38, June 2006.

[9] D. Bickson, Y. Tock, A. Zyrnnis, S. P. Boyd, and D. Dolev, "Distributed large scale network utility maximization," in *Proceedings of the 2009 IEEE international conference on Symposium on Information Theory*, 2009, pp. 829–833.

[10] N. Trichakis, A. Zymnis, and S. Boyd, "Dynamic network utility maximization with delivery contracts," in *Proceedings of the IFAC World Congress*, 2008, pp. 2907–2912.

[11] E. Wei, A. E. Ozdaglar, A. Erylimaz, and A. Jadbabaie, "A distributed newton method for dynamic network utility maximization with delivery contracts," in *Proceedings of the 46th Annual Conference on Information Sciences and Systems*, 2012.

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

[13] S. Samar, S. Boyd, and D. Gorinevsky, "Distributed estimation via dual decomposition," in *Proceedings of the European Control Conference*, 2007, pp. 1511–1516.

[14] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21," http://cvxr.com/cvx, April 2011.

[15] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, pp. 47–97, 2002.