Notes on Decomposition Methods

Stephen Boyd, Lin Xiao, Almir Mutapcic, and Jacob Mattingley
Notes for EE364B, Stanford University, Winter 2006-07

April 13, 2008

Contents

Decomposition is a general approach to solving a problem by breaking it up into smaller ones and solving each of the smaller ones separately, either in parallel or sequentially. (When it is done sequentially, the advantage comes from the fact that problem complexity grows more than linearly.)

Problems for which decomposition works in one step are called (block) *separable*, or *trivially parallelizable*. As a general example of such a problem, suppose the variable x can be partitioned into subvectors $x_1, \ldots, x_k$, the objective is a sum of functions of x_i, and each constraint involves only variables from one of the subvectors x_i. Then evidently we can solve each problem involving x_i separately (and in parallel), and then re-assemble the solution x. Of course this is a trivial, and not too interesting, case.

A more interesting situation occurs when there is some coupling or interaction between the subvectors, so the problems cannot be solved independently. For these cases there are techniques that solve the overall problem by iteratively solving a sequence of smaller problems. There are many ways to do this; in this note we consider some simple examples that illustrate the ideas.

Decomposition in optimization is an old idea, and appears in early work on large-scale LPs from the 1960s [DW60]. A good reference on decomposition methods is chapter 6 of Bertsekas [Ber99]. Some recent reference on decomposition applied to networking problems are Kelly et al [KMT97] and Chiang et al [CLCD07].

The idea of decomposition comes up in the context of solving linear equations, but goes by other names such as block elimination, Schur complement methods, or (for special cases) matrix inversion lemma (see [BV04, App. C]). The core idea, *i.e.*, using efficient methods to solve subproblems, and combining the results in such a way as to solve the larger problem, is the same, but the techniques are a bit different.

The original primary motivation for decomposition methods was to solve very large problems that were beyond the reach of standard techniques, possibly using multiple processors. This remains a good reason to use decomposition methods for some problems. But other reasons are emerging as equally (or more) important. In many cases decomposition methods yield decentralized solution methods. Even if the resulting algorithms are slower (in some cases, much slower) than centralized methods, decentralized solutions might be prefered for other reasons. For example, decentralized solution methods can often be translated into, or interpreted as, simple protocols that allow a collection of subsystems to coordinate their actions to achieve global optimality.

In §1, we describe the simplest decomposition method, which is called primal decomposition since the subsystems coordinate to choose the values of some primal variables. In §2 we describe dual decomposition, in which dual variables (prices) are manipulated to solve the global problem. We explore general decomposition structures, and the associated decomposition methods, in §4. In §5 and §6 we consider two more specific examples in more detail.

1 Primal decomposition

We'll consider the simplest possible case, an unconstrained optimization problem that splits into two subproblems. (But note that the most impressive applications of decomposition occur when the problem is split into many subproblems.) In our first example, we consider an unconstrained minimization problem, of the form

$$\text{minimize} \quad f(x) = f_1(x_1, y) + f_2(x_2, y) \tag{1}$$

where the variable is $x = (x_1, x_2, y)$. Although the dimensions don't matter here, it's useful to think of x_1 and x_2 as having relatively high dimension, and y having relatively small dimension. The objective is almost block separable in x_1 and x_2; indeed, if we fix the subvector y, the problem becomes separable in x_1 and x_2, and therefore can be solved by solving the two subproblems independently. For this reason, y is called the *complicating variable*, because when it is fixed, the problem splits or decomposes. In other words, the variable y complicates the problem. It is the variable that couples the two subproblems. We can think of x_1 (x_2) as the *private variable* or *local variable* associated with the first (second) subproblem, and y as the *public variable* or *interface variable* or *boundary variable* between the two subproblems.

The observation that the problem becomes separable when y is fixed suggests a method for solving the problem (1). Let $\phi_1(y)$ denote the optimal value of the problem

$$\text{minimize}_{x_1} \quad f_1(x_1, y), \tag{2}$$

and similarly, let $\phi_2(y)$ denote the optimal value of the problem

$$\text{minimize}_{x_2} \quad f_2(x_2, y). \tag{3}$$

(Note that if f_1 and f_2 are convex, so are ϕ_1 and ϕ_2.) We refer to (2) as *subproblem 1*, and (3) as *subproblem 2*.

Then the original problem (1) is equivalent to the problem

$$\text{minimize}_y \quad \phi_1(y) + \phi_2(y). \tag{4}$$

This problem is called the *master problem*. If the original problem is convex, so is the master problem. The variables of the master problem are the complicating or coupling variables of the original problem. The objective of the master problem is the sum of the optimal values of the subproblems.

A *decomposition method* solves the problem (1) by solving the master problem, using an iterative method such as the subgradient method. Each iteration requires solving the two subproblems in order to evaluate $\phi_1(y)$ and $\phi_2(y)$ and their gradients or subgradients. This can be done in parallel, but even if it is done sequentially, there will be substantial savings if the computational complexity of the problems grows more than linearly with problem size.

Let's see how to evaluate a subgradient of ϕ_1 at y, assuming the problem is convex. We first solve the associated subproblem, *i.e.*, we find $\bar{x}_1(y)$ that minimizes $f_1(x_1, y)$. Thus,

there is a subgradient of f_1 of the form $(0, g_1)$, and not surprisingly, g_1 is a subgradient of ϕ_1 at y. We can carry out the same procedure to find a subgradient $g_2 \in \partial\phi_2(y)$ Then $g_1 + g_2$ is a subgradient of $\phi_1 + \phi_2$ at y.

We can solve the master problem by a variety of methods, including bisection (if the dimension of y is one), gradient or quasi-Newton methods (if the functions are differentiable), or subgradient, cutting-plane, or ellipsoid methods (if the functions are nondifferentiable). This basic decomposition method is called *primal decomposition* because the master algorithm manipulates (some of the) primal variables.

When we use a subgradient method to solve the master problem, we get a very simple primal decomposition algorithm.

repeat

Solve the subproblems (possibly in parallel).

Find $\bar{x}_1$ that minimizes $f_1(x_1, y)$, and a subgradient $g_1 \in \partial\phi_1(y)$.

Find $\bar{x}_2$ that minimizes $f_2(x_2, y)$, and a subgradient $g_2 \in \partial\phi_2(y)$.

Update complicating variable.

$y := y - \alpha_k(g_1 + g_2)$.

Here α_k is a step length that can be chosen in any of the standard ways.

We can interpret this decomposition method as follows. We have two subproblems, with *private variables* or *local variables* x_1 and x_2, respectively. We also have the complicating variable y which appears in both subproblems. At each step of the master algorithm the complicating variable is fixed, which allows the two subproblems to be solved independently. From the two local solutions, we construct a subgradient for the master problem, and using this, we update the complicating variable. Then we repeat the process.

When a subgradient method is used for the master problem, and ϕ_1 and ϕ_2 are differentiable, the update has a very simple interpretation. We interpret g_1 and g_2 as the gradients of the optimal value of the subproblems, with respect to the complicating variable y. The update simply moves the complicating variable in a direction of improvement of the overall objective.

The primal decomposition method works well when there are few complicating variables, and we have some good or fast methods for solving the subproblems. For example, if one of the subproblems is quadratic, we can solve it analytically; in this case the optimal value is also quadratic, and given by a Schur complement of the local quadratic cost function. (But this trick is so simple that most people would not call it decomposition.)

The basic primal decomposition method described above can be extended in several ways. We can add separable constraints, *i.e.*, constraints of the form $x_1 \in \mathcal{C}_1$, $x_2 \in \mathcal{C}_2$. In this case (and also, in the case when $\mathbf{dom}\, f_i$ is not all vectors) we have the possibility that $\phi_i(y) = \infty$ (*i.e.*, $y \notin \mathbf{dom}\, \phi$) for some choices of y. In this case we find a cutting-plane that separates y from $\mathbf{dom}\, \phi$, to use in the master algorithm.

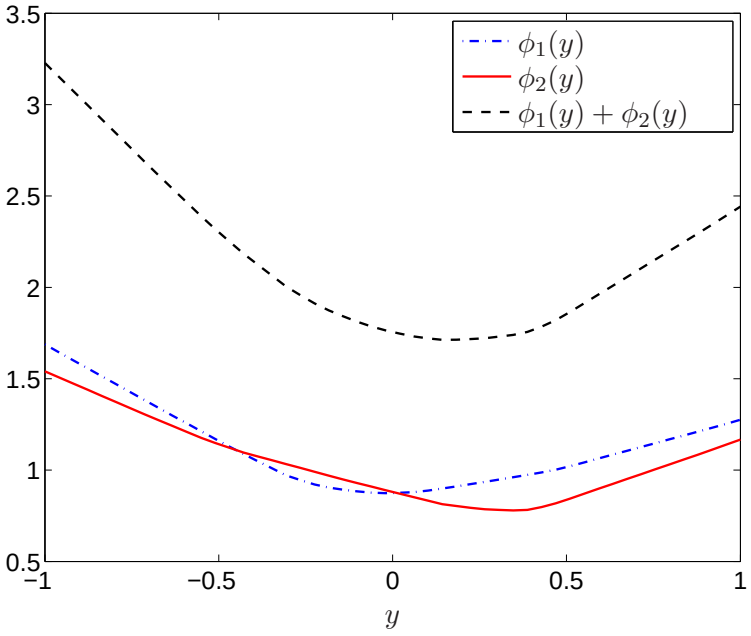

Figure 1: Objective function of master problem (*i.e.*, $\phi_1 + \phi_2$) and decomposed components (ϕ_1 and ϕ_2) as a function of y.

1.1 Simple example

We illustrate primal decomposition with a simple example, with a single (scalar) complicating variable. The problem has the form (1), where f_1 and f_2 are piecewise-linear convex functions of x_1 and y, and x_2 and y, respectively. For the particular problem instance we consider, $x_1 \in \mathbf{R}^{20}$, $x_2 \in \mathbf{R}^{20}$, and f_1 and f_2 are each the maximum of 100 affine functions. Since the complicating variable y is scalar, we can optimize over y using a bisection algorithm.

Figure 1 shows ϕ_1, ϕ_2, and $\phi_1 + \phi_2$ as a function of y. The optimal value of the problem is $p^\star \approx 1.71$, achieved for $y^\star \approx 0.14$. Figure 2 shows the progress of a bisection method for minimizing $\phi_1(y) + \phi_2(y)$, with initial interval $[-1, 1]$. At each step, the two subproblems are solved separately, using the current value of y.

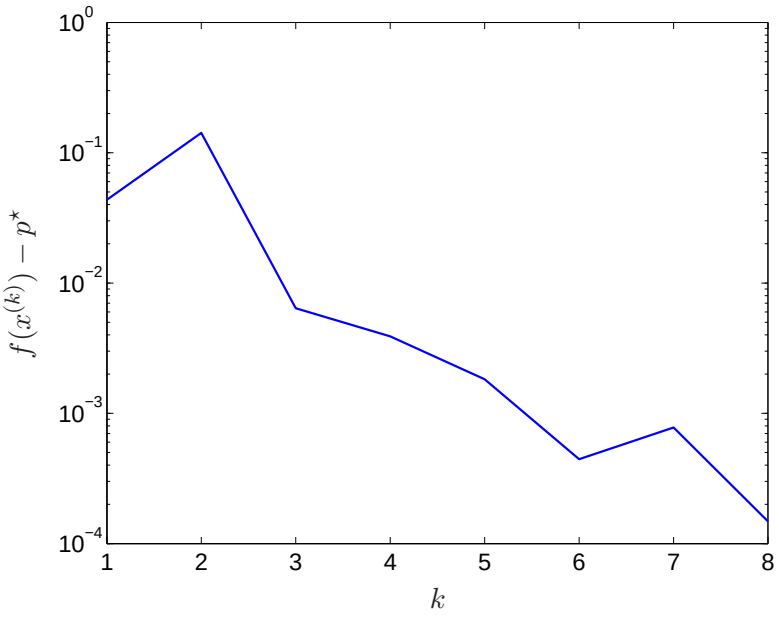

Figure 2: Suboptimality versus iteration number k for primal decomposition, with master problem solved by bisection.

2 Dual decomposition

We can apply decomposition to the problem (1) after introducing some new variables, and working with the dual problem. We first express the problem as

$$\begin{array}{ll} \text{minimize} & f(x) = f_1(x_1, y_1) + f_2(x_2, y_2) \\ \text{subject to} & y_1 = y_2, \end{array} \tag{5}$$

by introducing a new variable and equality constraint. We have introduced a *local version* of the complicating variable y, along with a *consistency constraint* that requires the two local versions to be equal. Note that the objective is now separable, with the variable partition (x_1, y_1) and (x_2, y_2).

Now we form the dual problem. The Lagrangian is

$$L(x_1, y_1, x_2, y_2, \nu) = f_1(x_1, y_1) + f_2(x_2, y_2) + \nu^T y_1 - \nu^T y_2,$$

which is separable. The dual function is

$$g(\nu) = g_1(\nu) + g_2(\nu),$$

where

$$g_1(\nu) = \inf_{x_1, y_1} \left(f_1(x_1, y_1) + \nu^T y_1 \right), \qquad g_2(\nu) = \inf_{x_2, y_2} \left(f_2(x_2, y_2) - \nu^T y_2 \right).$$

Note that g_1 and g_2 can be evaluated completely independently, *e.g.*, in parallel. Also note that g_1 and g_2 can be expressed in terms of the conjugates of f_1 and f_2:

$$g_1(\nu) = -f_1^*(0, -\nu), \qquad g_2(\nu) = -f_2^*(0, \nu).$$

The dual problem is

$$\text{maximize} \quad g_1(\nu) + g_2(\nu) = -f_1^*(0, -\nu) - f_2^*(0, \nu), \tag{6}$$

with variable ν. This is the master problem in *dual decomposition*. The master algorithm solves this problem using a subgradient, cutting-plane, or other method.

To evaluate a subgradient of $-g_1$ (or $-g_2$) is easy. We find $\bar{x}_1$ and $\bar{y}_1$ that minimize $f_1(x_1, y_1) + \nu^T y_1$ over x_1 and y_1. Then a subgradient of $-g_1$ at ν is given by $-\bar{y}_1$. Similarly, if $\bar{x}_2$ and $\bar{y}_2$ minimize $f_2(x_2, y_2) - \nu^T y_2$ over x_2 and y_2, then a subgradient of $-g_2$ at ν is given by $\bar{y}_2$. Thus, a subgradient of the negative dual function $-g$ is given by $\bar{y}_2 - \bar{y}_1$, which is nothing more than the consistency constraint residual.

If we use a subgradient method to solve the master problem, the dual decomposition algorithm has a very simple form.

repeat

Solve the subproblems (possibly in parallel).

Find x_1 and y_1 that minimize $f_1(x_1, y_1) + \nu^T y_1$.

Find x_2 and y_2 that minimize $f_2(x_2, y_2) - \nu^T y_2$.

Update dual variables (prices).

$\nu := \nu - \alpha_k (y_2 - y_1)$.

Here α_k is a step size which can be chosen several ways. If the dual function g is differentiable, then we can choose a constant step size, provided it is small enough. Another choice in this case is to carry out a line search on the dual objective. If the dual function is nondifferentiable, we can use a diminishing nonsummable step size, such as $\alpha_k = \alpha/k$.

At each step of the dual decomposition algorithm, we have a lower bound on $p^\star$, the optimal value of the original problem, given by

$$p^* \geq g(\nu) = f_1(x_1, y_1) + \nu^T y_1 + f_2(x_2, y_2) - \nu^T y_2.$$

where x_1, y_1, x_2, y_2 are the iterates. Generally, the iterates are not feasible for the original problem, *i.e.*, we have $y_2 - y_1 \neq 0$. (If they are feasible, we have maximized g.) A reasonable guess of a feasible point can be constructed from this iterate as

$$(x_1, \bar{y}), \qquad (x_2, \bar{y}),$$

where $\bar{y} = (y_1 + y_2)/2$. In other words, we replace y_1 and y_2 (which are different) with their average value. (The average is the projection of (y_1, y_2) onto the feasible set $y_1 = y_2$.) This gives an upper bound on $p^\star$, given by

$$p^\star \leq f_1(x_1, \bar{y}) + f_2(x_2, \bar{y}).$$

A better feasible point can be found by replacing y_1 and y_2 with their average, and then solving the two subproblems (2) and (3) encountered in *primal* decomposition, *i.e.*, by evaluating $\phi_1(\bar{y}) + \phi_2(\bar{y})$. This gives the bound

$$p^\star \leq \phi_1(\bar{y}) + \phi_2(\bar{y}).$$

Dual decomposition has an interesting economic interpretation. We imagine two separate economic units, each with its own private variables and cost function, but also with some coupled variables. We can think of y_1 as the amounts of some resources consumed by the first unit, and y_2 as the amounts of some resources generated by the second unit. Then, the consistency condition $y_1 = y_2$ means that supply is equal to demand. In primal decomposition, the master algorithm simply fixes the amount of resources to be transfered from one unit to the other, and updates these fixed transfer amounts until the total cost is minimized. In dual decomposition, we interpret ν as a set of prices for the resources. The master algorithm sets the prices, not the actual amount of the transfer from one unit to the other. Then, each unit independently operates in such a way that its cost, including the cost of the resource transfer (or profit generated from it), is minimized. The dual decomposition master algorithm adjusts the prices in order to bring the supply into consistency with the demand. In economics, the master algorithm is called a *price adjustment algorithm*, or *tatonnement* procedure.

There is one subtlety in dual decomposition. Even if we do find the optimal prices $\nu^\star$, there is the question of finding the optimal values of x_1, x_2, and y. When f_1 and f_2 are strictly convex, the points found in evaluating g_1 and g_2 are guaranteed to converge to optimal, but in general the situation can be more difficult. (For more on finding the primal solution

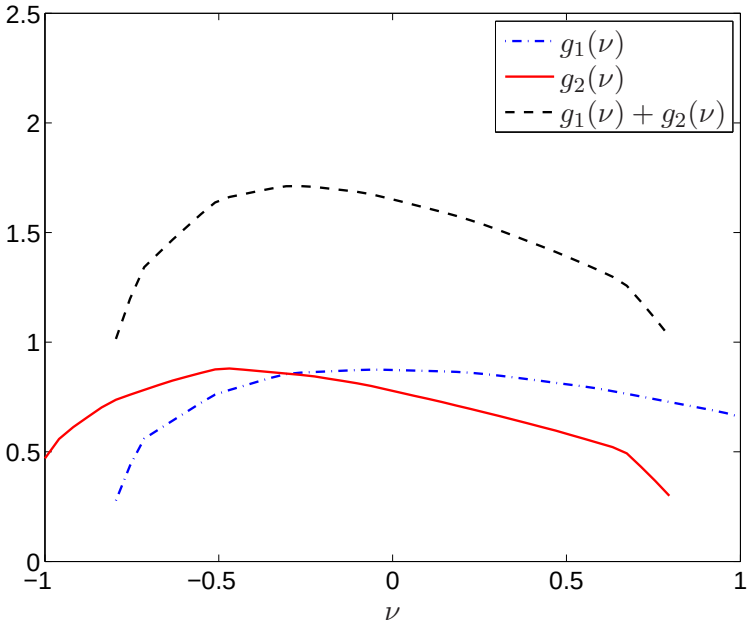

Figure 3: Dual functions versus ν.

from the dual, see [BV04, §5.5.5].) There are also some standard tricks for regularizing the subproblems that work very well in practice.

As in the primal decomposition method, we can encounter infinite values for the subproblems. In dual decomposition, we can have $g_i(\nu) = -\infty$. This can occur for some values of ν, if the functions f_i grow only linearly in y_i. In this case we generate a cutting-plane that separates the current price vector from $\mathbf{dom}\, g_i$, and use this cutting-plane to update the price vector.

2.1 Simple example

We illustrate dual decomposition with the same simple example used earlier. Figure 3 shows g_1, g_2, and $g_1 + g_2$ as functions of ν. The optimal value of ν is $\nu^\star \approx -0.27$. Figure 4 shows the progress of a bisection method for maximizing $g_1(\nu) + g_2(\nu)$, starting from initial interval $[-1, 1]$. At each step, the two subproblems are solved separately, using the current price ν. We also show two upper bounds on $p^\star$. The larger (worse) one is $f_1(x_1, \bar{y}) + f_2(x_2, \bar{y})$; the smaller (better) one is $\phi_1(\bar{y}) + \phi_2(\bar{y})$ (obtained by solving the subproblems (2) and (3)).

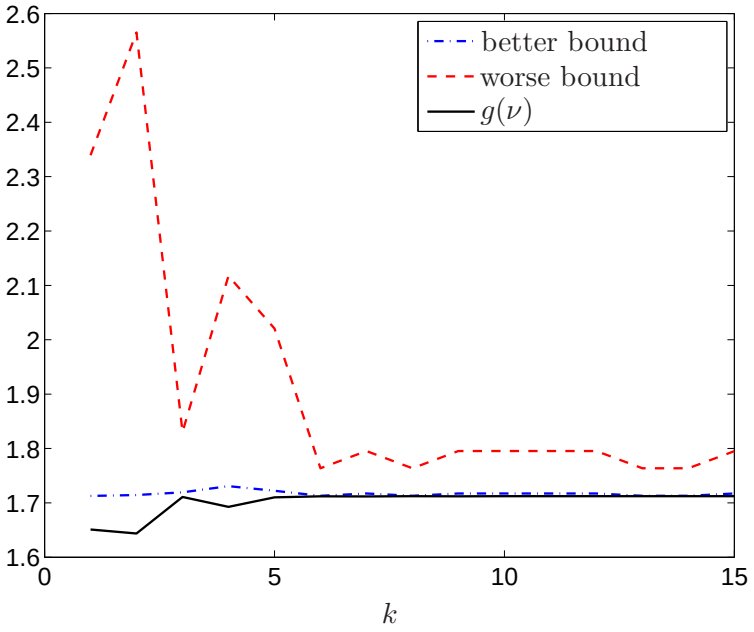

Figure 4: Convergence of dual function (lower bound), and simple and better upper bounds.

3 Decomposition with constraints

So far, we've considered the case where two problems would be separable, except for some complicating *variables* that appear in both subproblems. We can also consider the case where the two subproblems are coupled via *constraints* that involve both sets of variables. As a simple example, suppose our problem has the form

$$\begin{array}{ll} \text{minimize} & f_1(x_1) + f_2(x_2) \\ \text{subject to} & x_1 \in \mathcal{C}_1, \quad x_2 \in \mathcal{C}_2 \\ & h_1(x_1) + h_2(x_2) \preceq 0. \end{array} \tag{7}$$

Here $\mathcal{C}_1$ and $\mathcal{C}_2$ are the feasible sets of the subproblems, presumably described by linear equalities and convex inequalities. The functions $h_1 : \mathbf{R}^n \to \mathbf{R}^p$ and $h_2 : \mathbf{R}^n \to \mathbf{R}^p$ have components that are convex. The subproblems are coupled via the p constraints that involve both x_1 and x_2. We refer to these as *complicating constraints* (since without them, the problems involving x_1 and x_2 can be solved separately).

3.1 Primal decomposition

To use primal decomposition, we can introduce a variable $t \in \mathbf{R}^p$ that represents the amount of the resources allocated to the first subproblem. As a result, $-t$ is allocated to the second subproblem. The first subproblem becomes

$$\begin{array}{ll} \text{minimize} & f_1(x_1) \\ \text{subject to} & x_1 \in \mathcal{C}_1, \quad h_1(x_1) \preceq t, \end{array} \tag{8}$$

and the second subproblem becomes

$$\begin{array}{ll} \text{minimize} & f_2(x_2) \\ \text{subject to} & x_2 \in \mathcal{C}_2, \quad h_2(x_2) \preceq -t. \end{array} \tag{9}$$

Let $\phi_1(t)$ and $\phi_2(t)$ denote the optimal values of the subproblems (8) and (8), respectively. Evidently the original problem (7) is equivalent to the master problem of minimizing $\phi(t) = \phi_1(t) + \phi_2(t)$ over the allocation vector t. These subproblems can be solved separately, when t is fixed.

Not surprisingly, we can find a subgradient for the optimal value of each subproblem from an optimal dual variable associated with the coupling constraint. Let $p(z)$ be the optimal value of the convex optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in X, \quad h(x) \preceq z, \end{array}$$

and suppose $z \in \mathbf{dom}\, p$. Let $\lambda(z)$ be an optimal dual variable associated with the constraint $h(x) \preceq z$. Then, $-\lambda(z)$ is a subgradient of p at z. To see this, we consider the value of p at

another point $\tilde{z}$:

$$\begin{aligned} p(\tilde{z}) &= \sup_{\lambda \succeq 0} \inf_{x \in X} \left(f(x) + \lambda^T (h(x) - \tilde{z}) \right) \\ &\geq \inf_{x \in X} \left(f(x) + \lambda(z)^T (h(x) - \tilde{z}) \right) \\ &= \inf_{x \in X} \left(f(x) + \lambda(z)^T (h(x) - z + z - \tilde{z}) \right) \\ &= \inf_{x \in X} \left(f(x) + \lambda(z)^T (h(x) - z) \right) + \lambda(z)^T (z - \tilde{z}) \\ &= \phi(z) + (-\lambda(z))^T (\tilde{z} - z). \end{aligned}$$

This holds for all points $\tilde{z} \in \mathbf{dom}\, p$, so $-\lambda(z)$ is a subgradient of p at z. (See [BV04, §5.6].)

Thus, to find a subgradient of ϕ, we solve the two subproblems, to find optimal x_1 and x_2, as well as optimal dual variables λ_1 and λ_2 associated with the constraints $h_1(x_1) \preceq t$ and $h_2(x_2) \preceq -t$, respectively. Then we have $\lambda_2 - \lambda_1 \in \partial\phi(t)$. It is also possible that $t \notin \mathbf{dom}\, \phi$. In this case we can generate a cutting plane that separates t from $\mathbf{dom}\, \phi$, for use in the master algorithm.

Primal decomposition, using a subgradient master algorithm, has the following simple form.

repeat

Solve the subproblems (possibly in parallel).

Solve (8), to find an optimal x_1 and λ_1.

Solve (9), to find an optimal x_2 and λ_2.

Update resource allocation.

$t := t - \alpha_k(\lambda_2 - \lambda_1)$.

Here α_k is an appropriate step size. At every step of this algorithm we have points that are feasible for the original problem.

3.2 Dual decomposition

Dual decomposition for this example is straightforward. We form the partial Lagrangian,

$$\begin{aligned} L(x_1, x_2, \lambda) &= f_1(x_1) + f_2(x_2) + \lambda^T (h_1(x_1) + h_2(x_2)) \\ &= \left(f_1(x_1) + \lambda^T h_1(x_1) \right) + \left(f_2(x_2) + \lambda^T h_2(x_2) \right) \end{aligned}$$

which is separable, so we can minimize over x_1 and x_2 separately, given the dual variable λ, to find $g(\lambda) = g_1(\lambda) + g_2(\lambda)$. For example, to find $g_1(\lambda)$, we solve the subproblem

$$\begin{array}{ll} \text{minimize} & f_1(x_1) + \lambda^T h_1(x_1) \\ \text{subject to} & x_1 \in \mathcal{C}_1, \end{array} \tag{10}$$

and to find $g_2(\lambda)$, we solve the subproblem

$$\begin{array}{ll} \text{minimize} & f_2(x_2) + \lambda^T h_2(x_2) \\ \text{subject to} & x_2 \in \mathcal{C}_2. \end{array} \tag{11}$$

A subgradient of $-g_1$ at λ is, naturally, $h_1(\bar{x}_1)$, where $\bar{x}_1$ is any solution of subproblem (10).

To find a subgradient of g, the master problem objective, we solve both subproblems, to get solutions $\bar{x}_1$ and $\bar{x}_2$, respectively. A subgradient of $-g$ is then $h_1(\bar{x}_1) + h_2(\bar{x}_2)$. The master algorithm updates (the price vector) λ based on this subgradient.

If we use a projected subgradient method to update λ we get a very simple algorithm.

repeat

Solve the subproblems (possibly in parallel).

Solve (10) to find an optimal $\bar{x}_1$.

Solve (11) to find an optimal $\bar{x}_2$.

Update dual variables (prices).

$\lambda := (\lambda + \alpha_k(h_1(\bar{x}_1) + h_2(\bar{x}_2)))_+$.

At each step we have a lower bound on $p^\star$, given by

$$g(\lambda) = g_1(\lambda) + g_2(\lambda) = f_1(x_1) + \lambda^T h_1(x_1) + f_2(x_2) + \lambda^T h_2(x_2).$$

The iterates in the dual decomposition method need not be feasible, *i.e.*, we can have $h_1(x_1) + h_2(x_2) \not\preceq 0$. At the cost of solving two additional subproblems, however, we can (often) construct a feasible set of variables, which will give us an upper bound on $p^\star$. When $h_1(x_1) + h_2(x_2) \not\preceq 0$, we define

$$t = (h_1(x_1) - h_2(x_2))/2, \tag{12}$$

and solve the primal subproblems (8) and (9). This is nothing more than projecting the current (infeasible) resources used, $h_1(x_1)$ and $h_2(x_2)$, onto the set of feasible resource allocations, which must sum to no more than 0. As in primal decomposition, it can happen that $t \notin \mathbf{dom}\, \phi$. But when $t \in \mathbf{dom}\, \phi$, this method gives a feasible point, and an upper bound on $p^\star$.

3.3 Simple example

We look at a simple example to illustrate decomposition with constraints. The problem is a (strictly convex) QP with a pair of coupling resource constraints. It can be split into two subproblems, with $x_1 \in \mathbf{R}^{20}$, $x_2 \in \mathbf{R}^{20}$. Each subproblem has 100 linear inequalities, and the two subproblems share 2 complicating linear inequalities. The optimal value of the problem is $p^\star \approx -1.33$.

Figure 5 shows the progress of primal decomposition for the problem, using a subgradient method with step size $\alpha_k = 0.1$. Figure 6 shows the resources consumed by the first of the two subproblems.

We use the same example to illustrate dual decomposition, using a subgradient method, with step size $\alpha_k = 0.5/k$, to solve the master problem. Figure 7 shows the evolution of the resource prices. At each step we generate a feasible point for the original problem using the fixed resource allocation (12), and solving the primal subproblems (8) and (9). The associated upper bound on $p^\star$, which we denote $\hat{f}$, and the lower bound obtained from the

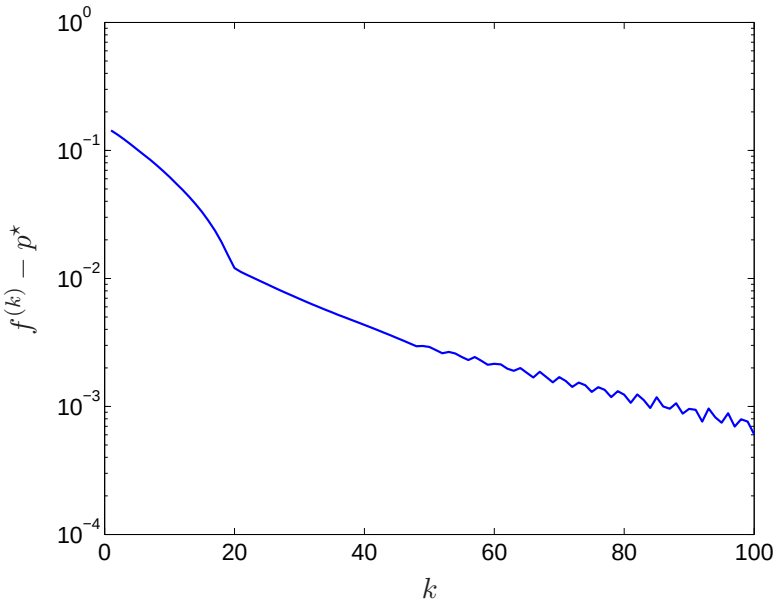

Figure 5: Suboptimality versus iteration number k for primal decomposition.

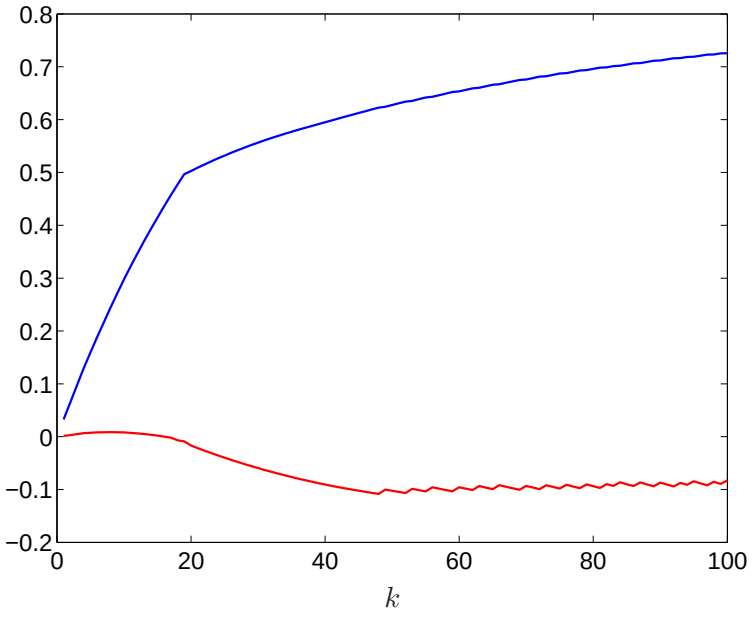

Figure 6: First subsystem resource allocation versus iteration number k for primal decomposition.

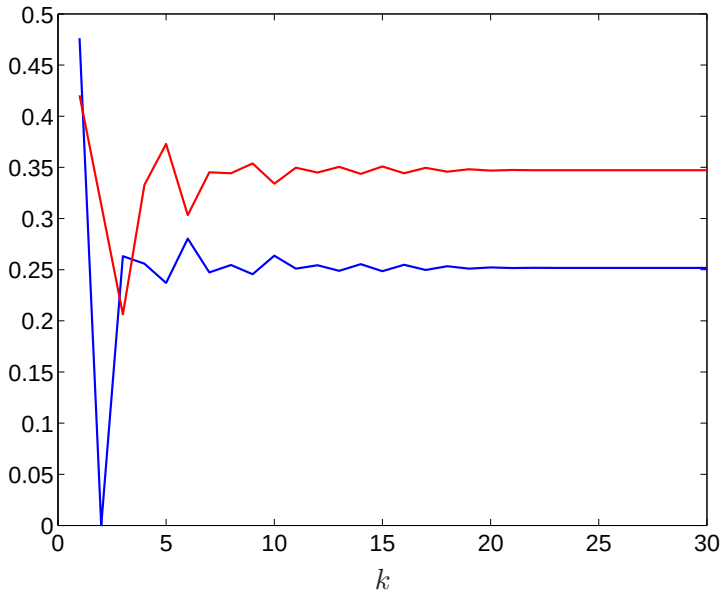

Figure 7: Resource prices versus iteration number k.

dual function $g(\lambda)$, are plotted in figure 8. The gap between the best upper and lower bounds is seen to be very small after just 5 iterations or so. Figure 9 shows the actual suboptimality of $\hat{f}$, and the gap between the upper and lower bounds.

3.4 Coupling constraints and coupling variables

Except for the details of computing the relevant subgradients, primal and dual decomposition for problems with coupling variables and coupling constraints seem quite similar. In fact, we can readily transform each into the other. For example, we can start with the problem with coupling constraints (7), and introduce new variables y_1 and y_2, that bound the subsystem coupling constraint functions, to obtain

$$\begin{array}{lll} \text{minimize} & f_1(x_1) + f_2(x_2) & \\ \text{subject to} & x_1 \in \mathcal{C}_1, \quad h_1(x_1) \preceq y_1 & \\ & x_2 \in \mathcal{C}_2, \quad h_2(x_2) \preceq -y_2 & \\ & y_1 = y_2. & \end{array} \tag{13}$$

We now have a problem of the form (5), *i.e.*, a problem that is separable, except for a *consistency constraint*, that requires two (vector) variables of the subproblems to be equal.

Any problem that can be decomposed into two subproblems that are coupled by some common variables, or equality or inequality constraints, can be put in this standard form, *i.e.*, two subproblems that are independent except for one consistency constraint, that requires a subvariable of one to be equal to a subvariable of the other. Primal or dual decomposition is then readily applied; only the details of computing the needed subgradients for the master problem vary from problem to problem.

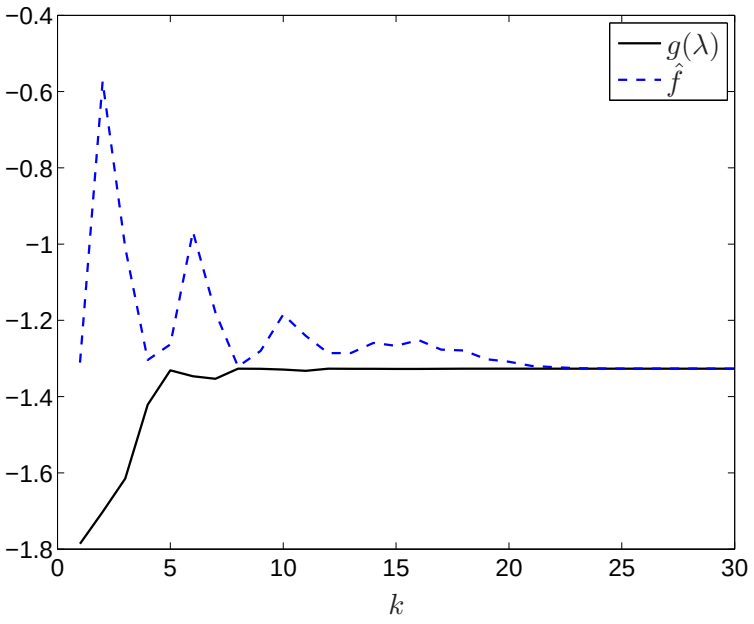

Figure 8: Upper and lower bounds versus iteration number k, for dual decomposition algorithm.

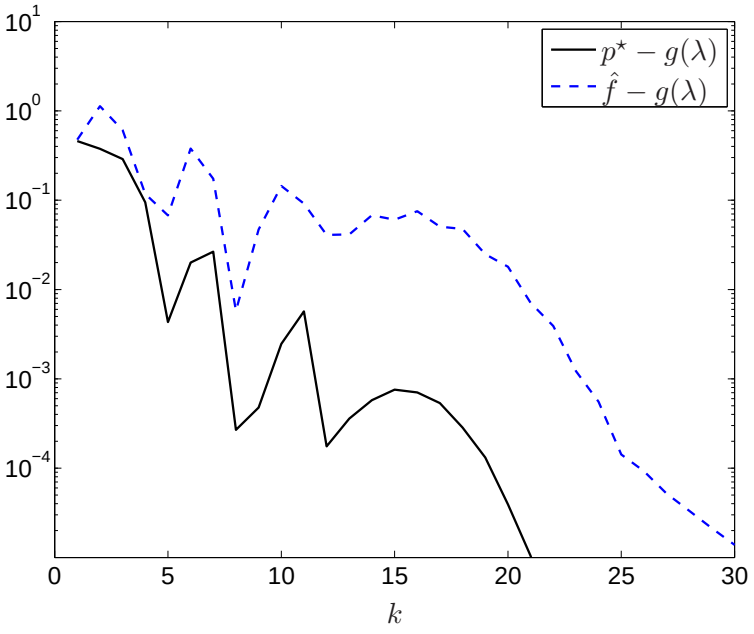

Figure 9: Gap between upper bound and $p^\star$, and gap between upper bound and lower bound (*i.e.*, $g(\lambda)$) versus iteration number k, for dual decomposition algorithm.

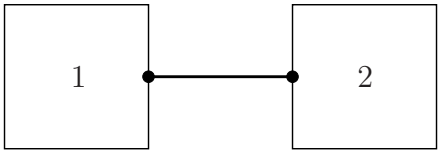

Figure 10: Simplest decomposition structure.

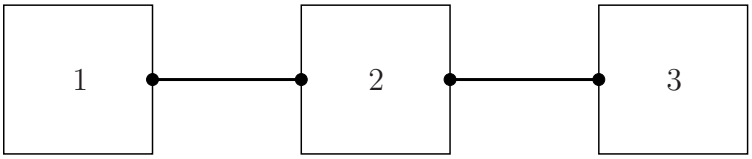

Figure 11: Chain structure, with three subsystems and two coupling constraints.

4 More general decomposition structures

So far we have studied the case where there are two subsystems that are coupled by shared variables, or coupling constraints. Clearly we can have more than one subproblem, with various subsets of them coupled in various ways. For example, the variables might be partitioned into subvectors, some of which are local (*i.e.*, appear in one subproblem only) and some of which are complicating (*i.e.*, appear in more than one subproblem). This decomposition structure can be represented by a hypergraph. The nodes are associated with the subproblems, which involve local variables, objective terms, and local constraints. The hyperedges or nets are associated with complicating variables or constraints. If a hyperedge is adjacent to only two nodes, we call it a link. A link corresponds to a shared variable or constraint between the two subproblems represented by the nodes. The simplest possible decomposition structure consists of two subsystems and some coupling between them, as shown in figure 10. In this figure the subsystems are shown as the boxes labeled 1 and 2; the coupling between them is shown as the link connecting the boxes.

Figure 11 shows another simple decomposition structure with three subsystems, labeled 1, 2, and 3. This problem consists of three subproblems, with some coupling between subproblems 1 and 2, and some coupling between subproblems 2 and 3. Adopting the canonical form for coupling, *i.e.*, as consistency constraints, we can write the associated problem as

$$\begin{array}{ll} \text{minimize} & f_1(x_1, y_1) + f_2(x_2, y_2, y_3) + f_3(x_3, y_4) \\ \text{subject to} & (x_1, y_1) \in \mathcal{C}_1, \quad (x_2, y_2, y_3) \in \mathcal{C}_2, \quad (x_3, y_4) \in \mathcal{C}_3 \\ & y_1 = y_2, \quad y_3 = y_4. \end{array}$$

Subsystem 1 has private or local variable x_1, and public or interface variable y_1. Subsystem 2 has local variable x_2, and interface variables y_2 and y_3. Subsystem 3 has local variable x_3, and interface variable y_4. This decomposition structure has two edges: the first edge corresponds to the consistency constraint $y_1 = y_2$, and the second edge corresponds to the consistency constraint $y_3 = y_4$.

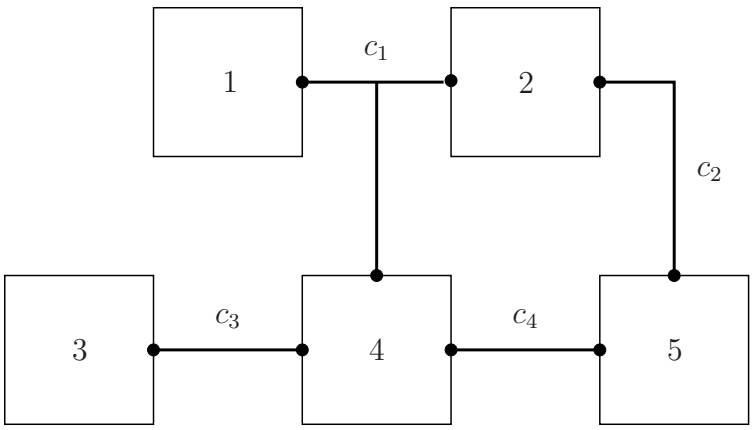

Figure 12: A more complex decomposition structure with 5 subsystems and 4 coupling constraints.

A more complex decomposition structure is shown in figure 12. This consists of 5 subsystems and 4 coupling constraints. Coupling constraint c_1, for example, requires that three public variables of subsystem 1, 2, and 3 should all be equal.

We will give the details later, but we describe, roughly, how primal and dual decomposition works in the more general setting with complex decomposition structure. In primal decomposition, each hyperedge or net has a single variable associated with it. Each subsystem is optimized separately, using the public variable values (asserted) on the nets. Each subsystem produces a subgradient associated with each net it is adjacent to. These are combined to update the variable value on the net, hopefully in such a way that convergence to (global) optimality occurs.

In dual decomposition, each subsystem has its own private copy of the public variables on the nets it is adjacent to, as well as an associated price vector. The subsystems use these prices to optimize their local variables, including the local copies of public variables. The public variables on each net are then compared, and the prices are updated, hopefully in a way that brings the local copies of public variables into consistency (and therefore also optimality).

4.1 General examples

Before introducing some formal notation for decomposition structures, we informally describe a few general examples.

In an optimal control problem, the state at time t is the complicating variable between the past (*i.e.*, variables associated with time before t) and the future (variables associated with time after t). In other words, if you fix the state in a dynamical system, the past and future have nothing to do with each other. (That's exactly what it means to be a state.) In terms of a decomposition graph, we have nodes associated with the system at times $t = 1, \ldots, T$;

each node is coupled to the node before and after, by the state equations. Thus, the optimal control problem decomposition structure is represented by a simple linear chain.

Decomposition structure arises in many applications in network optimization. For example, we can partition a network into subnetworks, that interact only via common flows, or their boundary connections. We can partition a multi-commodity flow problem into a set of single-commodity flow problems coupled by shared resources. the capacities of the links. In a flow control problem, we can view each flow in a network as a subsystem; these flows are coupled by sharing the capacities of the links.

In some image processing problems, variables associated with pixels are only coupled to the variables associated with some of their neighbors. In this case, any strip with a width exceeding the interaction distance between pixels, and which disconnects the image plane can be taken as a set of complicating variables. You can solve an image restoration problem, then, by fixing a strip (say, down the middle), and then (in parallel) solving the left and right image problems. (This can clearly be done recursively as well.)

Decomposition structure arises in hierarchical design. Suppose we are designing (via convex optimization) a large circuit (say) that consists of some subcircuits. Each subcircuit has many private variables, and a few variables that interact with other subcircuits. For example, the device dimensions inside each subcircuit might be local or private variables; the shared variables correspond to electrical connections between the subcircuits (*e.g.*, the load presented to one subcircuit from another) or objectives or constraint that couple them (*e.g.*, a total power or area limit). At each step of algorithm in primal decomposition, we fix the coupling variables, and then design the subcircuits (separately) to meet these fixed specifications on their boundaries. We then update the coupling variables in such a way that the total cost (say, power) eventually is minimized. In dual decomposition, we allow each subcircuit to choose its own values for its boundary variables, but add an extra cost, based on prices, to account for its effects on the overall circuit. These prices are updated to bring the design into consistency.

4.2 Framework for decomposition structures

In this section we describe decomposition with a general structure in more detail. We have K subsystems. Subsystem i has private variables $x_i \in \mathbf{R}^{n_i}$, public variables $y_i \in \mathbf{R}^{p_i}$, objective function $f_i : \mathbf{R}^{n_i} \times \mathbf{R}^{p_i}$, and local constraint set $\mathcal{C}_i \subseteq \mathbf{R}^{n_i} \times \mathbf{R}^{p_i}$. The overall objective is $\sum_{i=1}^K f_i(x_i, y_i)$, and the local constraints are $(x_i, y_i) \in \mathcal{C}_i$.

These subsystems are coupled through constraints that require various subsets of the components of the public variables to be equal. (Each of these subsets corresponds to a hyperedge or net in the decomposition structure.) To describe this we collect all the public variables together into one vector variable $y = (y_1, \ldots, y_K) \in \mathbf{R}^p$, where $p = p_1 + \cdots + p_K$ is the total number of (scalar) public variables. We use the notation $(y)_i$ to denote the ith (scalar) component of y, for $i = 1, \ldots, p$ (in order to distinguish it from y_i, which refers to the portion of y associated with subsystem i).

We suppose there are N nets, and we introduce a vector $z \in \mathbf{R}^N$ that gives the common values of the public variables on the nets. We can express the coupling constraints as $y = Ez$,

where $E \in \mathbf{R}^{p\times N}$ is the matrix with

$$E_{ij} = \begin{cases} 1 & (y)_i \text{ is in net } j \\ 0 & \text{otherwise.} \end{cases}$$

The matrix E specifies the netlist, or set of hyperedges, for the decomposition structure. We will let $E_i \in \mathbf{R}^{p_i\times N}$ denote the partitioning of the rows of E into blocks associated with the different subsystems, so that $y_i = E_i z$. The matrix E_i is a 0-1 matrix that maps the vector of net variables into the public variables of subsystem i.

Our problem then has the form

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^K f_i(x_i, y_i) \\ \text{subject to} & (x_i, y_i) \in \mathcal{C}_i, \quad i = 1, \ldots, K \\ & y_i = E_i z, \quad i = 1, \ldots, K, \end{array} \tag{14}$$

with variables x_i, y_i, and z. We refer to z as the vector of (primal) *net variables*.

4.3 Primal decomposition

In primal decomposition, at each iteration we fix the vector z of net variables, and we fix the public variables as $y_i = E_i z$. The problem is now separable; each subsystem can (separately) find optimal values for its local variables x_i. Let $\phi_i(y_i)$ denote the optimal value of the subproblem

$$\begin{array}{ll} \text{minimize} & f_i(x_i, y_i) \\ \text{subject to} & (x_i, y_i) \in \mathcal{C}_i, \end{array} \tag{15}$$

with variable x_i, as a function of y_i. The original problem (14) is equivalent to the primal master problem

$$\text{minimize} \quad \phi(z) = \textstyle\sum_{i=1}^K \phi_i(E_i z),$$

with variable z. To find a subgradient of ϕ, we find $g_i \in \partial\phi_i(y_i)$ (which can be done separately). We then have

$$g = \sum_{i=1}^K E_i^T g_i \in \partial\phi(z).$$

This formula has a simple interpretation: To find a subgradient for the net variable z_i, we collect and sum the subgradients over all components of the public variables adjacent to net i.

If the master problem is solved using a subgradient method, we have the following algorithm.

repeat

Distribute net variables to subsystems.

$y_i := E_i z$, $i = 1, \ldots, K$.

Optimize subsystems (separately).

Solve subproblems (15) to find optimal x_i, and $g_i \in \partial\phi_i(y_i)$, $i = 1, \ldots, K$.

Collect and sum subgradients for each net.
$g := \sum_{i=1}^K E_i^T g_i$.
Update net variables.
$z := z - \alpha_k g$.

Here α_k is an appropriate step size. This algorithm is decentralized: at each step, the actions taken involve only the subsystems, which act independently of each other, or the nets, which act independently of each other. The only communication is between subsystems and the nets they are adjacent to. There is no communication between (or, to anthropomorphize a bit, any awareness of) different subsystems, or different nets.

4.4 Dual decomposition

We form the partial Lagrangian of problem (14),

$$\begin{aligned} L(x,y,z,\nu) &= \sum_{i=1}^K f_i(x_i,y_i) + \nu^T(y - Ez) \\ &= \sum_{i=1}^K \left(f_i(x_i,y_i) + \nu_i^T y_i\right) - \nu^T E z, \end{aligned}$$

where $\nu \in \mathbf{R}^p$ is the Lagrange multiplier associated with $y = Ez$, and ν_i is the subvector of ν associated with the ith subsystem. To find the dual function we first minimize over z, which results in the condition $E^T\nu = 0$ for $g(\nu) > -\infty$. This condition states that for each net, the sum of the Lagrange multipliers over the net is zero. We define $g_i(\nu_i)$ as the optimal value of the subproblem

$$\begin{array}{ll} \text{minimize} & f_i(x_i,y_i) + \nu_i^T y_i \\ \text{subject to} & (x_i,y_i) \in \mathcal{C}_i, \end{array} \tag{16}$$

as a function of ν_i. A subgradient of $-g_i$ at ν_i is just $-y_i$, an optimal value of y_i in the subproblem (16).

The dual of the original problem (14) is

$$\begin{array}{ll} \text{maximize} & g(\nu) = \sum_{i=1}^K g_i(\nu_i) \\ \text{subject to} & E^T\nu = 0, \end{array}$$

with variable ν.

We can solve this dual decomposition master problem using a projected subgradient method. Projection onto the feasible set $\{\nu \mid E^T\nu = 0\}$, which consists of vectors whose sum over each net is zero, is easy to work out. The projection is given by multiplication by $I - E(E^TE)^{-1}E^T$, which has a particularly simple form, since

$$E^TE = \mathbf{diag}(d_1,\ldots,d_N),$$

where d_i is the degree of net i, *i.e.*, the number of subsystems adjacent to net i. For $u \in \mathbf{R}^p$, $(E^TE)^{-1}Eu$ gives the average, over each net, of the entries in the vector u. The vector

$(E^TE)^{-1}Eu$ is the vector obtained by replacing each entry of u with its average over its associated net. Finally, projection of u onto the feasible set is obtained by subtracting from each entry the average of other values in the associated net.

Dual decomposition, with a subgradient method for the master problem, gives the following algorithm.

given initial price vector ν that satisfies $E^T\nu = 0$ (*e.g.*, $\nu = 0$).

repeat

Optimize subsystems (separately).

Solve subproblems (16) to obtain x_i, y_i.

Compute average value of public variables over each net.

$\hat{z} := (E^TE)^{-1}E^Ty$.

Update prices on public variables.

$\nu := \nu + \alpha_k(y - E\hat{z})$.

Here α_k is an appropriate step size. This algorithm, like the primal decomposition algorithm, is decentralized: At each step, the actions taken involve only the subsystems, acting independently of each other, or the nets, acting independently of each other.

We note that $\hat{z}$, computed in the second step, gives a reasonable guess for $z^\star$, the optimal net variables. If we solve the primal subproblems (15), using $y_i = E_i\hat{z}$, we obtain a feasible point, and an associated upper bound on the optimal value.

The vector $y - Ez = (I - E(E^TE)^{-1}E^T)y$, computed in the last step, is the projection of the current values of the public variables onto the set of feasible, or consistent values of public variables, *i.e.*, those that are the same over each net. The norm of this vector gives a measure of the inconsistency of the current values of the public variables.

4.5 Example

Our example has the structure shown in figure 12. Each of the local variables has dimension 10, and all 9 public variables are scalar, so all together there are 50 private variables, 9 public variables, and 5 linear equality constraints. (The hyperedge labeled c_1 requires that three public variables be equal, so we count it as two linear equality constraints.) Each subsystem has an objective term that is a convex quadratic function plus a piecewise-linear function. There are no local constraints. The optimal value of the problem is $p^\star \approx 11.1$.

We use dual decomposition with fixed step size $\alpha = 0.5$. At each step, we compute two feasible points. The simple one is $(x, \hat{y})$, with $\hat{y} = E\hat{z}$. The more costly, but better, point is $(\hat{x}, \hat{y})$ where $\hat{x}$ is found by solving the primal decomposition subproblems using $\hat{y}$. Figure 13 shows $g(\nu)$, and the objectove for the two feasible points, $f(x, \hat{y})$ and $f(\hat{x}, \hat{y})$, versus iteration number. Figure 14 shows $\|y - E\hat{z}\|$, the norm of the consistency constraint residual, versus iteration number.

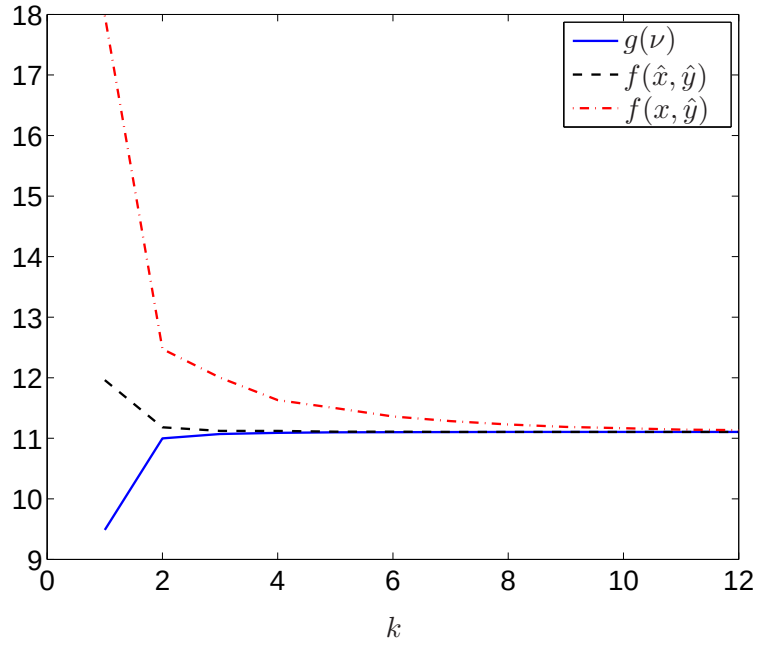

Figure 13: Upper and lower bounds versus iteration number k, for dual decomposition algorithm.

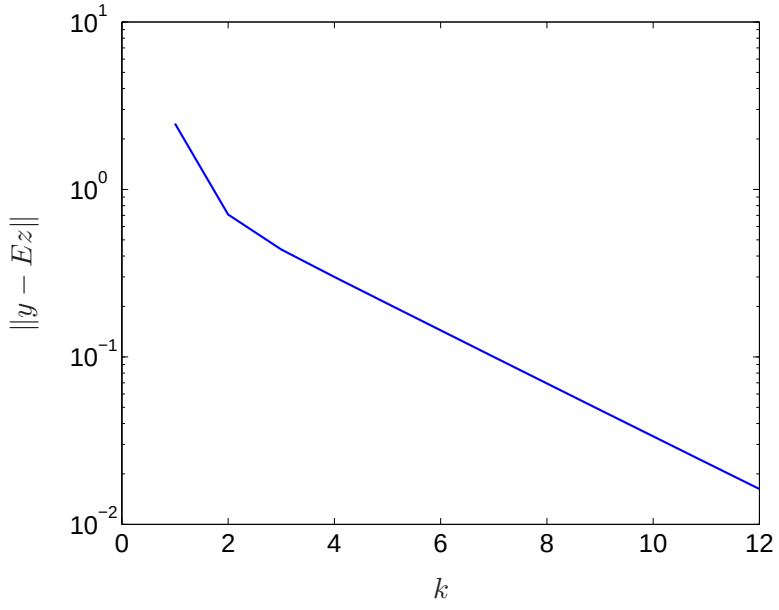

Figure 14: Norm of the consistency constraint residual, $\|y - Ez\|$, versus iteration number k, for dual decomposition algorithm.

5 Rate control

There are n *flows* in a network, each of which passes over a fixed route, *i.e.*, some subset of m links. Each flow has a nonnegative *flow rate* or *rate*, which we denote $f_1, \ldots, f_n$. With flow j we associate a utility function $U_j : \mathbf{R} \to \mathbf{R}$, which is strictly concave and increasing, with $\mathbf{dom}\, U_j \subseteq \mathbf{R}_+$. The utility derived by a flow rate f_j is given by $U_j(f_j)$. The total utility associated with all the flows is then $U(f) = U_1(f_1) + \cdots + U_n(f_n)$.

The total traffic on a link in the network, denoted $t_1, \ldots, t_m$, is the sum of the rates of all flows that pass over that link. We can express the link traffic compactly using the *routing* or *link-route* matrix $R \in \mathbf{R}^{m\times n}$, defined as

$$R_{ij} = \begin{cases} 1 & \text{flow } j \text{ passes over link } i \\ 0 & \text{otherwise,} \end{cases}$$

as $t = Rf$. Each link in the network has a (positive) *capacity* $c_1, \ldots, c_m$. The traffic on a link cannot exceed its capacity, *i.e.*, we have $Rf \preceq c$.

The *flow rate control problem* is to choose the rates to maximize total utility, subject to the link capacity constraints:

$$\begin{array}{ll} \text{maximize} & U(f) \\ \text{subject to} & Rf \preceq c, \end{array} \tag{17}$$

with variable $f \in \mathbf{R}^n$. This is evidently a convex optimization problem.

We can decompose the rate control problem in several ways. For example, we can view each flow as a separate subsystem, and each link capacity constraint as a complicating constraint that involves the flows that pass over it.

5.1 Dual decomposition

The Lagrangian (for the problem of minimizing $-U$) is

$$L(f, \lambda) = -\sum_{j=1}^{n} U_j(f_j) + \lambda^T(Rf - c),$$

and the dual function is given by

$$\begin{aligned} g(\lambda) &= \inf_f \left(\sum_{j=1}^{n} -U_j(f_j) + \lambda^T(Rf - c) \right) \\ &= -\lambda^T c + \sum_{j=1}^{n} \inf_{f_j} (-U_j(f_j) + (r_j^T\lambda) f_j) \\ &= -\lambda^T c - \sum_{j=1}^{n} (-U_j)^*(r_j^T\lambda), \end{aligned}$$

where r_j is the jth column of R. The number $r_j^T\lambda$ is the sum of the Lagrange multipliers associated with the links along route j.

The dual problem is

$$\begin{array}{ll} \text{maximize} & -\lambda^T c - \sum_{j=1}^n (-U_j)^*(-r_j^T \lambda) \\ \text{subject to} & \lambda \succeq 0. \end{array} \tag{18}$$

A subgradient of $-g$ is given by $R\bar{f} - c$, where $\bar{f}_j$ is a solution of the subproblem

$$\text{minimize} \quad -U_j(f_j) + (r_j^T \lambda) f_j,$$

with variable f_j. (The constraint $f_j \geq 0$ is implicit here.)

Using a projected subgradient method to solve the dual problem, we obtain the following algorithm.

given initial link price vector $\lambda \succ 0$ (*e.g.*, $\lambda = \mathbf{1}$).

repeat

Sum link prices along each route.

Calculate $\Lambda_j = r_j^T \lambda$.

Optimize flows (separately) using flow prices.

$f_j := \operatorname{argmax} (U_j(f_j) - \Lambda_j f_j)$.

Calculate link capacity margins.

$s := c - Rf$.

Update link prices.

$\lambda := (\lambda - \alpha_k s)_+$.

Here α_k is an appropriate stepsize.

This algorithm is completely decentralized: Each flow is updated based on information obtained from the links it passes over, and each link price is updated based only on the flows that pass over it. The algorithm is also completely natural. We can imagine that a flow is charged a price λ_l (per unit of flow) for passing over link i. The total charge for the flow is then $\Lambda_j f_j$. This charge is subtracted from its utility, and the maximum net utility flow rate chosen. The links update their prices depending on their capacity margin $s = c - t$, where $t = Rf$ is the link traffic. If the margin is positive, the link price is decreased (but not below zero). If the margin is negative, which means the link capacity constraint is violated, the link price is increased.

The flows at each step of the algorithm can violate the capacity constraints. We can generate a set of feasible flows by fixing an allocation of each link capacity to each flow that passes through it, and then optimizing the flows. Let $\eta \in \mathbf{R}^m$ be the factors by which the link traffic exceeds link capacity, *i.e.*, $\eta_i = t_i/c_i$, where $t = Rf$ is the traffic. If $\eta_i \leq 1$, link i is operating under capacity; if $\eta_i > 1$, link i is operating over capacity. Define f^{feas} as

$$f_j^{\text{feas}} = \frac{f_j}{\max\{\eta_i \mid \text{flow } j \text{ passes over link } i\}}, \quad j = 1, \ldots, n. \tag{19}$$

This flow vector will be feasible. Roughly speaking, each flow is backed off by the maximum over capacity factor over its route. (If all links on a route are under-utilized, this scheme will actually increase the flow.)

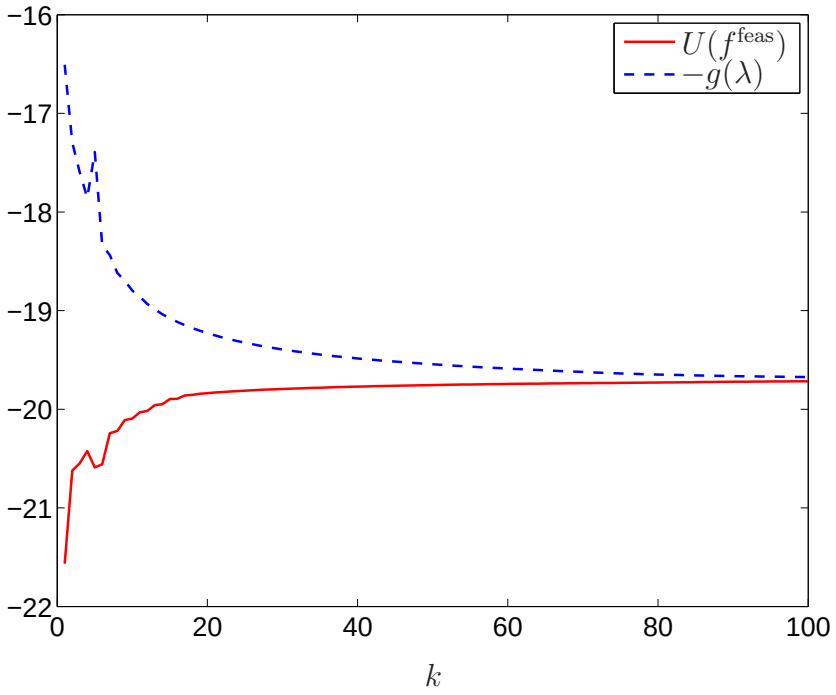

Figure 15: Upper bound $-g(\lambda)$ and lower bound $U(f^{\text{feas}})$ on optimal utility, versus iteration k.

5.2 Example

We consider an example with $n = 10$ flows and $m = 12$ links. The number of links per flow is either 3 or 4, and the number of flows per link is around 3. The link capacities are chosen randomly, uniformly distributed on $[0.1, 1]$. We use a log utility function, *i.e.*, $U_j(f_j) = \log f_j$. (This can be argued to achieve fairness among the flows.) The optimal flow, as a function of price, is

$$\operatorname{argmax}\left(U_j(f_j) - \Lambda_j f_j\right) = 1/\Lambda_j.$$

We initialize the link prices at $\lambda = \mathbf{1}$, and use a constant stepsize $\alpha_k = 3$. (The dual function g is differentiable, so a small enough constant step size with result in convergence.)

Figure 15 shows the evolution of the dual decomposition method. The upper plot shows the bound $-g(\lambda)$ on the optimal utility. The bottom plot shows the utility achieved by the feasible flow found from (19). Figure 16 shows the evolution of the maximum capacity violation, *i.e.*, $\max_i(Rf - c)_i$.

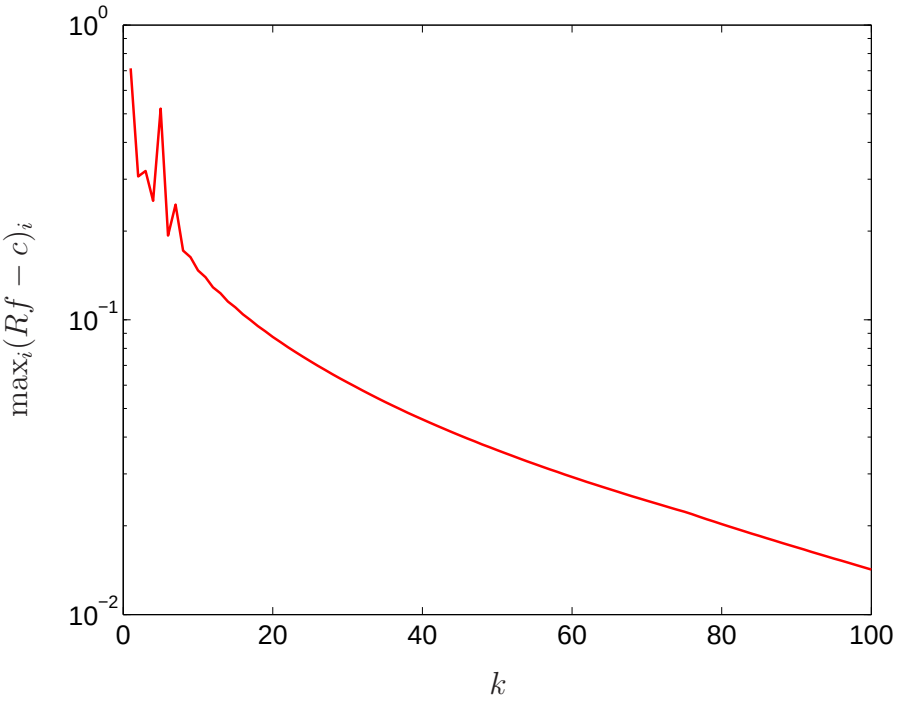

Figure 16: Maximum capacity violation versus iteration k.

6 Single commodity network flow

We consider a connected directed graph or network with n edges and p nodes. We let x_j denote the flow or traffic on arc j, with $x_j > 0$ meaning flow in the direction of the arc, and $x_j < 0$ meaning flow in the direction opposite the arc. There is also a given external source (or sink) flow s_i that enters (if $s_i > 0$) or leaves (if $s_i < 0$) node i. The flow must satisfy a conservation equation, which states that at each node, the total flow entering and leaving the node, including the external sources and sinks, is zero. This conservation equation can be expressed as $Ax + s = 0$ where $A \in \mathbf{R}^{p\times n}$ is the *node incidence matrix* of the graph,

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

Thus, each column of A describes a link; it has exactly two nonzero entries (one equal to 1 and the other equal to -1) indicating the end and start nodes of the link respectively. Each row of A describes all links incident to a node: the $+1$ entries indicate incoming links and the -1 entries indicate outgoing links.

The flow conservation equation $Ax + s = 0$ is inconsistent unless $\mathbf{1}^T s = 0$, which we assume is the case. (In other words, the total of the source flows must equal the total of the sink flows.) The flow conservation equations $Ax + s = 0$ are also redundant, since $\mathbf{1}^T A = 0$. To obtain an independent set of equations we can delete any one equation, to obtain $\tilde{A}x + \tilde{s} = 0$, where $\tilde{A} \in \mathbf{R}^{(p-1)\times n}$ is the *reduced node incidence matrix* of the graph (*i.e.*, the node incidence matrix with one row removed) and $\tilde{s} \in \mathbf{R}^{p-1}$ is reduced source vector (*i.e.*, s with the associated entry removed).

We will take traffic flows x as the variables, and the sources and network topology as given. We introduce the separable objective function

$$\phi(x) = \sum_{j=1}^{n} \phi_j(x_j),$$

where $\phi_j : \mathbf{R} \to \mathbf{R}$ is the flow cost function for arc j. We assume that the flow cost functions are strictly convex. We can impose other limits on the flow variables, *e.g.*, the condition $x_j \geq 0$ that flows must be in the direction of the arcs, by restricting the domain of the arc cost functions.

The problem of choosing the minimum cost flow that satisfies the flow conservation requirement is formulated as

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{n} \phi_j(x_j) \\ \text{subject to} & Ax + s = 0. \end{array} \tag{20}$$

This problem is also called the *single commodity network flow problem*.

The single commodity network flow problem is convex, and readily solved by standard methods, such as Newton's method (when ϕ_j are twice differentiable). Using dual decomposition, however, we obtain a decentralized method for solving the problem.

6.1 Dual decomposition

The Lagrangian is

$$\begin{aligned} L(x,\nu) &= \phi(x) + \nu^T(Ax+s) \\ &= \nu^T s + \sum_{j=1}^{n} \left(\phi_j(x_j) + (a_j^T\nu)x_j\right), \end{aligned}$$

where a_j is the jth column of A. We use the notation $\Delta\nu_j$ to denote $a_j^T\nu$, since it is the difference of the dual variable between the ending node and starting node of arc j. We will see that the dual variables ν_i can be interpreted as *potentials* on the network; $\Delta\nu_j$ is the potential difference appearing across arc j.

The dual function is

$$\begin{aligned} g(\nu) &= \inf_x L(x,\nu) \\ &= \nu^T s + \sum_{j=1}^{n} \inf_{x_j} \left(\phi_j(x_j) + (\Delta\nu_j)x_j\right) \\ &= \nu^T s - \sum_{j=1}^{n} \phi_j^*(-\Delta\nu_j), \end{aligned}$$

where ϕ_j^* is the conjugate function of ϕ_j, *i.e.*,

$$\phi_j^*(y) = \sup_x \left(yx_j - \phi_j(x_j)\right).$$

The dual problem is the unconstrained convex problem

$$\text{maximize} \quad g(\nu),$$

with variable $\nu \in \mathbf{R}^p$. It is easy to show that only differences in the potentials matter; we have $g(\nu) = g(\nu + c\mathbf{1})$ for any $c \in \mathbf{R}$. We can, without loss of generality, fix one of the dual variables to be zero.

There is no duality gap; the optimal values of the primal and dual problems are the same. Moreover, we can recover the primal solution from the dual solution. Since we assume the flow cost functions ϕ_j are strictly convex, for each y there is a unique maximizer of $yx_j - \phi_j(x_j)$. We will denote this maximizer as $x_j^*(y)$. If ϕ_j is differentiable, then $x_j^*(y) = (\phi_j')^{-1}(y)$, the inverse of the derivative function. We can solve the network flow problem via the dual, as follows. We first solve the dual by maximizing $g(\nu)$ over ν to obtain the optimal dual variable (or potentials) $\nu^\star$. Then the optimal solution of the network flow problem is given by

$$x_j^\star = x_j^*(-\Delta\nu_j^\star).$$

Thus, the optimal flow on link j is a function of the optimal potential difference across it. In particular, the optimal flow can be determined locally; we only need to know the optimal potential values at the two adjacent nodes to find the optimal flow on the arc.

A subgradient for the negative dual function $-g$ is

$$-(Ax^*(\Delta\nu) + s) \in \partial(-g)(\nu).$$

This is exactly the negative of the flow conservation residual. The ith component of the residual,

$$a_i^T x^*(\Delta\nu) + s_i,$$

is sometimes called the *flow surplus* at node i, since it is the difference between the total incoming and total outgoing flow at node i. Using a subgradient method to solve the dual, we obtain the following algorithm.

given initial potential vector ν.
repeat
Determine link flows from potential differences.
$x_j := x_j^*(-\Delta\nu_j), \quad j = 1, \ldots, n.$
Compute flow surplus at each node.
$S_i := a_i^T x + s_i, \quad i = 1, \ldots, p.$
Update node potentials.
$\nu_i := \nu_i + \alpha_k S_i, \quad i = 1, \ldots, p.$

Here $\alpha_k > 0$ is an appropriate step length.

The method proceeds as follows. Given the current value of the potentials, a flow is calculated. This is local; to find x_j we only need to know the two potential values at the ends of arc j. We then compute the flow surplus at each node. Again, this is local; to find the flow surplus at node i, we only need to know the flows on the arcs that enter or leave node i. Finally, we update the potentials based on the current flow surpluses. The update is very simple: we increase the potential at a node with a positive flow surplus (recall that flow surplus is $-g_i$ at node i), which (we hope) will result in reduced flow into the node. Provided the step length α_k can be computed locally, the algorithm is distributed; the arcs and nodes only need information relating to their adjacent flows and potentials. There is no need to know the global topology of the network, or any other nonlocal information, such as what the flow cost functions are.

At each step of the dual subgradient method, $g(\nu)$ is a lower bound on $p^\star$, the optimal value of the single-commodity network flow problem. (Note, however, that computing this lower bound requires collecting information from all arcs in the network.) The iterates are generally infeasible, *i.e.*, we have $Ax + s \neq 0$. The flow convervation constraint $Ax + s = 0$ is satisfied only in the limit as the algorithm converges.

There are methods to construct a feasible flow from x, an infeasible iterate of the dual subgradient method. Projecting onto the feasible set, defined by $Ax + s = 0$, can be done efficiently but is not decentralized.

6.2 Analogy with electrical networks

There is a nice analogy between the single commodity network flow problem and electrical networks. We consider an electrical network with topology determined by A. The variable

x_j is the current flow in branch j (with positive indicating flow in the reference direction, negative indicating current flow in the opposite direction). The source s_i is an external current injected at node i. Naturally, the sum of the external currents must be zero. The flow conservation equation $Ax + s = 0$ is Khirchhoff's current law (KCL).

The dual variables correspond to the node potentials in the circuit. We can arbitrarily choose one node as the ground or datum node, and measure potentials with respect to that node. The potential difference $\Delta\nu_j$ is precisely the voltage appearing across the jth branch of the circuit. Each branch in the circuit contains a nonlinear resistor, with current-voltage characteristic $I_j = x_j^*(-V_j)$.

It follows that the optimal flow is given by the current in the branches, with the topology determined by A, external current s, and current-voltage characteristics related to the flow cost functions. The node potentials correspond to the optimal dual variables. (It has been suggested to solve optimal flow equations using analog circuits.)

The subgradient algorithm gives us an iterative way to find the currents and voltages in such a circuit. The method updates the node potentials in the circuit. For a given set of node potentials we calculate the branch currents from the branch current-voltage characteristics. Then we calculate the KCL residual, *i.e.*, the excess current at each node, and update the potentials based on these mismatches. In particular, we increase the node potential at each node which has too much current flowing into it, and decrease the potential at each node which has too little current flowing into it. (For constant step size, the subgradient method corresponds roughly to putting a capacitor to ground at each node.)

6.3 Example

We now consider a more specific example, with flow cost function

$$\phi_j(x_j) = \frac{x_j}{c_j - x_j}, \qquad \mathbf{dom}\,\phi_j = [0, c_j),$$

where $c_j > 0$ are given link capacities. The domain restrictions mean that each flow is nonnegative, and must be less than the capacity of the link. This function gives the expected queuing delay in an M/M/1 queue, with exponential arrival times with rate x_j and exponential service time with rate c_j.

The conjugate of this function is

$$\phi_j^*(y) = \begin{cases} (\sqrt{c_j y} - 1)^2 & y > 1/c_j \\ 0 & y \leq 1/c_j. \end{cases}$$

The function and its conjugate are plotted in figure 17, for $c = 1$.

From the conjugate we can work out the function $x_j^*(-\Delta\nu_j)$:

$$x_j^* = \operatorname*{argmin}_{0 \leq z < c_j} (\phi_j(z) + \Delta\nu_j z) = \begin{cases} c_j - \sqrt{c_j/\Delta\nu_j} & \Delta\nu_j > 1/c_j \\ 0 & \Delta\nu_j \leq 1/c_j. \end{cases} \tag{21}$$

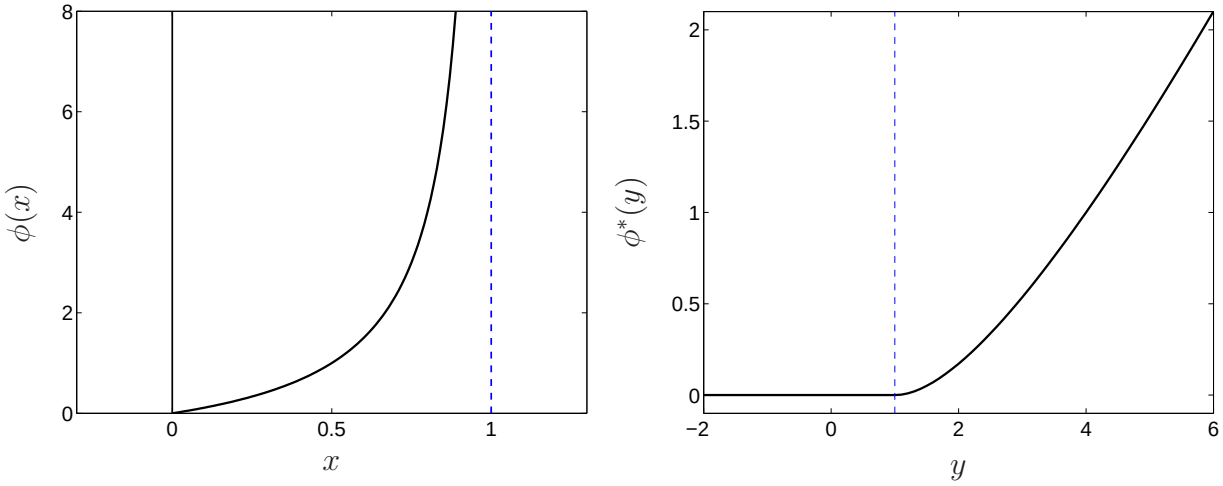

Figure 17: The queueing delay cost function $\phi(x)$ (left) and its conjugate function $\phi^*(y)$ (right), for capacity $c = 1$.

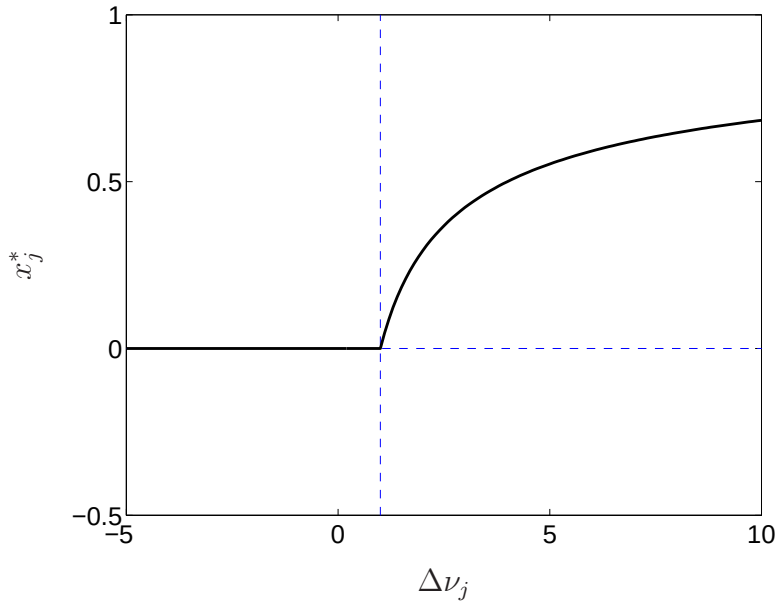

Figure 18: The function $x_j^*(-\Delta\nu_j)$, for $c_j = 1$. This can be interpreted as the current-voltage characteristic of a nonlinear resistor with diode-like characteristic.

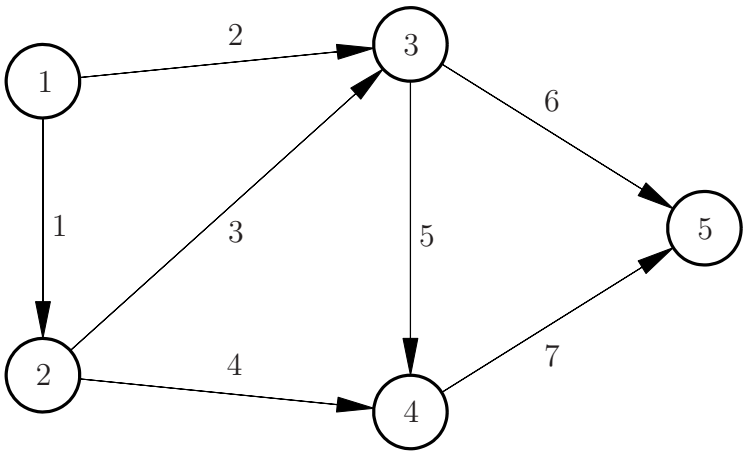

Figure 19: A network with 5 nodes and 7 arcs.

This function (which corresponds to the current-voltage characteristic of a nonlinear resistor in the analogous electrical network) is shown in figure 18. Note that it has a diode-like characteristic: current flows only in the reference direction.

Our problem instance has the network shown in figure 19, with $p = 5$ nodes and $n = 7$ arcs. Its incidence matrix is

$$A = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Each link has capacity $c_j = 1$, and the source vector is

$$s = (0.2, 0.6, 0, 0, -0.8).$$

The optimal flows are plotted in figure 20.

We use the subgradient method to solve the dual problem, with initial dual variables $\nu_i = 0$ for $i = 1, \ldots, p$. At each step of the subgradient method, we fix the value of ν_p and only update the dual variables at the remaining $p - 1$ nodes. We use a constant step size rule since the dual function is differentiable. The algorithm is guaranteed to converge for small enough step sizes.

Figure 21 shows the value of the dual function at each iteration, for three different fixed step sizes, and figure 22 shows the corresponding primal residual $\|Ax+s\|_2$ (which is precisely the norm of the subgradient). The plots suggests that for $\alpha = 3$, the algorithm does not converge to the optimal point. For $\alpha = 1$, the algorithm converges rather well; for example, the primal residual reduces to 4.6×10^{-3} after 100 iterations. Figure 23 shows the convergence of the dual variables for $\alpha = 1$.

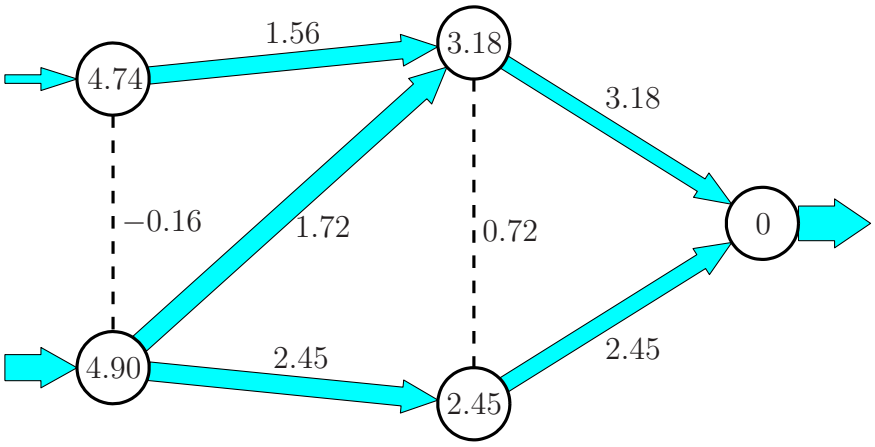

Figure 20: Optimal flows plotted as width of the arrows, and optimal dual variables (potentials). The potential difference $\Delta\nu_j$ is shown next to each link.

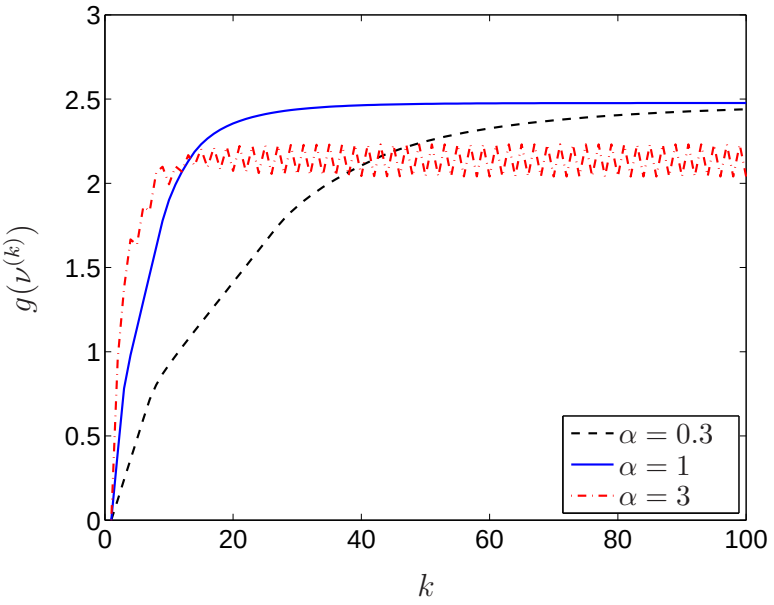

Figure 21: Dual function value versus iteration number k, for the subgradient method with the fixed step size rule.

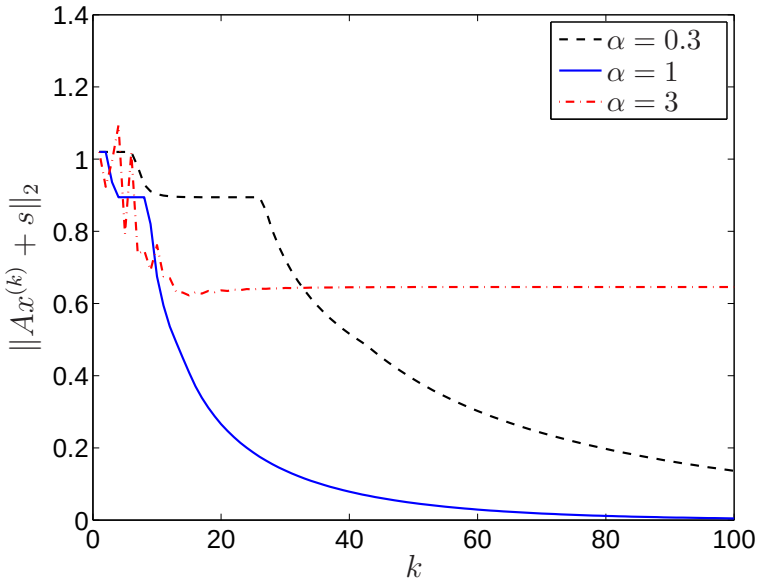

Figure 22: The primal residual $\|Ax + s\|_2$ versus iteration number k, in the subgradient method with the fixed step size.

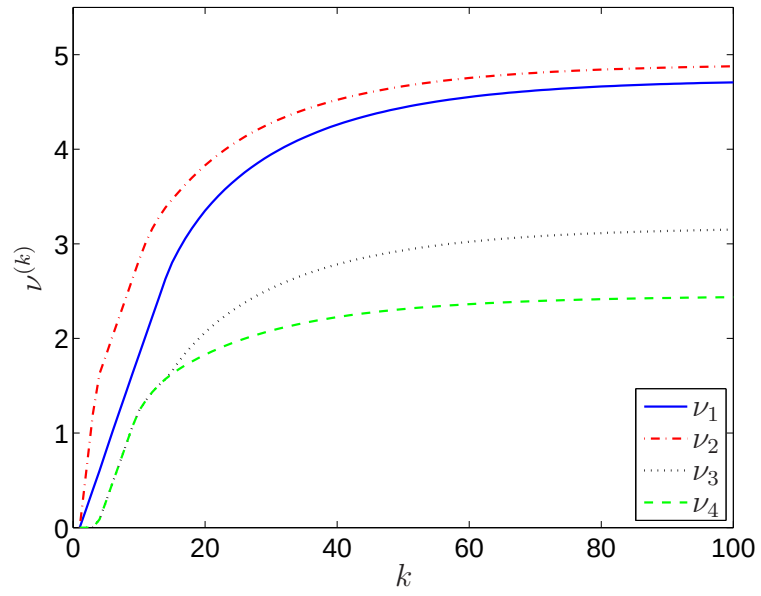

Figure 23: Dual variables $\nu^{(k)}$ versus iteration number k, with fixed step size rule $\alpha = 1$. Note that ν_5 is fixed to zero.

References

[Ber99] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CLCD07] 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*, January 2007. To appear.

[DW60] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[KMT97] F. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1997.