An Interior-Point Method for Large-Scale Network Utility Maximization

Argyrios Zymnis Nikolaos Trichakis
Stephen Boyd Dan O' Neill

Electrical Engineering Department, Stanford University

Basic NUM Problem

primal problem

$$\begin{array}{ll} \text{maximize} & U(f)=\sum_{j=1}^n U_j(f_j) \\ \text{subject to} & Rf \leq c, \quad f \geq 0 \end{array}$$

with variable f

- $f \in \mathbf{R}_+^n$ is vector of flow rates
- $U_j : \mathbf{R} \to \mathbf{R}$ concave and twice differentiable
- $R \in \mathbf{R}^{m \times n}$ is route matrix ($R_{ij} \in \{0,1\}$)
- $c \in \mathbf{R}^m$ is vector of capacities

Dual Decomposition

dual problem

$$\begin{array}{ll} \text{minimize} & \lambda^T c + \sum_{j=1}^n (-U_j)^*(-r_j^T\lambda) \\ \text{subject to} & \lambda \geq 0 \end{array}$$

with variable $\lambda \in \mathbf{R}^m$

- **assume U_j are strictly concave**
- projected subgradient method update:

$$\begin{array}{rcl} \lambda & := & (\lambda - \alpha(c - Rf))_+ \\ f_j & := & \underset{z}{\operatorname{argmax}} \left(U_j(z) - (R^T\lambda)_j z\right) \end{array}$$

- first order method, decentralized

Primal-Dual Interior-Point Method

- basic approach:
  - compute Newton step for (modified) optimality conditions
  - carry out line search and update
- typically converges in a few tens of steps, independent of problem dimensions or data (!!)
- for extremely large-scale problems, compute search direction approximately via iterative method (PCG)
- not decentralized (requires a few inner products each step)

Modified Optimality Conditions

$$\begin{array}{rcl} -\nabla U(f) + R^T\lambda - \mu & = & 0 \\ \mathbf{diag}(\lambda)(c - Rf) & = & (1/t)\mathbf{1} \\ \mathbf{diag}(\mu)f & = & (1/t)\mathbf{1}, \end{array}$$

- $t > 0$ controls quality of approximation
- for $t \to \infty$ we recover the optimality conditions
- write as $r_t(f,\lambda,\mu) = 0$, where

$$r_t(f,\lambda,\mu) = \begin{bmatrix} -\nabla U(f) + R^T\lambda - \mu \\ \mathbf{diag}(\lambda)(c - Rf) - (1/t)\mathbf{1} \\ \mathbf{diag}(\mu)f - (1/t)\mathbf{1} \end{bmatrix}$$

Primal-Dual Algorithm Outline

given initial strictly feasible point ($Rf < c$, $\lambda > 0$, $\mu > 0$)

- update t (based on duality gap estimate)
- compute search direction from $Dr_t \begin{bmatrix} \Delta f \\ \Delta\lambda \\ \Delta\mu \end{bmatrix} = -r_t$
- compute step length γ via line search on $\|r_t\|$
- update variables: $\begin{bmatrix} f \\ \lambda \\ \mu \end{bmatrix} := \begin{bmatrix} f \\ \lambda \\ \mu \end{bmatrix} + \gamma \begin{bmatrix} \Delta f \\ \Delta\lambda \\ \Delta\mu \end{bmatrix}$

Computing the Search Direction

reduces to solving

$$(R^T D R + \tilde{D})\Delta f = b$$

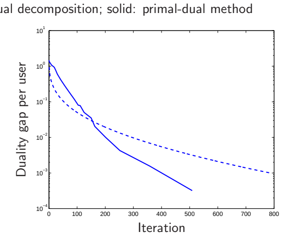
- D, $\tilde{D}$ diagonal, positive
- for 10^4 or fewer links, can solve via direct (sparse) methods
- for very large problems, solve approximately via PCG
  - simple diagonal preconditioner
  - not fully decentralized: requires two inner products per iteration

Simple Example

- $n = 10^5$ users and $m = 2 \times 10^5$ links
- log utilities: $U_j = \log f_j$
- capacities c_i chosen randomly in $[0.1, 1]$
- random routes, each passing through approximately 10 links

Convergence

dashed: dual decomposition; solid: primal-dual method

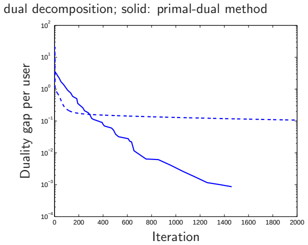


More Interesting Example

- add bottlenecks: 0.1% of links are used by 30% of users
- add long routes: 0.1% of users have route length $\sqrt{m}$
- same as adding some dense(r) rows and columns to R

Convergence

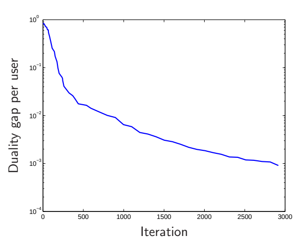
dashed: dual decomposition; solid: primal-dual method



Example with Linear Utilities

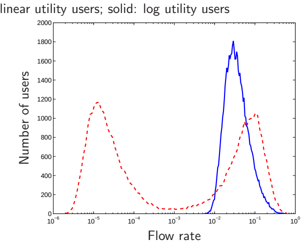
- 60% of users have log utility $U_j = \log f_j$
- 40% of users have linear utility $U_j = w_j f_j$, weights w_j random in $[10, 30]$
- **cannot** solve using dual decomposition
- for users with linear utility, optimal flow rate can be zero

Convergence



Distribution of (Almost) Optimal Flow Rates

dashed: linear utility users; solid: log utility users



Conclusions

- we can reliably and very efficiently solve very large NUM problems, including those with non strictly concave utility (linear, piecewise-linear)
- in many cases, (very much) outperforms dual decomposition
- method is not decentralized; it requires a few inner products each step (but, inner products can be approximately computed in decentralized way via distributed averaging or gossip algorithms)