MCMCF
A Tool for Network Design

Jeffrey D. Oldham
Department of Computer Science
Stanford University
oldham@cs.stanford.edu

1997 September 04
A Multicommodity Flow Example

Specify:
- network topology
- edge costs
- peak call demand

Goal: Satisfy peak demand with minimum cost.

<table>
<thead>
<tr>
<th>Peak Demands</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LA–Dallas</td>
<td>35 calls</td>
</tr>
<tr>
<td>LA–NYC</td>
<td>80 calls</td>
</tr>
<tr>
<td>Dallas–NYC</td>
<td>70 calls</td>
</tr>
</tbody>
</table>

Use: MCMCF (Minimum-Cost MultiCommodity Flow)
Linear Programming Based Solution

Disadvantages:

**Size:** Problem specification: \( O(k + m) \) space

Linear programs: \( O(k(n + m)) \) variables
\( O(kn + m) \) inequalities

- \( n \) is the number of nodes.
- \( m \) is the number of arcs.
- \( k \) is the number of commodities.

**LP solution time:**

- experimentally quadratic in \( k \)
- experimentally quadratic in network size

**design tradeoff:**

- slow, exact solution
- fast approximation
Combinatorial Solution

Combinatorial program MCMCF:

$\epsilon$-approximation:

- flow uses at most $(1 + \epsilon)$ arc capacity
- flow cost at most $(1 + \epsilon)$ minimum cost

Main idea:

- reduce to single-commodity problems
- relate commodities using potential function

Theoretical advantage:

- time: $\tilde{O}(k)$ (time for min-cost flow)
- space: $O(k(n + m))$

Practical advantages:

- trade off time for accuracy
The Potential Function

Problem:
Several objectives:

- minimize total cost
- capacity constraints for every arc

Not smooth!

Solution:
Aggregate into smooth potential function $\phi$

$$
\phi = \exp \left( \alpha \left( \frac{\text{flow's cost}}{\text{desired cost}} \right) \right) + \sum_{\text{arcs } a} \exp \left( \alpha \left( \frac{\text{flow}(a)}{\text{capacity}(a)} \right) \right)
$$

small $\phi \quad \Rightarrow \quad$ good solution
Outline of the Algorithm

Goal: Reduce potential function $\phi$.

Main ideas:
- Move in direction $(-\nabla \phi)$.
- Maintain flow satisfying demands.

Until $\varepsilon$-optimal solution found:
1. Choose a commodity to improve.
2. Compute $\nabla \phi$.
3. Use $\nabla \phi$ as arc costs.
5. Improvement step: $(1 - \sigma)f + \sigma f^*$.
The Algorithm’s Running Time

The theoretical running time is

\[ \tilde{O}(\epsilon^{-3}k) \text{ (time for min-cost flow)} \]

[Karger and Plotkin, 1995],
[Plotkin, Shmoys, Tardos, 1995],
[Leighton et al., 1995].

Advantages:

- (almost) linear dependence on number \( k \) of commodities
- uses well-understood single-commodity flow subroutine
The Algorithm’s Running Time (cont’d)

Direct implementation runs slower than LP.

Problem:

- pessimistic parameters
- guarantee progress but not practical progress

Solution:

- dynamically adjust parameters

Key Idea:

- use theory to yield practical modifications
Problem Instances

Problem instances from two different families:

**multigrid**
- two-dimensional grids
- few additional arcs

**rmfgen**
- series of two-dimensional grids
- connect grids via random node permutation

Commodity sources, sinks, demands randomly chosen.
Choosing the Step Size $\sigma$

Improvement step:

$$(1 - \sigma)f + \sigma f^*.$$ 

Theory:

- fixed step size $\sigma = O(\epsilon^{-3})$

Practice:

- Compute $\sigma$ to minimize potential function.
- Use Newton-Raphson method.
- Newton requires first and second derivatives.

Result: (Sun Enterprise 3000)

<table>
<thead>
<tr>
<th>instance</th>
<th>$\epsilon$</th>
<th>time (seconds)</th>
<th>Newton</th>
<th>theoretical</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmfgen-d-4-12-020</td>
<td>0.01</td>
<td>64</td>
<td></td>
<td>3842</td>
</tr>
<tr>
<td>rmfgen-d-7-10-020</td>
<td>0.01</td>
<td>257</td>
<td></td>
<td>15203</td>
</tr>
<tr>
<td>multigrid-008-016-0100</td>
<td>0.01</td>
<td>3</td>
<td></td>
<td>95</td>
</tr>
</tbody>
</table>
Choosing $\alpha$

Constant $\alpha$ in potential function:

$$\phi = \exp \left( \alpha \left( \frac{\text{flow's cost}}{\text{desired cost}} \right) \right) + \sum_{\text{arcs } a} \exp \left( \alpha \left( \frac{\text{flow}(a)}{\text{capacity}(a)} \right) \right)$$

**Theory:**

- fixed (large) value $\Rightarrow$ guarantee progress
- progress inversely proportional to $\alpha$

**Practice:**

- choose (smaller) value guaranteeing progress
- compute occasionally—expensive

**Result:** (Sun Enterprise 3000)

<table>
<thead>
<tr>
<th>instance</th>
<th>$\epsilon$</th>
<th>time (seconds)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>adaptive</td>
<td>theoretical</td>
<td></td>
</tr>
<tr>
<td>rmfgen-d-7-10-020</td>
<td>0.01</td>
<td>56</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>rmfgen-d-7-10-240</td>
<td>0.03</td>
<td>238</td>
<td>738</td>
<td></td>
</tr>
<tr>
<td>multigrid-032-128-0080</td>
<td>0.01</td>
<td>42</td>
<td>47</td>
<td></td>
</tr>
</tbody>
</table>
Updating MCF Routine

Theory:

- Use any minimum-cost flow routine.

Practice:

- Costs and capacities do not vary much.
- Simplex MCF can update from feasible flow.
- Use commodity’s current flow.

Result: (Sun UltraSPARC-2)

<table>
<thead>
<tr>
<th>instance</th>
<th>$\epsilon$</th>
<th>time (seconds)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>updating</td>
<td>no updating</td>
</tr>
<tr>
<td>rmfgen-d-7-10-020</td>
<td>0.01</td>
<td>87</td>
<td>180</td>
</tr>
<tr>
<td>rmfgen-d-7-10-240</td>
<td>0.01</td>
<td>454</td>
<td>835</td>
</tr>
<tr>
<td>multigrid-032-128-0080</td>
<td>0.01</td>
<td>21</td>
<td>37</td>
</tr>
</tbody>
</table>
**Small Incremental Flow Change**

**Theory:**
- Flow can change on all arcs.

**Practice:**
- Flow changes on few arcs.
- Routines for $\sigma$ use only nonzero differences.

**Result: (Sun UltraSPARC-2)**

<table>
<thead>
<tr>
<th>instance</th>
<th>$\epsilon$</th>
<th>time (seconds)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>use nonzero</td>
<td>use all</td>
<td></td>
</tr>
<tr>
<td>rmfgen-d-7-10-020</td>
<td>0.01</td>
<td>87</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>rmfgen-d-7-10-240</td>
<td>0.01</td>
<td>454</td>
<td>972</td>
<td></td>
</tr>
<tr>
<td>multigrd-032-128-0080</td>
<td>0.01</td>
<td>21</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>
Termination Criteria

Stop algorithm when have $\epsilon$-optimal solution.

Theory:

- small $\phi \implies \epsilon$-optimal

Practice:

- Compute a lower bound using LP dual.
- Compute occasionally—$k$ MCF computations
Comparisons with Linear Programming

MCMCF

$\varepsilon$-approximation:
- Flow uses at most $(1 + \varepsilon)$ arc capacity
- Flow cost at most $(1 + \varepsilon)$ minimum cost

CPLEX

dual simplex: exact solutions
primal simplex
- permits stopping to yield $\varepsilon$-approximation
- experimentally 10x slower than dual
Dependence on $k$

Multigrid Instances

![Graph showing the dependence of running time on the number of commodities. The graph plots the running time (in minutes) against the number of commodities.]
Dependence on $k$ (cont’d)

Rmfggen Instances

![Graph showing running time vs. number of commodities for CPLEX and MCMCF]
Dependence on Problem Size

The tripartite generator was designed to produce problems difficult for MCMCF to solve.

Tripartite Instances

![Graph showing running time against problem size with CPLEX and MCMCF (2%) compared.](image)
Dependence on the Approximation $\epsilon$

The dependence is approximately $O(\epsilon^{-1.5})$. 

![Graph showing the relationship between $1/\epsilon$ and the number of MCF Computations for RMfgen Instances. The graph plots lines that illustrate a logarithmic relationship, increasing as $1/\epsilon$ decreases.]
Conclusions

**theoretical algorithm**

- theoretically fast
- practically slower than LP

**practical modifications**

- guided by theory
- yield fast, provably correct implementation

**resulting advantages**

- faster than all other algorithms
- solve larger problems than all other algorithms
- fast approximations—good for design
- trade time for accuracy