The Primal-Dual Algorithm (reminder)

Primal-Dual Algorithm

Input: Graph $G = (V, A)$, capacities $u$, excess values $b$, costs $c$

- Construct a pair of initial solutions $x$, $y$.
- While $x$ is not feasible:
  - If there exists an $x$-augmenting path $P$ of equality arcs:
    - Determine the width of the path
    - Augment the flow $x$ along $P$
  - Otherwise:
    - Find a vertex set $R$ blocking all such paths, and change $y_v$ for all $v \in V \setminus R$
      (as described on page 18–12)

We were not happy with this algorithm because it seems we may need quite a number of dual steps (change of potentials) until we can make the next primal step (sending flow from an $x$-source to an $x$-sink)
To be more precise: Because each dual step increases the size of the blocking set $R$ by at least one vertex, at most $n - 1$ dual steps are necessary.

For integer-valued data, it is clear that each primal step (augmenting flow) decreases the imbalance by at least 1; so the number of augmentations is bounded by the initial imbalance

$$B_{x^0} := \sum_v \max\{0, b_v - f_{x^0}(v)\},$$

where $x^0$ is the initial feasible solution.

For non-negative costs, we could start with the zero flow $x^0 = 0$, so we have at most

$$B_0 = \sum_v \max\{0, b_v\}$$

augmentations.

So again, we will get a **pseudo-polynomial algorithm** of running time $O(S(n, m) \cdot n \cdot B_{x^0})$, where $S(n, m)$ is the running time of a shortest-path computation.

(Knowing this more precisely does not make us happier, though.)
This observation suggests a new algorithm, due to Busacker–Gowen [1961]

**Primal-Dual Algorithm with Least-Cost Augmenting Paths**

Input: Graph $G = (V, A)$, capacities $u$, excess values $b$, costs $c$

- **Construct a pair of initial solutions $x$, $y$.**
- **While $x$ is not feasible:**
  - Find a least-cost (with respect to reduced costs $\tilde{c}$) $x$-incrementing path $P_v$ from an $x$-source to $v$, for each $v \in V$ (**one nonnegative-cost shortest-path-tree calculation** in a graph with an artificial source); denote by $\sigma_v$ the costs of the paths.
  - Choose an $x$-sink $s$ such that $\sigma_s$ is minimum
  - Update the potentials $y_v := y_v + \min\{\sigma_v, \sigma_s\}$ for $v \in V$.
  - Augment $x$ on $P_s$.

**Lemma**

*This algorithm maintains the optimality conditions on $x$ and $y$ in each step.*
Efficiency of the Algorithm, Initial Feasible Solution

- Because the dual update can be done in one step, using a single shortest-path-tree computation, this is quite a bit faster. The running time reduces to $O(S(n, m) \cdot n \cdot B_{x^0})$.

**How do we construct a pair of initial solutions, by the way?**

- If all costs are non-negative, can use $x = 0$, $y = 0$.
- We could try to set $y = 0$ (or arbitrary), and set $x_{v,w} = u_{v,w}$ if $\bar{c}_{v,w} < 0$ and $x_{v,w} = 0$ to satisfy the optimality conditions. However, this fails if some $u_{v,w} = \infty$.

**General solution:** (updated)

- Solve a maximum-flow problem to find out whether there is a feasible flow; discard the solution.
- Solve a shortest path problem (in a directed graph $G^\infty$ that only has the arcs with infinite capacities, using the original costs $c$).
- If there is no feasible shortest-path potential, there exists a negative-cost directed cycle of infinite capacity; so the problem is unbounded (no optimal solution).
- Otherwise, we obtain a feasible shortest-paths potential $y$ on $G^\infty$; so we have $y_w \leq y_v + c_{v,w}$ for all $(v, w)$ with $u_{v,w} = \infty$.
- We use this $y$ as the initial potential. From the above inequality we have $\bar{c}_{v,w} \geq 0$ for all arcs $(v, w)$ with $u_{v,w} = \infty$.
- Now set $x_{v,w} = u_{v,w}$ if $\bar{c}_{v,w} < 0$ and $x_{v,w} = 0$. (Note that no $x_{v,w}$ will be infinite.)
By a **scaling technique** (where demands $b_v$ are replaced by $\lfloor b_v/2^k \rfloor$), Edmonds–Karp [1972] obtained a **polynomial-time variant**. The running time is $O(n \cdot S(n, m) \cdot (1 + \log \max\{B_0, U\}))$, where $U$ is the largest finite arc capacity.

The **scale-and-shrink algorithm** (following from work by Tardos [1985], Orlin [1985], Fujishige [1986]) is a **strongly polynomial-time** variant, with a running time of $O((m_0 + n)n \log n \cdot S(n, m))$. 
There’s much more of optimization to learn!

We have only scratched the surface…

- MAT-168 (Spring 2009) – Linear Programming
- 2009/2010: Year-long program (VIGRE RFG) on optimization:
  - Optimization seminar
  - Reading courses
  - 258A (Fall 2009) – Numerical Optimization
  - 258B (Winter 2010) – Variational Analysis and Mixed-Integer Nonlinear Programming
  - 280 (Spring 2010) – Integer Programming