Almost Linear Time Algorithms for Flows in Graphs

Victor Sanches Portella[1]
Supervisor: prof. Marcel Kenji de Carli Silva[1]

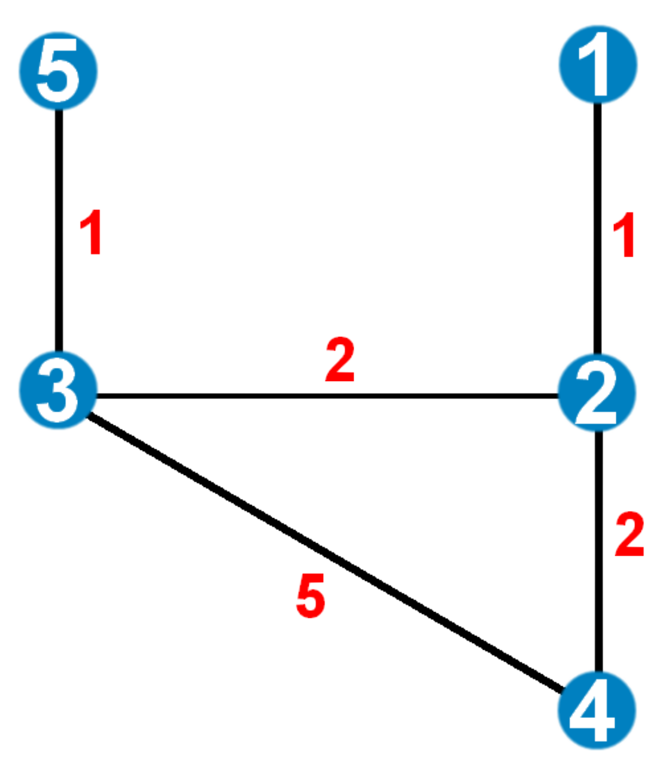
[1] Institute of Mathematics and Statistics - University of São Paulo

The Challenge of *Big Data*

- Huge data sets are becoming more and more common;
- Using quadratic algorithms on such data sets is already impractical;
- This trend is motivating research of nearly-linear time approximation algorithms for a host of combinatorial problems.

Laplacian Systems

We denote the **Laplacian matrix** of G with positive weights $w \in \mathbb{R}^E$ by $\mathcal{L}_G(w)$.



$$\mathcal{L}_G(w) = \begin{pmatrix} \enclose{circle}{1} & \boxed{-1} & 0 & 0 & 0 \\ \boxed{-1} & \enclose{circle}{5} & \boxed{-2} & \boxed{-2} & 0 \\ 0 & \boxed{-2} & \enclose{circle}{8} & \boxed{-5} & \boxed{-1} \\ 0 & \boxed{-2} & \boxed{-5} & \enclose{circle}{7} & 0 \\ 0 & 0 & \boxed{-1} & 0 & \enclose{circle}{1} \end{pmatrix}$$

Edge entry i, j: minus its weight

Diagonal entry i, i: sum of weights incident to i

A **Laplacian system** is a linear system of the type $Lx = b$, where L is the Laplacian of a graph. A major breakthrough in the area of nearly-linear time algorithms was Spielman and Teng's nearly linear time Laplacian solver, which solves a Laplacian system in time $\tilde{O}(m) = O(m \log^c m)$ for some constant $c \in \mathbb{N}$, where m is the number of edges.

Electrical and Feasible Flows

Let $G = (V, E)$ be a graph, and let $s, t \in V$ be distinct. Then $f \in \mathbb{R}^E$ is an (s,t)-**flow** in G if it obeys the flow-conservation constraints, that is,

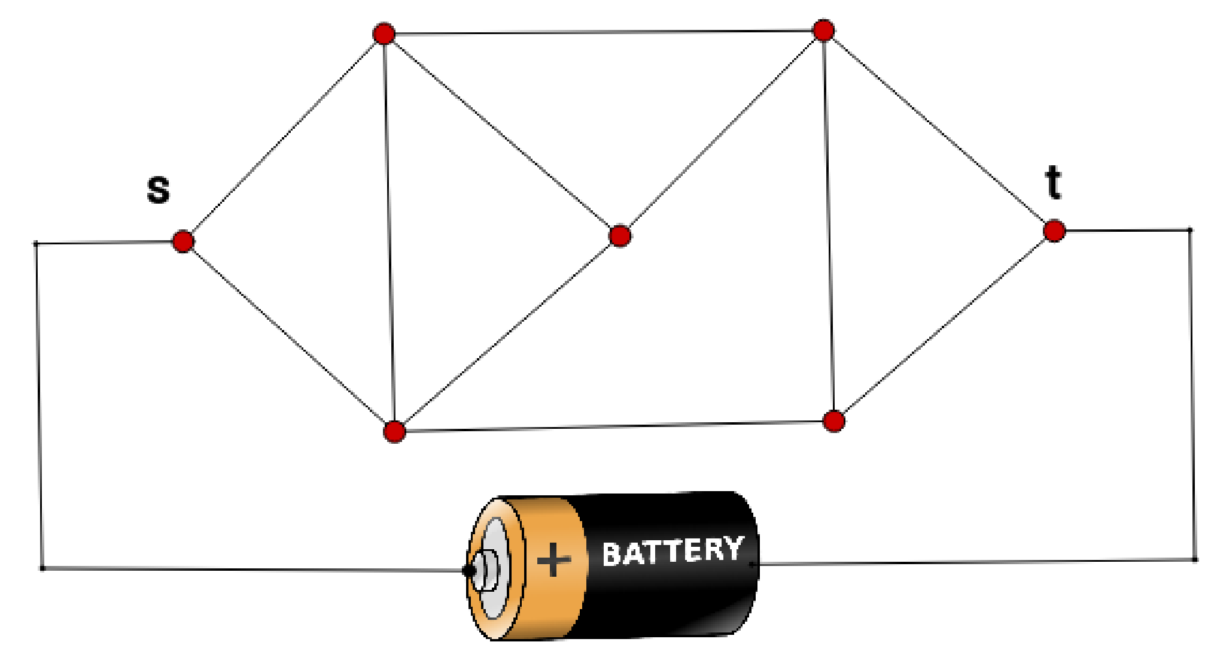
$$\sum_{e \in \delta(v)} f_e = 0 \qquad \forall v \in V \setminus \{s, t\}.$$

The value of a flow is the total amount of flow leaving s. If positive capacities $u \in \mathbb{R}^E$ are associated to the edges, we say that a flow is **feasible** if $|f_e| \leq u_e$ for each $e \in E$. If $G = (V, E, r)$ is a weighted graph with positive weights $r \in \mathbb{R}^E$, the **energy** of an (s,t)-flow f in G is

$$\mathcal{E}(f) := \sum_{e \in E} f_e^2 r_e.$$

An (s,t)-flow of value $\alpha \in \mathbb{R}_+$ is **electrical** if it minimizes the energy among all (s,t)-flows in G of value α. Surprisingly, such a flow can be computed from a solution of a Laplacian system.

Physical Intuition Behind Electrical Flows



One may imagine a weighted graph $G = (V, E, r)$ as the representation of an electrical network, where each $e \in E$ is a resistor with resistance r_e, and the vertices represent the junction of many resistors. Then, an electrical (s,t)-flow represents the currents passing through the resistors when connecting s and t to the poles of an external current source.

An $\tilde{O}(m^{3/2}\varepsilon^{-5/2})$ Algorithm for Approximately Maximum Flow

Approximately Electrical Flows in Nearly-linear Time

Since we can find an electrical flow by solving a Laplacian system, a natural idea is to use approximate Laplacian solvers to efficiently compute electrical flows. One problem that arises is that the solution that the solver yields may not satisfy the flow conservation constraints. However, such a vector is close to being a flow. Hence, we may "round" it to a flow without increasing by much its energy. This process yields a flow that is **approximately electrical**.

Boosting a Crude Approximation

The Multiplicative Weights Update (MWU) method is a meta-algorithm that can be seen as a way of computing a good solution to a problem by using many crude approximations of. The general and intuitive framework is the following:

- There are n decisions, and a *decision maker* repeatedly needs to choose a decision and get an associated payoff.
- The objective of the *decision maker* is to obtain a total sum of payoffs near to the maximum payoff of the case of choosing a fixed decision with the benefit of hindsight.

The MWU method achieves such a guarantee by maintaining weights over the decisions, and at each round it proceeds as follows:

- It randomly picks a decision with probability proportional to its weights;
- After the payoffs are revealed, the method updates the weights of each decision by a multiplicative factor which depends on the payoff, penalizing bad decisions.

Feasible Flows Through Electrical Flows

The following pseudo-code describes, in a high-level, the algorithm that finds a feasible flow of value at least $(1 - 3\varepsilon)\alpha$ using electrical flows and the MWU method. Moreover, this algorithm runs in time $\tilde{O}(m^{3/2}\varepsilon^{-5/2})$, where m is the number of edges in the graph.

```
Input: A graph $G = (V, E)$ with capacities $u \in \mathbb{R}^E_+$, vertices $s, t \in V$, and $\alpha \in \mathbb{R}_+$.
Output: It returns a feasible $(s,t)$-flow of $G$ of value at least $(1 - 3\varepsilon)\alpha$.
  $w^0 \leftarrow \mathbb{1} \in \mathbb{R}^E$
  Let $C \in O(\varepsilon^{-1})$ be a constant.
  for $i = 1$ to $N$ do
      Let $r_e^i \in O(w_e/u_e^2)$ for each $e \in E$.
      Compute an approximately electrical flow $f^i$ in $(V, E, r)$ of value $\alpha$.
      for all $e \in E$ do
          $w_e^i \leftarrow w_e^{i-1}(1 + C(f_e/u_e))$.
  return The average of all $f^i$.
```

Further information

For more information, see www.linux.ime.usp.br/~victorvsp/mac0499/. This works is partially supported by São Paulo Research Foundation (FAPESP) grant n° 2015/24747-3

LaTeX TikZposter