Humboldt-Universität zu Berlin
Department of Computer Science
Modeling and Analysis of Complex Systems

Faster Optimization of Resistance-based Graph Robustness

Maria Predari[1], Rob Kooij[2], Henning Meyerhenke[1]

[1] Humboldt-Universität zu Berlin, Germany

[2] TU Delft, The Netherlands

NAS Group Seminar, TU Delft, The Netherlands – March 2023

Proc. ASONAM '22

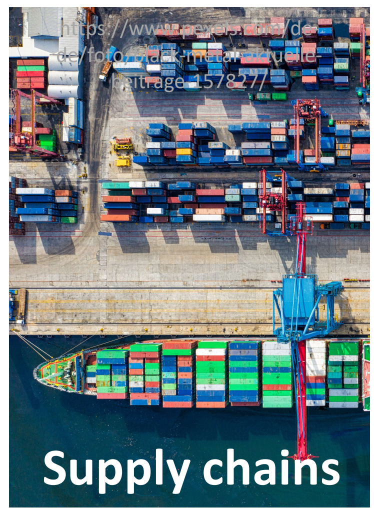
Robustness in Networks

Introduction and Motivation

- Networks appear in many applications
- Robustness a major issue (**failures/attacks**) → are there (many) alternative routes?

Computer networks

[https://sites.google.com/site/technologynattayac/network-system]



k-GRIP with Total Graph Resistance

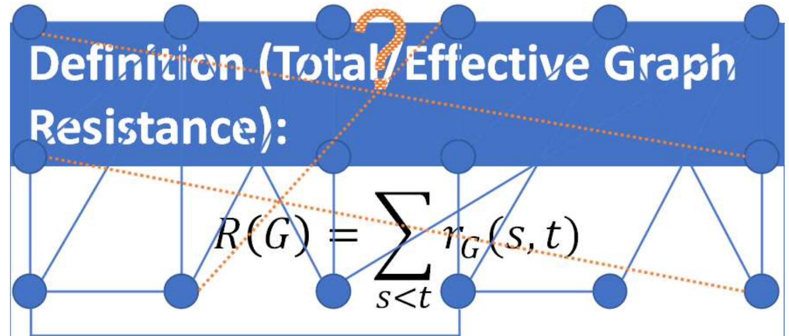
Definition:

Given a graph $G = (V, E)$ and a budget of k edges to be added, find a set $S \subset \binom{V}{2} \setminus \mathrm{E}$ of size k whose addition optimizes the robustness of G.

Notions of Robustness:

- Structural: vertex/edge connectivity
- Spectral: Fiedler vector
- Centrality: betweenness, closeness, …
- Functional: service-/process-oriented

Definition (Total/Effective Graph Resistance):



$$R(G) = \sum_{s<t} r_G(s,t)$$

Definition (Effective Resistance):

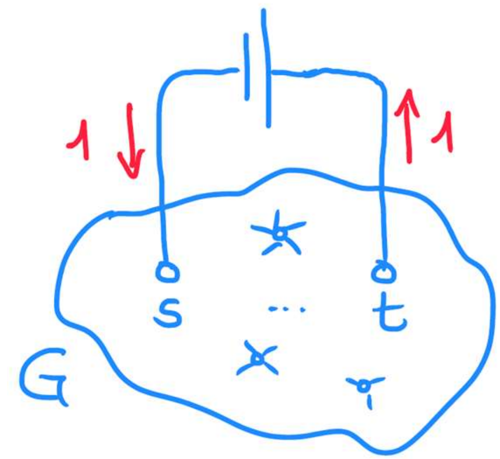
$$r_G(s,t) = l^+_{s,s} - 2l^+_{s,t} + l^+_{t,t}$$

Theorem (Kooij, Achterberg):

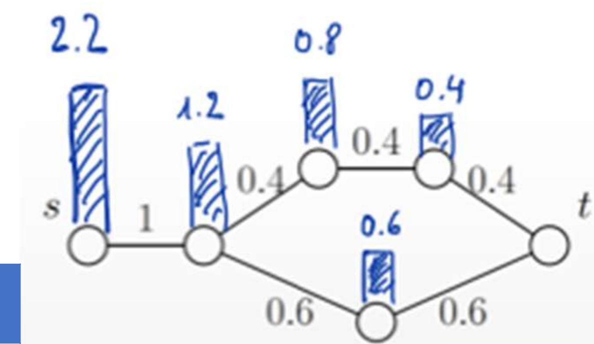
k-GRIP with $R(G)$ is NP-hard.

Robustness via Effective Resistance

- Alternative metric / distance measure: $r_G(\cdot,\cdot)$
- Considers all possible paths (shorter ones more important)
- Spectral connections



- Graph as electrical network
- Ohm's law: $U = R \cdot I$
- Kirchhoff's laws $\rightsquigarrow Lx = e_s - e_t$



Definition (Effective Resistance):

$$r_G(s,t) = x(s) - x(t) = (e_s - e_t)^T L^+ (e_s - e_t) = l^+_{s,s} - 2l^+_{s,t} + l^+_{t,t}$$

SotA: Greedy Framework

StGreedy [Summers et al., ECC 2015]; Complexity: $O(kn^3)$

Algorithm 1 General framework for k-GRIP

function GreedyFramework(G, k, ϵ)
    Input: Graph $G = (V, E)$, $k \in \mathbb{Z}_{>0}$, accuracy $0 < \epsilon < 1$
    Output: G_k – graph after k edge insertions
    $G_0 \leftarrow G$
    ComputeObj(G_0, ...) ▷ compute step -- compute: $O(n^3)$
    $s \leftarrow$ CandidateSize(m, n, k, ϵ)
    for $r \leftarrow 0, \dots, k-1$ **do** ▷ main loop -- total time: $O(kn^3)$
        $\mathcal{S} \leftarrow$ Candidates(s, G_r, ...)
        for each $\{a, b\} \in \mathcal{S} \times \mathcal{S}$ **do** ▷ # of evaluations -- total # evals: $O(kn^2)$
            gain(a, b) $\leftarrow$ Eval(a, b, ...) ▷ single evaluation -- single eval: $O(n)$
        $(a^*, b^*) \leftarrow \text{argmax}_{a \in S \times b \in S}\, \text{gain}(a, b)$
        $G_{r+1} = G_r \cup (a^*, b^*)$
        Update(G_{r+1}, ...) ▷ update step -- update: $O(n^2)$
    return G_{r+1}

State of the Art (SotA)

for GRIP with graph resistance

- 1-GRIP with graph resistance:
  - Simple combinatorial and more involved spectral heuristics [Wang et al., Europ. Phys. J. 2014]
  - Genetic algorithm: $O(n^5)$ time [Pizzuti & Socievole, Complex Networks 2018]
- k-GRIP with res. is **not submodular**
- Still: greedy algorithm works very well in practice [Summers et al., ECC 2015]
- For **submodular** problems: stochastic greedy algorithm works well and fast, provides approximation guarantee

Resulting Research Questions:

- Does stochastic greedy work well for us, too? **(accuracy)**
- How to select good candidates quickly? **(speed)**

Contribution

Acceleration of greedy optimization

- 3-4 ways of candidate selection, according to:
  - Stochastic greedy
  - Columns of L^+
  - Random projection (JLT)
  - Spectral approximation with low-rank technqiues

- Solution quality often mostly preserved
- **Example:** 2-15% away from greedy solution, but $3.3 - 68\times$ faster
- Much larger graphs can be processed than with SotA

Algorithmic Approach 0

Simple stochastic greedy (SIMPLSTOCH)

- Initial comp.: L^+
- Candidates S: size only
  $s := \frac{n^2-m}{k} \log\left(\frac{1}{\delta}\right)$
- Function evaluations via fast L^+ updates:

$$\mathbf{L}^{\dagger}_{G'} = \mathbf{L}^{\dagger}_{G} - \frac{1}{1+\mathbf{r_G}(a,b)} \mathbf{L}^{\dagger}_{G}(\mathbf{e}_a - \mathbf{e}_b)(\mathbf{e}_a - \mathbf{e}_b)^T \mathbf{L}^{\dagger}_{G}.$$

- Savings on GREEDY: factor of $k/\log(1/\delta)$

Algorithm 1 General framework for k-GRIP

```
function GREEDYFRAMEWORK(G, k, ε)
    Input: Graph G = (V, E), k ∈ Z_{>0}, accuracy 0 < ε < 1
    Output: G_k – graph after k edge insertions
    G_0 ← G
    COMPUTEOBJ(G_0, ...)                          -- compute: O(n^3)
    s ← CANDIDATESIZE(m, n, k, ε)
    for r ← 0, ..., k − 1 do                      -- total time: Õ(n^3)
        S ← CANDIDATES(s, G_r, ...)
        for each {a, b} ∈ S × S do                ▷ # of evaluations
            gain(a, b) ← EVAL(a, b, ...)          -- eval: O(n)
        (a*, b*) ← argmax_{a∈S×b∈S} gain(a, b)
        G_{r+1} = G_r ∪ (a*, b*)
        UPDATE(G_{r+1}, ...)                      -- update: O(n^2)
    return G_{r+1}
```

Algorithmic Approach 1

ColStoch

$$s = n\sqrt{\frac{1}{k} \cdot \log(\frac{1}{\delta})}$$

- Avoids pseudoinversion of L, uses diag. approx.
- Vertex sampling: probabilities derived from approximate electrical closeness
- Linear system solved for each candidate vertex
- Generate and update columns of L^+ on demand

Algorithm 1 General framework for k-GRIP

function GreedyFramework(G, k, ϵ)
    Input: Graph $G = (V, E)$, $k \in \mathbb{Z}_{>0}$, accuracy $0 < \epsilon < 1$
    Output: G_k – graph after k edge insertions
    $G_0 \leftarrow G$
    ComputeObj(G_0, ...) -- compute: $\tilde{O}(sm \log n)$
    $s \leftarrow$ CandidateSize(m, n, k, ϵ)
    for $r \leftarrow 0, \ldots, k-1$ **do** -- total time: $\tilde{O}(n^3)$
        $\mathcal{S} \leftarrow$ Candidates(s, G_r, ...)
        for each $\{a, b\} \in \mathcal{S} \times \mathcal{S}$ **do** ▷ # of evaluations
            gain$(a, b) \leftarrow$ Eval(a, b, ...) -- eval: $O(n)$
        $(a^*, b^*) \leftarrow \text{argmax}_{a \in S \times b \in S}$ gain(a, b)
        $G_{r+1} = G_r \cup (a^*, b^*)$
        Update(G_{r+1}, ...) -- update: $\tilde{O}(sm \log n)$
    return G_{r+1}

Algorithmic Approach 2

*STOCHJLT: SIMPLSTOCHJLT AND COLSTOCHJLT

$$s = n\sqrt{\frac{1}{k} \cdot \log(\frac{1}{\delta})}$$

- Gain computation needs two

$$\mathbf{r}_{\mathbf{G}}(a,b) = \left\|\mathbf{B}\mathbf{L}^{\dagger}(\mathbf{e}_a - \mathbf{e}_b)\right\|^2$$

$$\mathbf{b}_{\mathbf{G}}(a,b) = \left\|\mathbf{L}^{\dagger}(\mathbf{e}_a - \mathbf{e}_b)\right\|^2$$

- JLT: random projection for dimensionality reduction with matrices P and Q, $Y = L^{+}P^{T}$
- $O(\log s)$ linear systems
- Also works with stochastic greedy (SIMPLSTOCHJLT)

Algorithm 1 General framework for k-GRIP

```
function GREEDYFRAMEWORK(G, k, ε)
    Input: Graph G = (V, E), k ∈ Z_{>0}, accuracy 0 < ε < 1
    Output: G_k – graph after k edge insertions
    G_0 ← G
    COMPUTEOBJ(G_0, ...)                        -- compute: Õ(m log n log s)
    s ← CANDIDATESIZE(m, n, k, ε)
    for r ← 0, ..., k − 1 do                    -- total time: Õ(n^2 log n)
        S ← CANDIDATES(s, G_r, ...)
        for each {a, b} ∈ S × S do              ▷ # of evaluations
            gain(a, b) ← EVAL(a, b, ...)        -- eval: O(log s)
        (a*, b*) ← argmax_{a∈S×b∈S} gain(a, b)
        G_{r+1} = G_r ∪ (a*, b*)
        UPDATE(G_{r+1}, ...)                    -- update: Õ(m log n log s)
    return G_{r+1}
```

$$\text{gain}(a,b) \approx \frac{\left\|\mathbf{P}\mathbf{L}^{\dagger}(\mathbf{e}_a - \mathbf{e}_b))^2\right\|}{1 + \left\|\mathbf{QBYP}(\mathbf{e}_a - \mathbf{e}_b)^2\right\|}$$

Algorithmic Approach 3

SPECSTOCH

- Express gain function by spectral decomposition
- Approximation with small eigenvalues and corresponding eigenvectors, plus largest eigenvalue
- c eigenpairs, computed with Lanczos
- New upper and lower bounds for gain
- Bootstrapping for eigenpair updates

Algorithm 1 Spectral-based stochastic greedy for k-GRIP

function SPECSTOCH(G, k, δ, c)
  Input: Graph $G = (V, E)$, $k \in \mathbb{Z}_{>0}$, accuracy $0 < \delta < 1$, cut-off $c \geq 1$
  Output: G_k – graph after k edge insertions
  $G_0 \leftarrow G$
  $\{\Lambda_{G_0}, U_{G_0}\} \leftarrow$ EIGS($\mathbf{L}_{G_0}$, c) -- compute: $\tilde{O}(cm)$
  $s \leftarrow \frac{n^2-m}{k} \cdot \log \frac{1}{\delta}$
  for $r \leftarrow 0, \ldots, k-1$ **do** -- total time: $\tilde{O}(cn^2)$
    $\mathcal{S} \leftarrow$ CANDIDATES(s, G_r, RANDOM) $\subset V \times V \setminus E$
    for each $\{a, b\} \in \mathcal{S}$ **do** ▷ total # evals – $\mathcal{O}(n^2)$
      gain$(a,b) \leftarrow$ APPROXBOUND($\Lambda_{G_r}, U_{G_r}, c$) ▷ single eval – $\mathcal{O}(c)$
    $(a^*, b^*) \leftarrow \text{argmax}_{a,b \in \mathcal{S}}\, \text{gain}(a, b)$
    $G_{r+1} = G_r \cup (a^*, b^*)$
    $\{\Lambda_{G_{r+1}}, U_{G_{r+1}}\} \leftarrow$ EIGSBOOTSTRAP($\mathbf{L}_{G_{r+1}}$, c) -- update: $\tilde{O}(cm)$
  return G_{r+1}

$$\text{gain}(a,b) = \frac{\sum_{i=2}^{n} \frac{1}{(\lambda_i)^2} \cdot (\mathbf{u_i}[a] - \mathbf{u_i}[b])^2}{1 + \sum_{i=2}^{n} \frac{1}{\lambda_i} \cdot (\mathbf{u_i}[a] - \mathbf{u_i}[b])^2}$$

Experiments

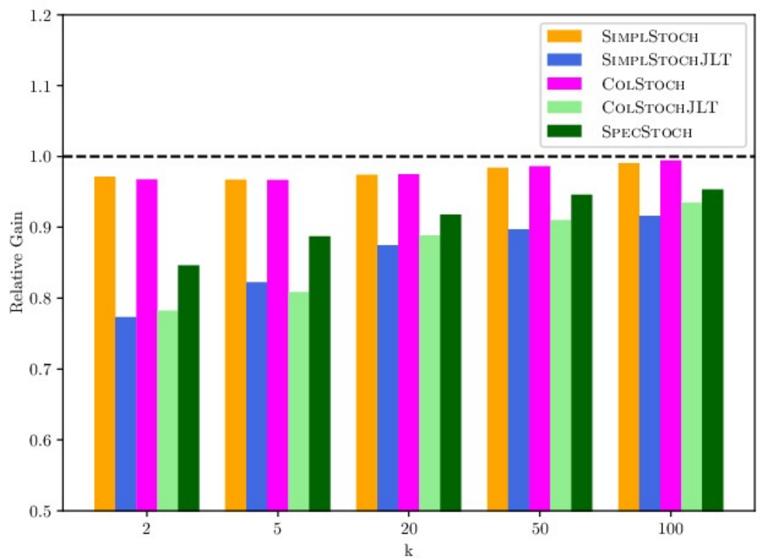
Setup

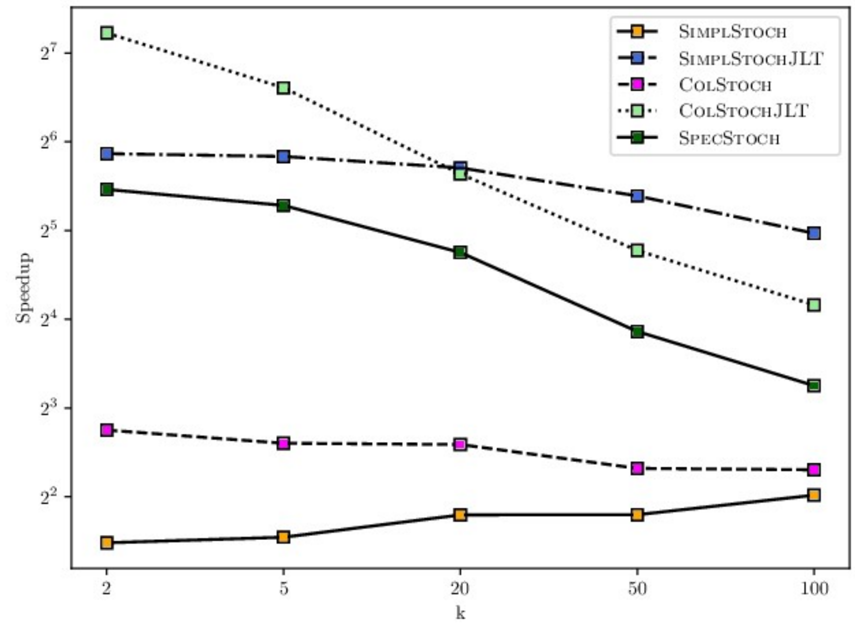
- C++ implementation using NetworKit
- Laplacian systems via LAMG (NetworKit) and SparseLU (Eigen); eigensolver: Slepc
- StGreedy: same quality as exhaustive for synthetic instances
- Real-world instances in table

Graph	$\|V\|$	$\|E\|$
inf-power	4K	6K
facebook-ego-combined	4K	8.8K
web-spam	4K	37K
Wiki-Vote	7K	100K
p2p-Gnutella09	8K	2.6K
p2p-Gnutella04	10K	39K
web-indochina	11K	47K
ca-HepPh	11K	117K
web-webbase-2001	16K	25K
arxiv-astro-ph	17K	196K
as-caida20071105	26K	53K
cit-HepTh	27K	352K
ia-email-EU	32K	54.4K
loc-brightkite	57K	213K
soc-Slashdot0902	82K	504K
ia-wiki-Talk	92K	360K
flickr	106K	2.31M
livemocha	104K	2.19M
road-usroads	129K	165K

Experiments

Medium-sized real-world instances (n < 57K)

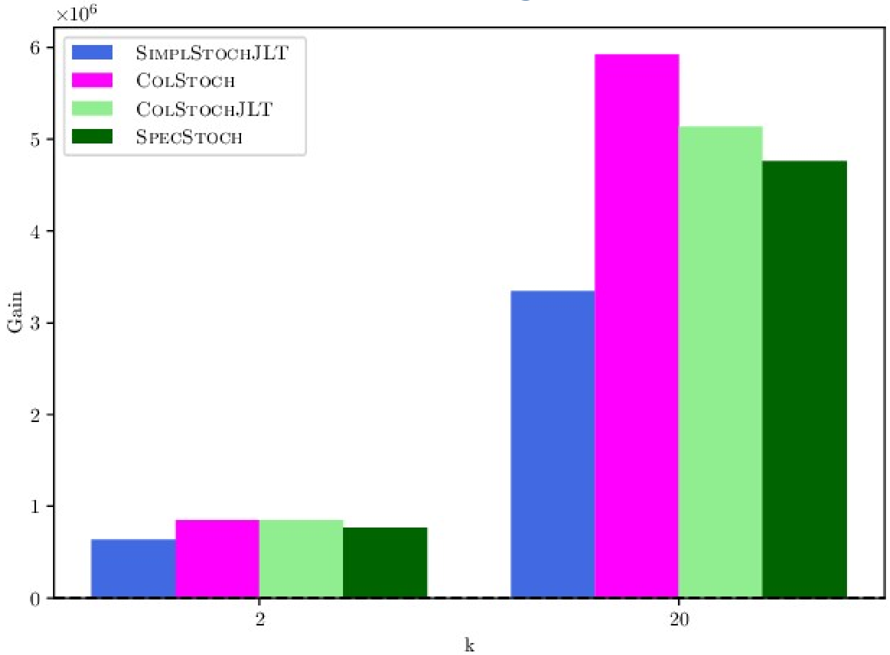




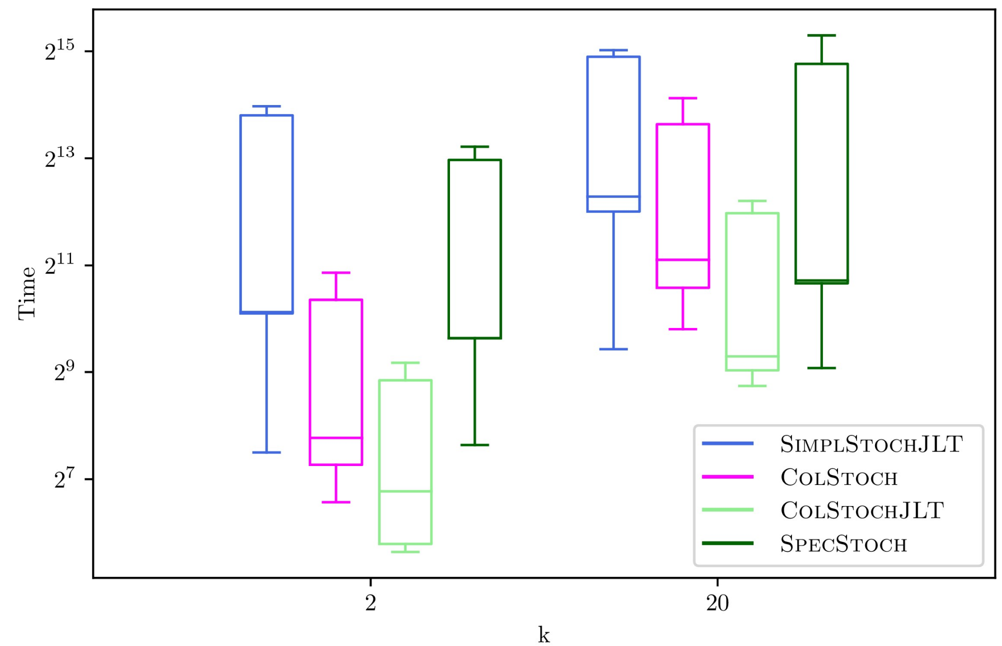
- ColStoch close to Greedy in quality (2% off), 6x faster
- *JLT and SpecStoch much faster, but with some quality loss
- ”Best” tradeoff: SpecStoch (9% off, 26x faster)

Experimental Results: Large Graphs

Absolute gain



Absolute running time



- Best approaches for large graphs: ColStochJLT, ColStoch
- ColStochJLT fastest, on average 2 and 20 minutes for k=2,20

Conclusions

- Robustness in networks has many applications
- k-GRIP: add k edges to optimize robustness; here: resistance

- **Contributions / Results:**
  - Stochastic greedy algorithm: faster than SotA, similar quality
  - Faster selection of candidates: three strategies (others possible)
  - Different tradeoffs: ColStoch already 6x faster, low quality loss

- **Future work:**
  - Meaningful bounds: exploit curvature, submodularity ratio, …
  - Other optimization problems: delete k edges, …
  - Other robustness measures, other algorithmic approaches

Acknowledgements

Co-authors:

- Maria Predari, HU Berlin
- Rob Kooij, TU Delft

Funding:

DFG grant ME 3619/4-1: Accelerating Matrix Computations for Mining Large Dynamic Complex Networks

Thank you for your attention!

NetworKit
Large-scale Network Analysis