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

ACDA Seminar – Aussois, France – September 2022

Accepted by ASONAM '22

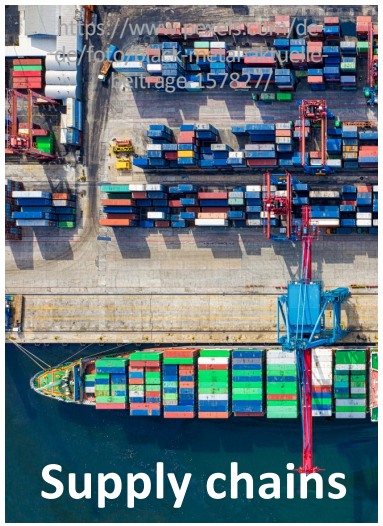
Robustness in Networks

Introduction and Motivation

- Networks appear in many applications
- Robustness a major issue –
  are there (many) alternative routes?

Computer networks

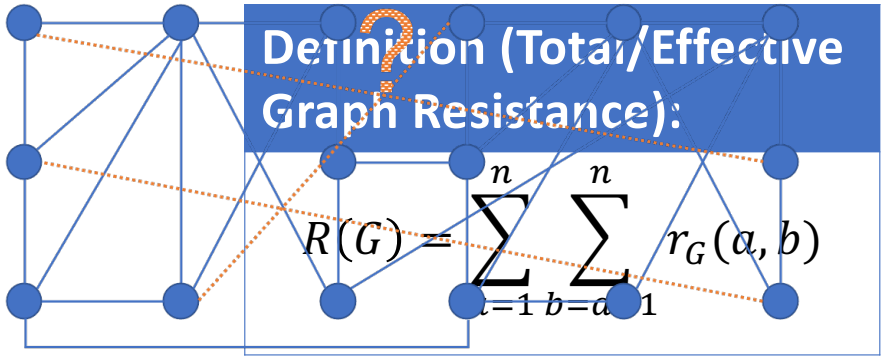
[https://sites.google.com/site/technologynaitayac/network-system]



k-GRIP with Total Graph Resistance

Definition:

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



Definition (Total/Effective Graph Resistance):

$$R(G) = \sum_{a=1}^{n} \sum_{b=a+1}^{n} r_G(a, b)$$

Notions of Robustness:

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

Definition (Effective Resistance):

$$r_G(a, b) = l^+_{a,a} - 2l^+_{a,b} + l^+_{b,b}$$

State of the Art (SotA)

- 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 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?
- How to select good candidates quickly?

Greedy Framework

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 step
    s ← CandidateSize(m, n, k, ϵ)
    for r ← 0, ..., k − 1 do                      ▷ main loop
        S ← Candidates(s, G_r, ...)
        for each {a, b} ∈ S × S do                ▷ # of evaluations
            gain(a, b) ← Eval(a, b, ...)          ▷ single evaluation
        (a*, b*) ← argmax_{a∈S×b∈S} gain(a, b)
        G_{r+1} = G_r ∪ (a*, b*)
        Update(G_{r+1}, ...)                      ▷ update step
    return G_{r+1}
```

Contribution

Acceleration of greedy optimization

- 3(-4) ways of candidate selection, according to:
  - 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× faster
- Much larger graphs can be processed than with SotA

Algorithmic Approach 0

Simple stochastic greedy (SIMPLSTOCH)

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

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

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 step
    s ← CANDIDATESIZE(m, n, k, ϵ)
    for r ← 0, ..., k − 1 do                              ▷ main loop
        S ← CANDIDATES(s, G_r, ...)
        for each {a, b} ∈ S × S do                        ▷ # of evaluations
            gain(a, b) ← EVAL(a, b, ...)                  ▷ single evaluation
        (a*, b*) ← argmax_{a∈S×b∈S} gain(a, b)
        G_{r+1} = G_r ∪ (a*, b*)
        UPDATE(G_{r+1}, ...)                              ▷ update step
    return G_{r+1}
```

Algorithmic Approach 1

ColStoch

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

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 step
    s ← CandidateSize(m, n, k, ε)
    for r ← 0, ..., k − 1 do                              ▷ main loop
        S ← Candidates(s, G_r, ...)
        for each {a, b} ∈ S × S do                        ▷ # of evaluations
            gain(a, b) ← Eval(a, b, ...)                  ▷ single evaluation
        (a*, b*) ← argmax_{a∈S×b∈S} gain(a, b)
        G_{r+1} = G_r ∪ (a*, b*)
        Update(G_{r+1}, ...)                              ▷ update step
    return G_{r+1}
```

Algorithmic Approach 2

*StochJLT: SimplStochJLT and ColStochJLT

- Gain computation needs two distances:

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

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

- JLT: random projection for dimensionality reduction
- $O(\log s)$ linear systems
- Works with
  - Stochastic greedy (SimplStochJLT) and
  - Column-based stochastic greedy (ColStochJLT)

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 step
    s ← CandidateSize(m, n, k, ε)
    for r ← 0, ..., k − 1 do                      ▷ main loop
        S ← Candidates(s, G_r, ...)
        for each {a, b} ∈ S × S do                ▷ # of evaluations
            gain(a, b) ← Eval(a, b, ...)          ▷ single evaluation
        (a*, b*) ← argmax_{a∈S×b∈S} gain(a, b)
        G_{r+1} = G_r ∪ (a*, b*)
        Update(G_{r+1}, ...)                      ▷ update step
    return G_{r+1}
```

$$\mathrm{gain}(a,b) \approx \frac{\left\|\mathbf{PL}^\dagger(\mathbf{e}_a - \mathbf{e}_b))^2\right\|}{1 + \left\|\mathbf{QB}^T\mathbf{YP}(\mathbf{e}_a - \mathbf{e}_b)^2\right\|}$$

Algorithmic Approach 3

SpecStoch

- Express gain function by spectral decomposition:

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

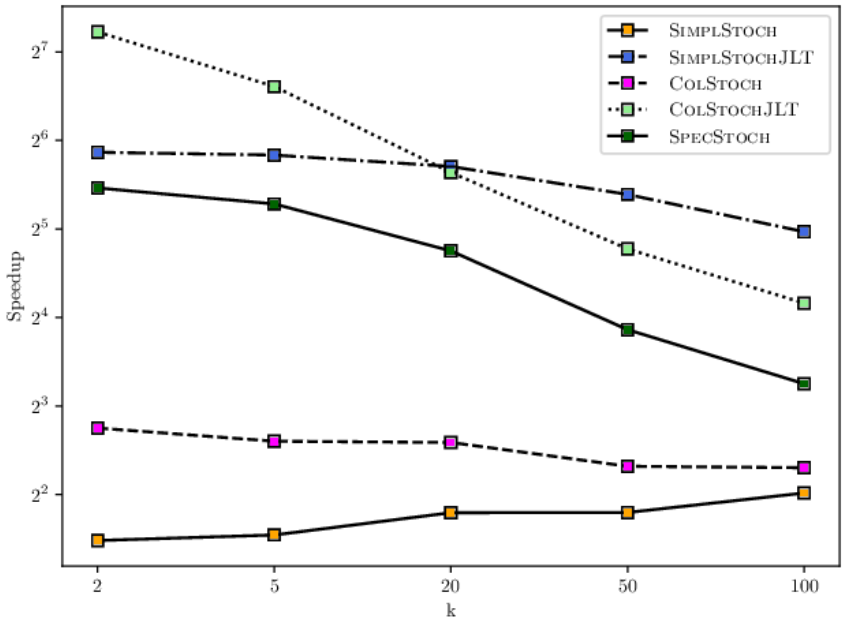
- Approximation with small eigenvalues and corresponding eigenvectors, plus largest eigenvalue
- c eigenpairs, computed with Lanczos
- New upper and lower bounds for gain
- Bootstrapping for eigenpair updates

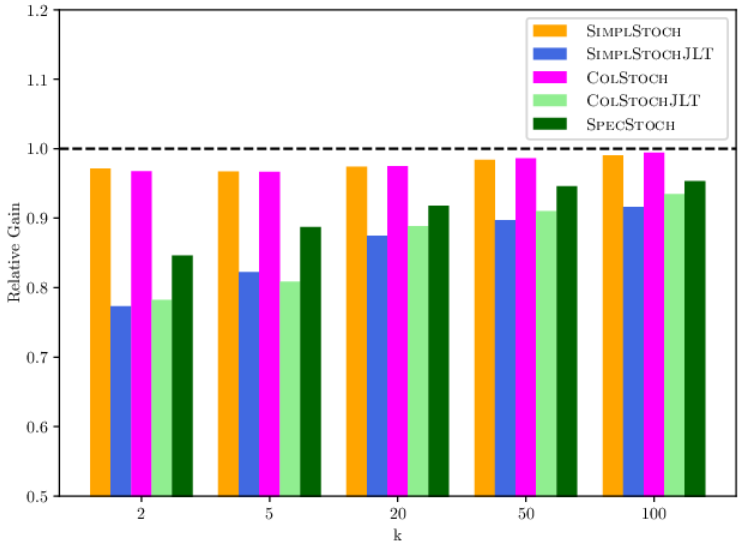
Algorithm 1 General framework for k-GRIP

```
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 step
    s ← CandidateSize(m, n, k, ϵ)
    for r ← 0, ..., k − 1 do                      ▷ main loop
        S ← Candidates(s, G_r, ...)
        for each {a, b} ∈ S × S do                ▷ # of evaluations
            gain(a, b) ← Eval(a, b, ...)          ▷ single evaluation
        (a*, b*) ← argmax_{a∈S×b∈S} gain(a, b)
        G_{r+1} = G_r ∪ (a*, b*)
        Update(G_{r+1}, ...)                      ▷ update step
    return G_{r+1}
```

Experiments

- C++ implementation with NetworKit
- Real-world instances:



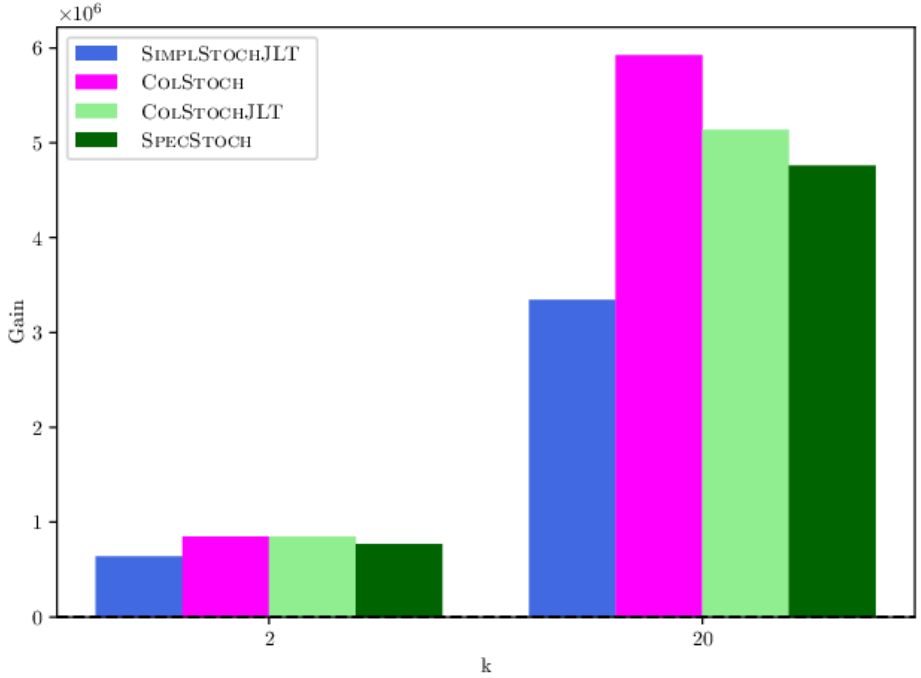


Medium-sized instances:

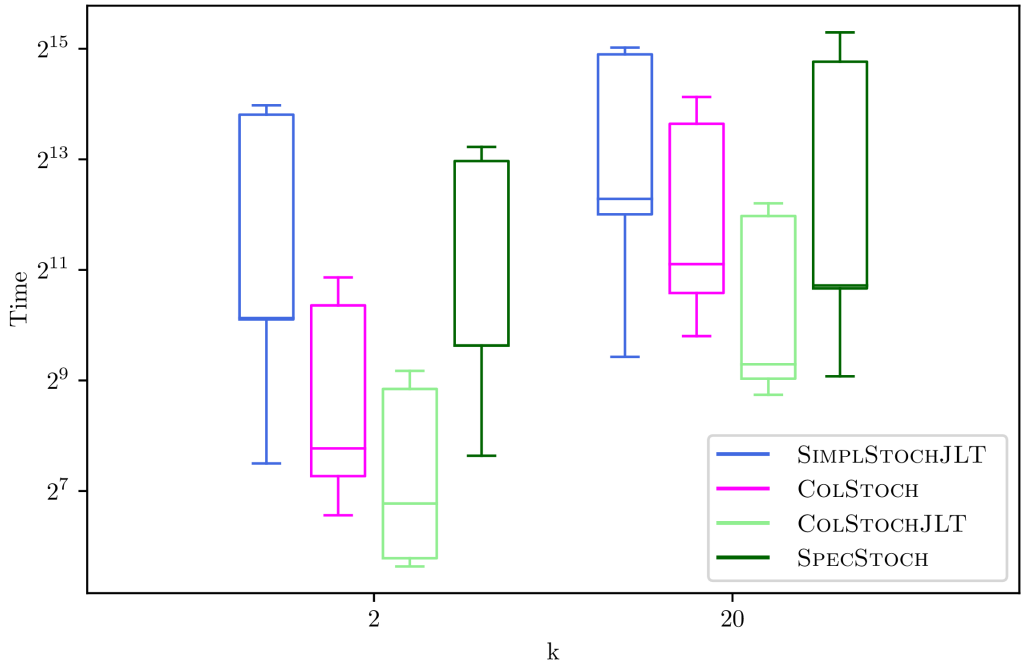
- ColStoch close to Greedy in quality, 5-6x faster
- *JLT and SpecStoch much faster, but with some quality loss
- Best tradeoff: SpecStoch

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
- Stochastic greedy algorithm: faster than SotA, similar quality
- Faster selection of candidates: three strategies (others possible)
- Different tradeoffs: ColStoch already 5-6x 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

Other Recent Works from Our Group

- Eugenio Angriman, Michal Boron, Henning Meyerhenke: A Batch-dynamic Suitor Algorithm for Approximating Maximum Weighted Matching. ACM J. Exp. Algorithmics 27: 1.6:1-1.6:41 (2022)
- Alexander van der Grinten, Elisabetta Bergamini, Oded Green, David A. Bader, Henning Meyerhenke: Scalable Katz Ranking Computation in Large Static and Dynamic Graphs. ACM J. Exp. Algorithmics 27: 1.7:1-1.7:16 (2022)
- Eugenio Angriman, Fabian Brandt-Tumescheit, Leon Franke, Alexander van der Grinten, Henning Meyerhenke: Interactive Visualization of Protein RINs using NetworKit in the Cloud. IPDPS Workshops 2022: 255-264
- Alexander van der Grinten, Geert Custers, Duy Le Thanh, Henning Meyerhenke: Fast Dynamic Updates and Dynamic SpGEMM on MPI-Distributed Graphs. CoRR abs/2202.08808 (2022) – accepted by CLUSTER‘22.
- Alexander van der Grinten, Maria Predari, Florian Willich: A Fast Data Structure for Dynamic Graphs Based on Hash-Indexed Adjacency Blocks. SEA 2022: 11:1-11:18