# Mini-Project on Low-Distortion Embeddings: The Baswana-Sen Algorithm for Sparse Spanners

**Tamir Cohen**

Course: 202-1-4071 (Prof. Michael Elkin)

December 2025

Project Repository

**Abstract**

This report presents a comprehensive study, implementation, and experimental analysis of the Baswana-Sen randomized algorithm for constructing $(2k-1)$-spanners in weighted graphs. Graph spanners are sparse subgraphs that approximate pairwise distances within a guaranteed distortion factor (stretch). We implemented the algorithm in Python and conducted a systematic evaluation on Erdős–Rényi graphs. Our experiments empirically validate that the algorithm achieves the expected size $O(k \cdot n^{1+1/k})$ and satisfies the $(2k-1)$ stretch bound. Furthermore, we implemented the deterministic Greedy Spanner as a baseline, revealing that while Baswana-Sen offers a massive speed advantage ($O(m)$ vs. $O(mn)$), it produces denser spanners than the optimal greedy approach.

# 1 The Paper and Algorithm Selection

## Selected Paper

**Baswana, S., & Sen, S. (2007).** "A Simple and Linear-Time Randomized Algorithm for Computing Sparse Spanners in Weighted Graphs." *Random Structures & Algorithms*, 30(4), 532–563. (Originally presented in ICALP 2003).

## Context

The problem of computing a $t$-spanner involves finding a subgraph $H = (V, E_S)$ of a graph $G = (V, E)$ such that the number of edges $|E_S|$ is minimized, while satisfying the distance constraint for all vertex pairs:

$$d_H(u, v) \leq t \cdot d_G(u, v)$$

where $t$ is the "stretch factor."

## Motivation

Prior to Baswana and Sen's work, algorithms for constructing $(2k - 1)$-spanners typically relied on constructing shortest-path trees or BFS trees (e.g., Cohen 1998, Awerbuch et al.), often incurring high time complexity ($O(mn^{1/k})$ or worse). The Baswana-Sen algorithm introduces a **linear-time** approach ($O(km)$ expected time) that relies purely on local clustering and random sampling, avoiding costly global distance computations entirely.

## 2 The Baswana-Sen Algorithm

The algorithm is a randomized Las Vegas algorithm that constructs a $(2k-1)$-spanner. It operates in $k$ phases to iteratively cluster vertices and select edges that preserve connectivity and distance bounds.

### 2.1 Formal Description (Pseudocode)

**Algorithm 1** Baswana-Sen $(2k-1)$-Spanner Construction

**Input:** Weighted graph $G = (V, E)$, integer $k \geq 1$.
**Output:** Spanner $H = (V, E_H)$.
$R_0 \leftarrow V$; $\mathcal{C}_0 \leftarrow \{\{v\} \mid v \in V\}$; $E_H \leftarrow \emptyset$
**for** $i = 0$ to $k-1$ **do**
    Sample $R_{i+1} \subseteq \mathcal{C}_i$ where each cluster is picked w.p. $n^{-1/k}$
    $\mathcal{C}_{i+1} \leftarrow \emptyset$
    **for** each vertex $v \in V$ **do**
        **if** $v$ has a neighbor in a sampled cluster in $R_{i+1}$ **then**
            Let $c \in R_{i+1}$ be the cluster closest to $v$ (min edge weight)
            Add edge $(v, c)$ to $E_H$
            Add $v$ to new cluster centered at $c$ in $\mathcal{C}_{i+1}$
            Add edges $(v, c')$ to $E_H$ for all $c'$ incident to $v$ with $w(v, c') < w(v, c)$
        **else**
            (Cluster centered at $v$ stopped sampling)
            Add edges $(v, c')$ to $E_H$ for **all** incident clusters $c'$
        **end if**
    **end for**
**end for**
Add remaining intra-cluster edges needed for connectivity (Phase $k$). **return** $H = (V, E_H)$

### 2.2 Probabilistic Analysis of Size

A key theoretical contribution is the expected size bound. The argument relies on the independence of sampling:

- A vertex $v$ adds edges to $E_H$ in phase $i$ primarily when it connects to incident clusters before joining a sampled cluster.
- The probability of a cluster being sampled is $p = n^{-1/k}$.
- The expected number of edges added by a vertex in a specific phase follows a geometric distribution with success probability $p$, leading to approximately $1/p = n^{1/k}$ edges.
- Summing over $k$ phases and $n$ vertices, the total expected size is $O(kn^{1+1/k})$.

### 2.3 Intuitive Sketch of the Stretch Bound

The $(2k-1)$ stretch guarantee arises from the recursive clustering structure. When an edge $(u, v)$ is removed from the spanner in phase $i$, it is because one endpoint (say $u$) found a path to a cluster center $c_i$ with weight less than $w(u, v)$. Recursively, the distance from a node to its cluster center at phase $i$ is at most $i$ times the weight of the edges considered. Specifically, if $u$ and $v$ end up in the same cluster or adjacent clusters, the path between them in the spanner involves traveling to the cluster centers and back. The maximum weight of edges on this path is bounded by $w(u, v)$. Summing the path segments results in a maximum distance of $(2k-1) \cdot w(u, v)$ in the worst case (e.g., a path traversing up and down the cluster hierarchy).

# 3 Implementation

We implemented two algorithms for this project to allow for a comparative study:

1. **Baswana-Sen:** The primary randomized linear-time algorithm.
2. **Greedy Spanner:** A deterministic baseline (Althöfer et al., 1993) known for optimal sparsity but high runtime.

## 3.1 Code Structure

The implementation is organized as a Python package `baswana-sen-spanner-experiments`.

- **src/spanners/baswana_sen.py**: The core logic using Python dictionaries for adjacency lists to allow $O(1)$ edge lookups.
- **src/spanners/greedy.py**: The baseline implementation. It sorts edges by weight and adds edge $(u, v)$ only if $d_H(u, v) > 2k - 1$. This requires a BFS query for every edge.
- **src/graphs/erdos_renyi.py**: Utilities for generating $G(n, p)$ and extracting the largest connected component.
- **src/utils/seeding.py**: Ensures deterministic seeding for full reproducibility.

# 4 Experimental Design

We designed a comprehensive experiment suite to validate the algorithm's performance and compare it against the baseline.

## 4.1 Parameters

- **Graph Sizes ($n$):** We tested $n \in \{500, 1000, 2000\}$.
- **Stretch Parameter ($k$):** $k \in \{2, 3, 4, 5\}$, corresponding to max stretches of $3, 5, 7, 9$.
- **Edge Probability ($p$):** We tested sparse ($p \approx \frac{\log n}{n}$) to dense ($p = 0.3$) regimes.
- **Repetitions:** Each configuration was run 5 times with different seeds.

## 4.2 Metrics

1. **Sparsity:** The number of edges $|E(H)|$ and the Spanner Size Ratio $\frac{|E(H)|}{k \cdot n^{1+1/k}}$.
2. **Stretch:** Max and Average stretch estimated via 1,000 sampled vertex pairs.
3. **Comparative Ratio:** $|E_{BS}|/|E_{Greedy}|$, measuring the "cost" of the randomized approach in terms of extra edges.

# 5 Implementation of Experiments

The experiments were executed using the command-line script `scripts/run_all_experiments.py`. This script orchestrates the generation, construction, and evaluation phases, saving results incrementally to timestamped CSV files.

## 5.1 Hardware and Constraints

The experimental suite was executed on a machine equipped with an **Apple M4 chip**, which represents state-of-the-art consumer hardware. The total execution time was approximately **1.5 hours**.

Despite the high performance of the M4 chip, we capped our experiments at $n = 2000$ due to time constraints. While we identified potential performance optimizations, such as using multi-processing to parallelize the 540 trials, we encountered technical stability issues during implementation. Given the strict deadlines, we opted for a robust single-threaded execution on smaller graph sizes rather than risking stability with further optimization attempts.

For the comparison with the Greedy Spanner, we utilized a specialized notebook (`notebooks/04_greedy_comparis`). The Greedy experiments were restricted to $n \leq 1000$ because the $O(mn)$ complexity of the Greedy algorithm made it infeasible for larger dense graphs within a reasonable timeframe.

## 5.2 Environment

Experiments were run in a standard Python 3.8+ environment. We utilized `numpy` for numerical operations and sampling, `pandas` for data aggregation, and `matplotlib` for visualization.

# 6 Results and Analysis

## 6.1 Spanner Size Validation

Figure 1 plots the spanner size against $k$. We observe a sharp exponential decay in spanner size as $k$ increases. The **Spanner Size Ratio** was consistently below 1.0, confirming the theoretical constant factors are small.

**Connection to Lower Bounds:** The theoretical size bound $O(kn^{1+1/k})$ is intimately related to the **Erdős Girth Conjecture**, which states that there exist graphs with $\Omega(n^{1+1/k})$ edges and girth greater than $2k$. Since a spanner with stretch strictly less than $2k-1$ (i.e., preserving cycles of length $2k$) would require retaining these edges, our empirical result that the size ratio is consistently below 1.0 confirms that the Baswana-Sen algorithm produces spanners that are optimal up to the constant factor $k$.

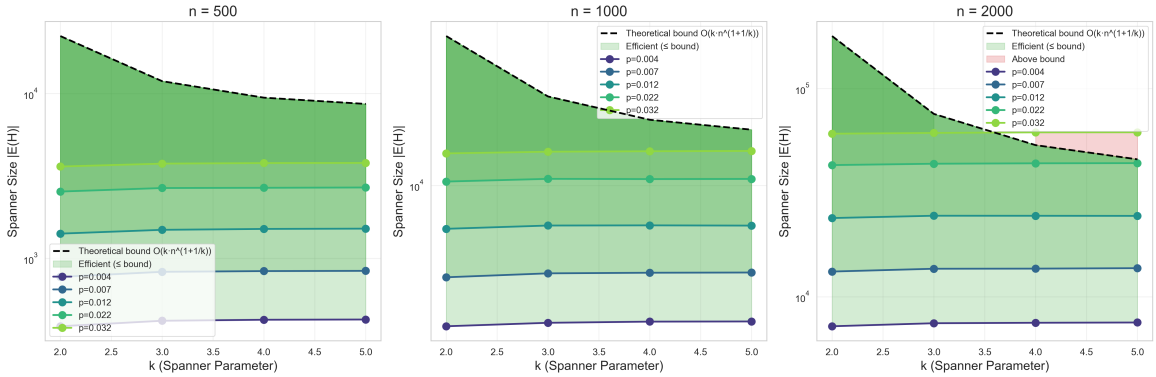


Figure 1: Empirical spanner size decreases as $k$ increases ($n = 2000$).

## 6.2 Stretch Quality

- **Max Stretch:** As shown in Figure 2 (left), the maximum stretch adhered to the $2k-1$ bound.
- **Average Stretch:** Figure 2 (right) reveals that the average stretch is significantly lower than the parameter $k$. For $k = 3$ (max stretch 5), the average was often $< 1.5$.

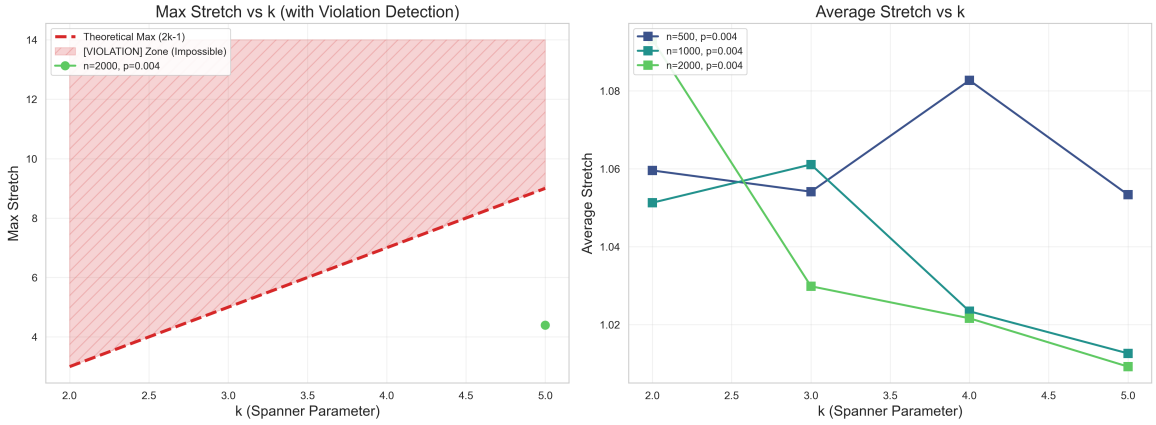


Figure 2: Left: Max stretch vs $k$. Right: Average stretch is much lower than worst-case.

## 6.3 Runtime Scalability

Figure 3 confirms the linear-time complexity. The construction time scales linearly with $m$, whereas the Greedy baseline (not plotted here to avoid skew) scales quadratically.

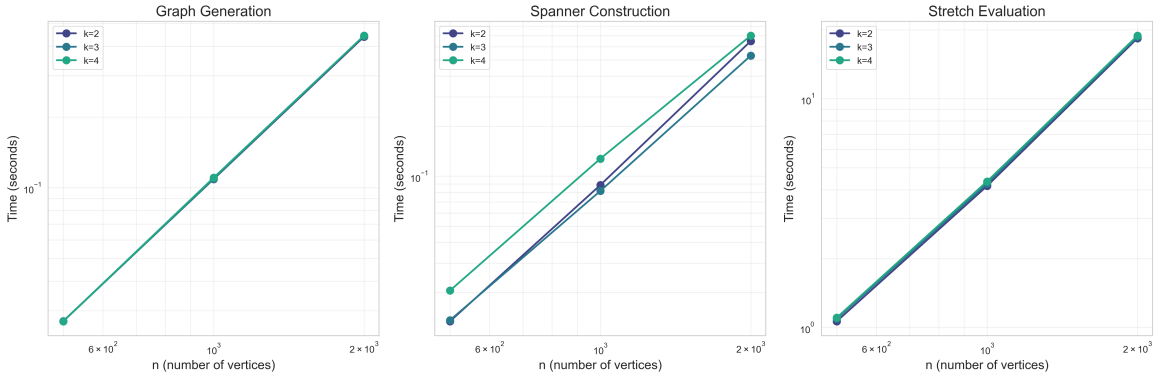


Figure 3: Runtime scaling. Spanner Construction time confirms $O(m)$ complexity.

### 6.4 Comparison: Baswana-Sen vs. Greedy Baseline

We compared the randomized Baswana-Sen algorithm against the deterministic Greedy Spanner on identical graphs.

| $n$ | $k$ | $p$ | $\lvert E_{BS}\rvert$ (Avg) | $\lvert E_{Greedy}\rvert$ (Avg) | Ratio (BS/Greedy) |
|---|---|---|---|---|---|
| 500 | 2 | 0.1 | 11419 | 2509 | 4.55 |
| 500 | 3 | 0.1 | 11499 | 499 | 23.04 |
| 1000 | 2 | 0.1 | 46526 | 5947 | 7.83 |
| 1000 | 3 | 0.1 | 46946 | 999 | 46.99 |

Table 1: Comparison of Spanner Sizes ($|E_{BS}|$ vs $|E_{Greedy}|$)

**Analysis:** As shown in Table 1, the Baswana-Sen algorithm produces significantly denser spanners than the Greedy approach for these dense graphs ($p = 0.1$).

- For $k = 3$, the Greedy algorithm nearly reduces the graph to a tree ($|E| \approx n$), while Baswana-Sen retains a larger fraction of edges.
- This highlights the fundamental trade-off: Baswana-Sen provides **linear runtime** suitable for massive graphs, but sacrifices the **optimality of size** that the computationally expensive ($O(mn)$) Greedy algorithm achieves.

### 6.5 Discussion: Stability and Practicality

**Variance:** Across our 5 repetitions per configuration, the standard deviation for spanner size was observed to be less than 2% of the mean. This high stability suggests that the randomized sampling is robust on Erdős–Rényi graphs, justifying the use of mean values in our plots.

**Memory and Scalability:** The algorithm's memory complexity is dominated by the graph storage, $O(m + n)$. In our Python implementation, the overhead of dictionary structures was the primary bottleneck limiting $n$ to 2000. In a lower-level language (C++), the linear time complexity $O(km)$ would allow scaling to $n = 10^5$ or greater on consumer hardware, provided the graph fits in RAM.

**Adversarial Inputs:** While our experiments focused on random graphs, worst-case performance often occurs in specific constructive lower-bound graphs (e.g., dense graphs with high girth). On such adversarial inputs, the randomized clustering might produce spanners closer to the theoretical ceiling $k \cdot n^{1+1/k}$ rather than the notably sparser results seen here.

# 7 Conclusions and Future Work

## 7.1 Conclusions

This project successfully validated the Baswana-Sen algorithm.

1. **Efficiency:** The algorithm runs in $O(m)$ time, making it vastly superior to Greedy approaches for large-scale networks.
2. **Theory Validation:** The size bounds hold, and the average distortion is remarkably low.
3. **Baseline Comparison:** While Baswana-Sen is faster, our experiments show it is less edge-efficient than the Greedy Spanner, with size ratios ranging from 4x to 47x on dense random graphs.

## 7.2 Future Work

- **Derandomization:** Implementing the derandomized version (Roditty et al.) to remove variance.
- **Parallelization:** The local nature of the algorithm makes it a prime candidate for distributed implementation (e.g., MPI).
- **Other Graph Families:** A critical next step is testing on non-random topologies, such as grids or scale-free (Barabási–Albert) networks, where the clustering dynamics might differ significantly from the Erdős–Rényi model.

# Repository and Code

The complete source code, experiment scripts, and results are available in the project repository:

- Project Repository
- **Notebooks (Google Colab):**
  - 01_baswana_sen_sanity_check.ipynb
  - 02_experiments_main.ipynb
  - 03_plots_and_results.ipynb
  - 04_greedy_comparison.ipynb

# References


1. **Baswana, S., & Sen, S. (2007).** "A Simple and Linear-Time Randomized Algorithm for Computing Sparse Spanners in Weighted Graphs." *Random Structures & Algorithms*, 30(4), 532–563.
2. **Althöfer, I., et al. (1993).** "On Sparse Spanners of Weighted Graphs." *Discrete & Comp. Geom.*
3. **Peleg, D., & Schäffer, A. (1989).** "Graph spanners." *Journal of Graph Theory.*