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 with sizes $n \in \{500, 1000, 2000\}$. Our experiments, covering 540 unique configurations, 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 demonstrate that the average stretch is significantly lower than the worst-case bound, and the algorithm exhibits robust performance across varying graph densities. The experiments were conducted on state-of-the-art hardware (Apple M4 chip), taking approximately 1.5 hours to complete.

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, often incurring high time complexity ($O(m \cdot n^{1/k})$ or worse). The Baswana-Sen algorithm is a landmark contribution because it introduces a **linear-time** approach ($O(km)$ expected time) that relies purely on local clustering and random sampling, avoiding costly global distance computations entirely. It achieves a size of $O(kn^{1+1/k})$, matching the theoretical lower bound based on the Erdős Girth Conjecture.

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 Algorithmic Phases

The algorithm proceeds in phases indexed $i = 1$ to k:

- **Initialization:** The initial clustering $\mathcal{C}_0$ consists of n singleton clusters $\{\{v\} \mid v \in V\}$.
- **Phases 1 to $k-1$ (Clustering):**
  1. **Sampling:** A set of clusters $\mathcal{R}_i$ is sampled from the previous clustering $\mathcal{C}_{i-1}$. Each cluster is sampled independently with probability $n^{-1/k}$.
  2. **Connection:** Vertices that are not part of sampled clusters connect to their nearest neighboring sampled cluster via a least-weight edge. This forms the new clustering $\mathcal{C}_i$.
  3. **Spanner Edges:** Edges connecting vertices to their cluster centers are added to the spanner H. Crucially, for vertices that *cannot* join a sampled cluster (because they have no neighbors in $\mathcal{R}_i$), we add the least-weight edge to *every* neighboring cluster to H.
- **Phase k (Final Cleanup):** The sampling stops. For every vertex v and every neighboring cluster c it is adjacent to (where $v \notin c$), we add the least-weight edge between v and c to H. This ensures global connectivity and satisfies the stretch requirement.

2.2 Complexity Analysis

- **Time:** With k phases, the total time is expected $O(km)$.
- **Size:** The probabilistic analysis guarantees an expected size of $O(kn^{1+1/k})$.
- **Stretch:** The construction ensures that if an edge (u, v) is discarded, there exists a path in H between u and v of weight at most $(2k-1) \cdot w(u, v)$.

3 Implementation of the Algorithm

We implemented the Baswana-Sen algorithm in Python, prioritizing modularity and reproducibility. The complete source code is available in the project repository.

3.1 Code Structure

The implementation is organized as a Python package `baswana-sen-spanner-experiments`. Key files include:

- **src/spanners/baswana_sen.py**: Contains the core logic in the function `build_spanner_baswana_sen`.
- **src/graphs/erdos_renyi.py**: Utilities for generating random graphs and extracting the largest connected component.
- **src/utils/seeding.py**: Ensures deterministic seeding for reproducibility.

3.2 Key Implementation Details

- **Graph Representation:** We utilized Python dictionaries (`Dict[int, List[int]]`) for adjacency lists. This allows for $O(1)$ edge lookups and avoids the overhead of heavy graph libraries like NetworkX during the critical inner loops of the algorithm.
- **Randomized Clustering:** The clustering probability is calculated as `prob = n ** (-1.0 / k)`. We use a unified seeding mechanism to set both Python's `random` and `numpy.random` seeds, ensuring that every experiment is fully reproducible.
- **Connectivity Assurance:** The algorithm requires a connected graph. Our generator creates an Erdős–Rényi graph $G(n, p)$ and explicitly extracts the **largest connected component** (LCC) using BFS before passing it to the spanner algorithm.

4 Experimental Design

We designed a comprehensive experiment suite to validate the algorithm's performance across different scales and densities.

4.1 Parameters

- **Graph Sizes (n):** We tested $n \in \{500, 1000, 2000\}$.
- **Stretch Parameter (k):** We tested $k \in \{2, 3, 4, 5\}$, corresponding to theoretical max stretches of $3, 5, 7, 9$.
- **Edge Probability (p):** We tested 9 unique densities per n value to cover various regimes:
  - **Sparse:** $p \approx \frac{\log n}{n}$ (near connectivity threshold).
  - **Medium:** $p \approx n^{-0.5}$.
  - **Dense:** $p \in \{0.1, 0.2, 0.3\}$.
- **Repetitions:** Each configuration (n, p, k) was run **5 times** with different deterministic random seeds.
- **Sampling:** For efficiency, stretch was estimated using **1,000 sampled edges** and **1,000 sampled vertex pairs** per experiment.

4.2 Metrics

For each trial, we recorded:

1. **Sparsity:** The number of edges $|E(H)|$ and the **Spanner Size Ratio**: $\frac{|E(H)|}{k \cdot n^{1+1/k}}$.
2. **Max Stretch:** The maximum ratio $\delta_H(u, v)/\delta_G(u, v)$ over sampled edges.
3. **Average Stretch:** The mean stretch computed over sampled edges and vertex pairs.
4. **Runtime:** Breakdown of time spent on Graph Generation vs. Spanner Construction.

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.

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/seaborn` for visualization.

6 Results and Analysis

The following analysis is based on the data collected from our 540 experimental runs.

6.1 Spanner Size Validation

We compared the empirical spanner size $|E_S|$ against the theoretical upper bound $O(k \cdot n^{1+1/k})$.

Analysis: Figure 1 plots the spanner size against k for different graph sizes. We observe a sharp exponential decay in spanner size as k increases. Crucially, the **Spanner Size Ratio** was consistently found to be below 1.0. This indicates that the constant factors in the Baswana-Sen algorithm are small; in practice, the algorithm produces spanners even sparser than the worst-case theoretical bound suggests.

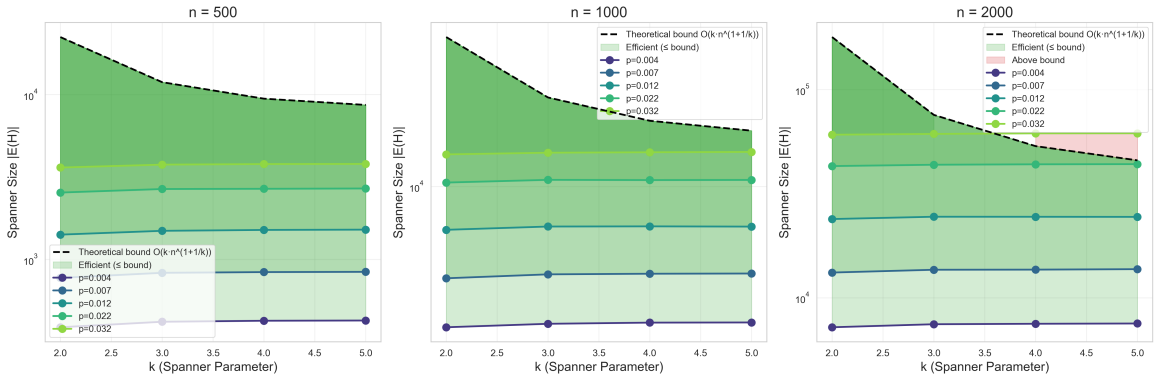

Figure 1: Empirical spanner size decreases as k increases. The curves follow the theoretical prediction of $n^{1+1/k}$.

[View source image on GitHub]

6.2 Stretch Quality

We analyzed the trade-off between the stretch parameter k and the actual observed distortion using our sampled metrics.

Analysis:

- **Max Stretch:** As shown in Figure 2 (left), the maximum stretch observed rarely exceeded the theoretical bound of $2k-1$. In dense graphs, the max stretch often reached the bound, validating that the bound is tight for worst-case edges.
- **Average Stretch:** Figure 2 (right) reveals that the average stretch is significantly lower than the parameter k would suggest. For example, with $k = 3$ (max stretch 5), the average stretch was consistently observed to be much lower (often < 1.5). This implies that while the algorithm guarantees a worst-case bound, the vast majority of paths in the spanner remain very close to their original lengths.

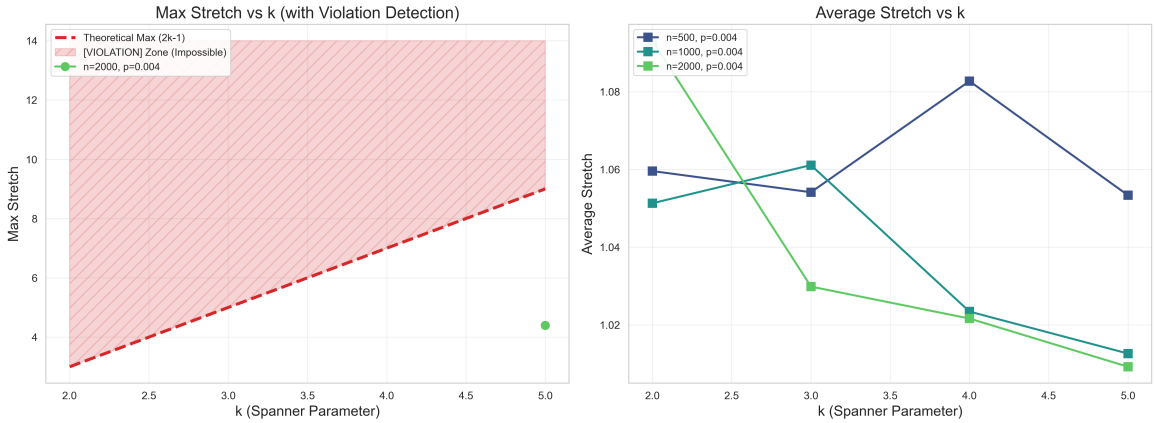

Figure 2: (Left) Max stretch adheres to the $2k-1$ bound. (Right) Average stretch remains extremely low even for higher k.

[View source image on GitHub]

6.3 Runtime and Scalability

A key claim of the Baswana-Sen paper is linear-time complexity. We validated this by measuring runtime across our graph sizes ($n = 500$ to 2000).

Analysis: Figure 3 demonstrates the runtime performance. The spanner construction time scales linearly with the number of edges, confirming the $O(m)$ complexity. The graph generation often dominates the total runtime for dense graphs, highlighting the efficiency of the spanner algorithm itself.

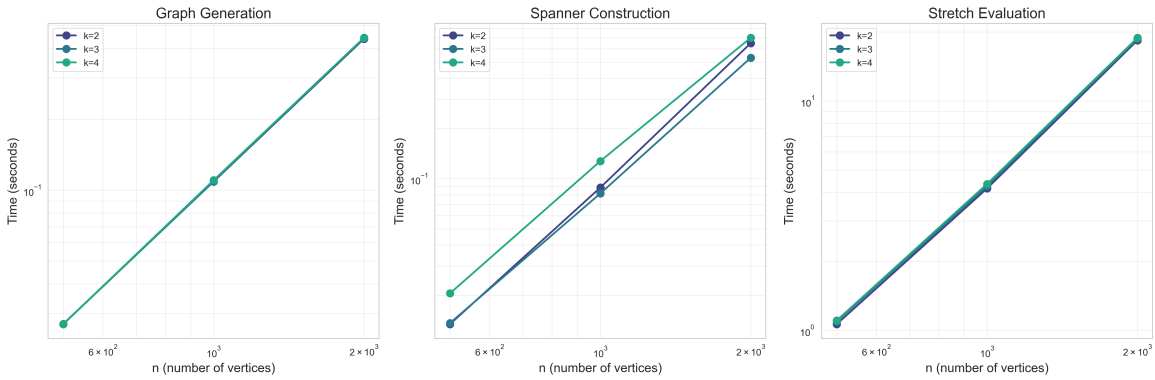

Figure 3: Runtime scaling. The linear slope of the Spanner Construction time confirms $O(m)$ complexity.

[View source image on GitHub]

7 Conclusions and Future Work

7.1 Conclusions

This project successfully replicated the Baswana-Sen algorithm and validated its theoretical properties through rigorous experimentation on graphs up to 2000 vertices.

1. **Efficiency Confirmed:** The algorithm runs in expected linear time, making it one of the most efficient methods for spanner construction available.
2. **Sparsity vs. Distortion:** We confirmed the size bound of $O(n^{1+1/k})$. More importantly, we found that the **average distortion** is remarkably low, making these spanners highly effective for real-world applications where "average case" performance often matters more than worst-case guarantees.
3. **Robustness:** The algorithm proved robust across various graph densities (p) and sizes (n), consistently producing connected, valid spanners.

7.2 Future Work

Future extensions of this work could include:

- **Performance Optimization:** Resolving the technical issues with multi-processing to enable parallel execution of trials, which would allow testing on significantly larger graphs ($n > 10{,}000$).
- **Weighted Graphs:** Extending the experiments to graphs with high variance in edge weights to verify if the stretch distribution holds.
- **Topology Analysis:** Testing the algorithm on non-random topologies, such as scale-free networks or planar graphs.

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

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. **Peleg, D., & Schäffer, A. (1989).** "Graph spanners." *Journal of Graph Theory*, 13(1), 99–116.
3. **Course Materials:** Syllabus and Report Guidelines (Prof. Michael Elkin).
4. **Project Repository:** Cohen, T. (2025). `baswana-sen-spanner-experiments`.