# GG-SSMs: Graph-Generating State Space Models

## Supplementary Material

## 1. Chazelle's MST Algorithm

In this section, we provide a concise overview of *Chazelle's MST algorithm* [1] and why it is instrumental to our proposed Graph-Generating State Space Models (GG-SSMs). Although multiple efficient minimum spanning tree (MST) algorithms exist (e.g., Kruskal's, Prim's, or Borůvka's), Chazelle's algorithm is particularly interesting due to its near-linear time complexity in the general graph setting.

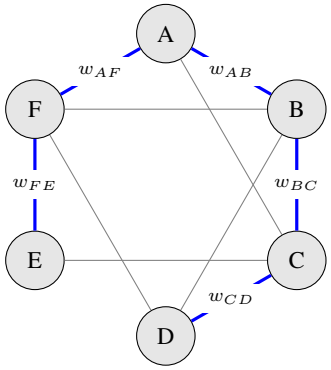


Figure 1. **Chazelle's MST Overview**. Soft heaps allow near-linear sorting of edges. MST edges (in blue) form a spanning structure with no cycles, connecting all vertices using the smallest weights $w$.

### 1.1. Core Idea and Time Complexity

Chazelle's MST algorithm belongs to the family of *soft heap* approaches. Its most prominent feature is achieving a runtime of $\mathcal{O}(E\,\alpha(E,V))$, where

- $V$ is the number of vertices in the graph,
- $E$ is the number of edges in the graph,
- $\alpha(\cdot,\cdot)$ is the *inverse Ackermann function*, a function that grows extremely slowly (much more slowly than $\log\log n$).

For any practical input size (e.g., up to millions of edges), $\alpha(E,V)$ remains a small constant (typically $\leq 4$). Therefore, the runtime is effectively *linear* for all real-world purposes.

**Algorithmic Outline.** At a high level, Chazelle's algorithm proceeds by maintaining a specialized priority queue known as a *soft heap* to handle edge weight comparisons and merges. It selectively *corrupts* (or perturbs) a small fraction of keys but guarantees a sufficiently accurate ordering to recover the MST. The soft heap structure allows various operations like insertions, extractions, and merges to be executed in approximately constant amortized time, modulo the very slowly growing $\alpha$ factor.

The algorithm can be outlined in three major steps:

1. **Sort or partition the edges** using the soft-heap structure such that they can be processed in a non-decreasing order of weights.
2. **Merge edge sets** while extracting edge candidates for the MST. These extractions remain almost linear because each edge is either integrated into the MST structure or discarded.
3. **Selective Corruption & Verification**: Since the soft heap may slightly perturb edge weights, a verification step ensures that these corrupted weights do not impact the MST correctness. With high probability, only a small fraction of edges require re-checking.

By the end, the edges forming the MST are collected using a Union-Find data structure (or a similar disjoint set data structure), combining efficiency with theoretical guarantees of correctness.

### 1.2. Practical Implications for GG-SSMs

In our GG-SSM framework, each layer requires constructing an MST on feature embeddings (e.g., pixel or token embeddings) to identify critical connections before performing state propagation. Since the number of edges $E$ can be large in dense graphs, employing Chazelle's MST algorithm with its near-linear time complexity is particularly appealing:

- **Scalability:** For a graph of $L$ vertices (e.g., $L$ embeddings), we can handle $\mathcal{O}(L^2)$ potential edges in worst-case dense settings or employ faster approximate methods in sparser representations. Either way, Chazelle's $\mathcal{O}(E\,\alpha(E,V))$ ensures minimal overhead when $E$ is proportional to $L$ or $L\log L$.
- **Uniqueness of MST Paths:** MST ensures exactly $L-1$ edges and a *unique path* between any two nodes. This property is crucial for our GG-SSMs, as it cleanly defines the path by which hidden states propagate. It further limits redundant computations and fosters efficient parameter sharing in state updates.

### 1.3. Intuition and Benefits

Intuitively, MST-based graph construction identifies the *closest* (or most similar) neighbors by selecting edges of the smallest weight (lowest dissimilarity). This yields a minimal set of edges connecting all nodes, providing a concise but effective skeleton to diffuse signals across the entire feature set. Consequently, even high-dimensional data with long-range dependencies can be efficiently processed with minimal redundancy. The slow-growing $\alpha(\cdot,\cdot)$ factor en-

sures that the overhead of constructing and maintaining this structure remains negligible for practical dataset sizes.

## References

[1] Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *J. ACM*, 47(6): 1028–1047, 2000. 1