Local Search for finding variable orderings of graphical models
About a quarter of a century, the notion of treewidth has now played a role in many investigations in algorithmic graph theory. While for a long time, the use of treewidth was limited to theoretical investigations, and it sometimes was believed that it could not play a role in practical applications, nowadays there is a growing tendency to use it in an actual applied setting.

(Bodlaender 2005)
Naturally modeled as a SLS problem because we can use w* as our evaluation criterion

SLS, repeated min-fill, and QuickBB are all *anytime*. It is possible to compare best ordering found so far as a function of time.
Basic Algorithm: Gradient Descent

- (1) Initialize to a random ordering or a min-fill ordering
- (2) Search adjacent orderings with lower induced width
- (3) When no such ordering exists, record it if it is the best so far, and go back to (1)
- Terminate after a fixed amount time (or iterations, or some other criterion)
Modifications

- Plateau Search: If there are no adjacent orderings with better $w^*$, "Search sideways" to one with the same $w^*$
  - Doesn't converge
Modifications

- **Plateau Search**: If there are no adjacent orderings with better \( w^* \), “Search sideways” to one with the same \( w^* \)
  - Doesn't converge

- **Tabu Search**: While searching sideways, cache all the orderings you visit. These are ineligible to be reached again. Clear the cache when you find an ordering with better \( w^* \)
  - Converges, but loses guarantees on space complexity
Alternative Modifications

- Simulated Annealing

- Initialize a countdown timer \( t \) to some amount of time when \( t = 0 \), terminate and return best ordering found so far whenever a better ordering is found update \( t \) :

\[
t = \text{elapsed\_time}
\]
Defining Neighborhood

- Two orderings are adjacent if you can reach one from the other by swapping a pair of vertices
Defining Neighborhood

- Two orderings are adjacent if you can reach one from the other by swapping a pair of vertices
  - At each iteration calculate $w^*$ of each adjacent ordering: $O(n)$ for each ordering
  - $| \text{Neighborhood} | = \binom{n}{2} = O(n^2)$
  - $O(n^3)$ work at each iteration
Defining Neighborhood

- This is too slow to do in a straight-forward manner. Need a more complex data structure that can quickly compute $w^*$ of each adjacent ordering.
Defining Neighborhood

- This is too slow to do in a straight-forward manner. Need a more complex data structure that can quickly compute $w^*$ of each adjacent ordering.
- Need not be ”swap”
Defining Neighborhood

- This is too slow to do in a straight-forward manner. Need a more complex data structure that can quickly compute $w^*$ of each adjacent ordering.
- Need not be "swap"
- (i) Go to the adjacent ordering that improves $w^*$ the most
- (ii) Go to any adjacent ordering that improves $w^*$
  - Same worst-case time as (i)
  - Heuristics for which pairs to consider first