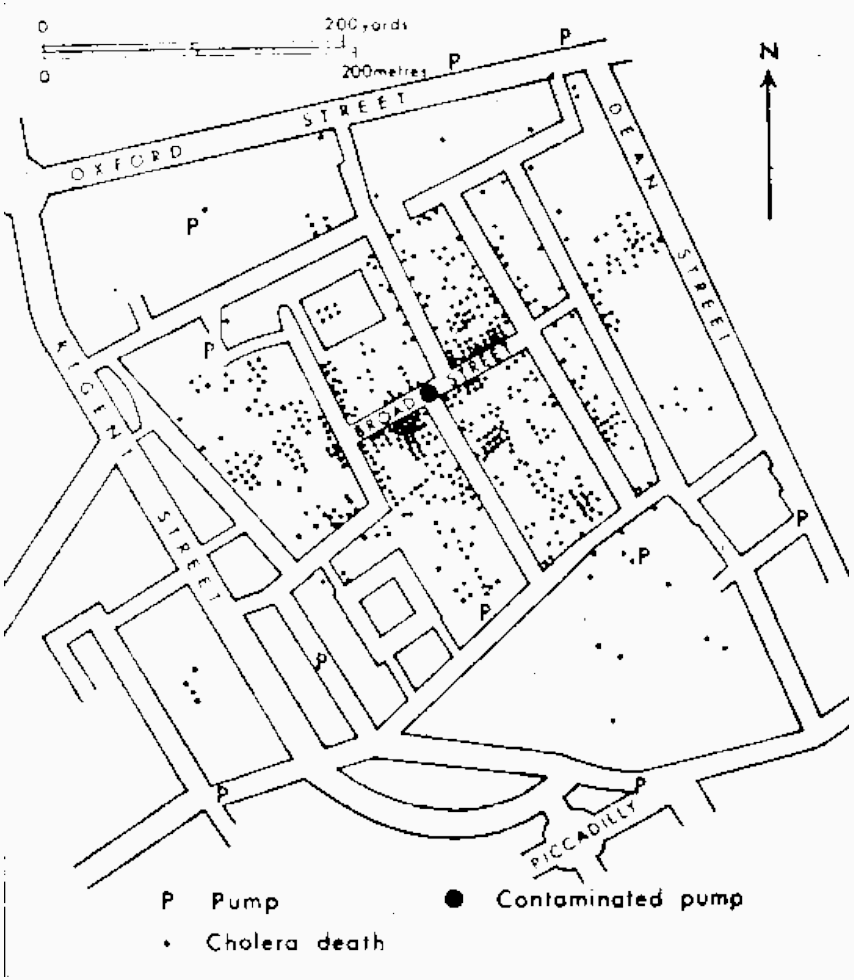
The similar connected partition problem

BCTCS 2025

Lewis Dyer

University of Glasgow

Some motivation



Aim to estimate disease risk over a geographic area - naturally represented as a graph
But many methods are computationally intensive - need to shrink down a graph to group similar points together, while respecting the graph structure.

Similar-Connected-Partition

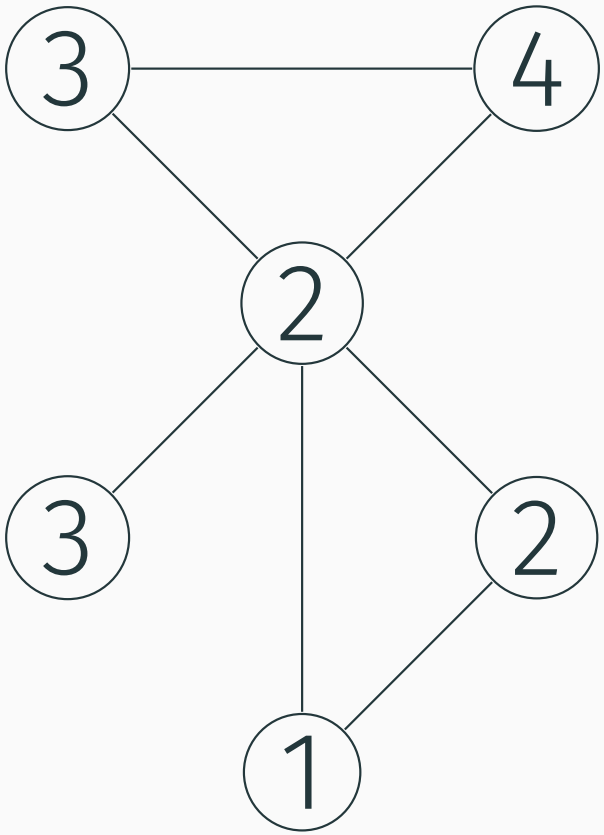
Input: A graph $G = (V, E)$, a number of parts $p \in \mathbb{N}$, a maximum part width $\delta \in \mathbb{Q}$, two part size bounds $B_1, B_2 \in \mathbb{N}$ and a vertex function $f : V \to \mathbb{Q}$.

Question: Is it possible to partition V into exactly p parts $R_1, \ldots, R_p$ such that, for each i:

1. R_i induces a connected subgraph of G
2. $B_1 \leq |R_i| \leq B_2$
3. For all pairs of vertices $x, y \in R_i$, $|f(x) - f(y)| \leq \delta$

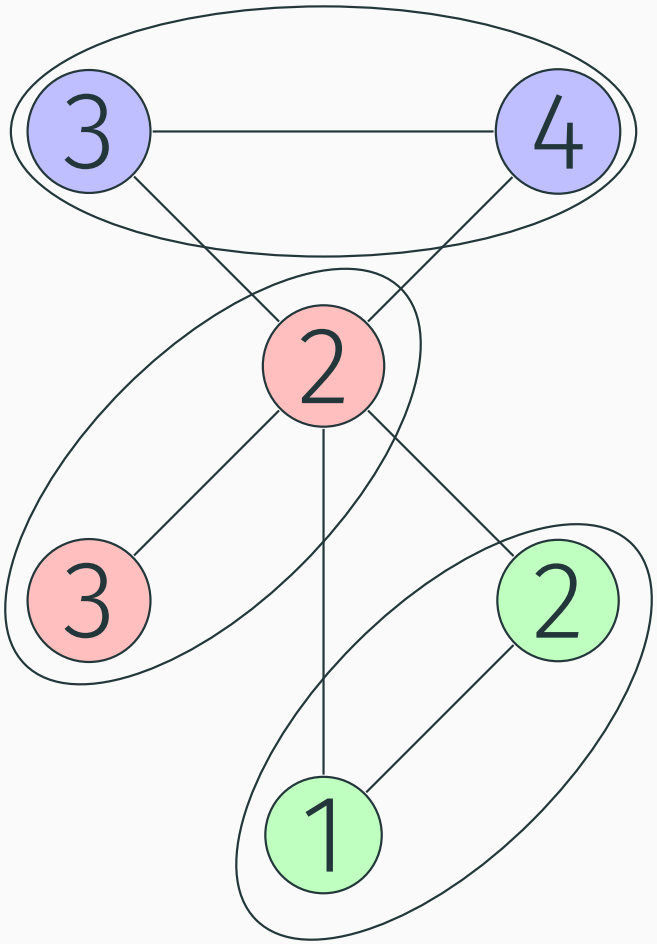
A quick example

Take $p = 3, \delta = 1, B_1 = 2$ and $B_2 = 3$.



A quick example

Take $p = 3, \delta = 1, B_1 = 2$ and $B_2 = 3$.



Some preliminaries

Similar-Connected-Partition is NP-hard, even for planar graphs or for any fixed p - reduction from Equitable-Connected-Partition which requires balanced part sizes.

When parts have exactly 2 vertices, solvable in polynomial time by reduction from finding perfect matchings.

We conjecture Similar-Connected-Partition is NP-hard when $B_1 = B_2 = k$ for any fixed $k \geq 3$.

Some more preliminaries

When $B_1 = 1$, can relax ”exactly p parts” to ”at most p parts” - we can ”peel off” vertices from bigger parts.

When G is a complete graph, solvable in polynomial time using a dynamic program (list vertex values in ascending order, and show a valid partition exists iff a valid partition with contiguous blocks in this order exists)

Maximum weight independent sets on trees

Given a graph G with vertex weights, find an independent set of G with maximum weight

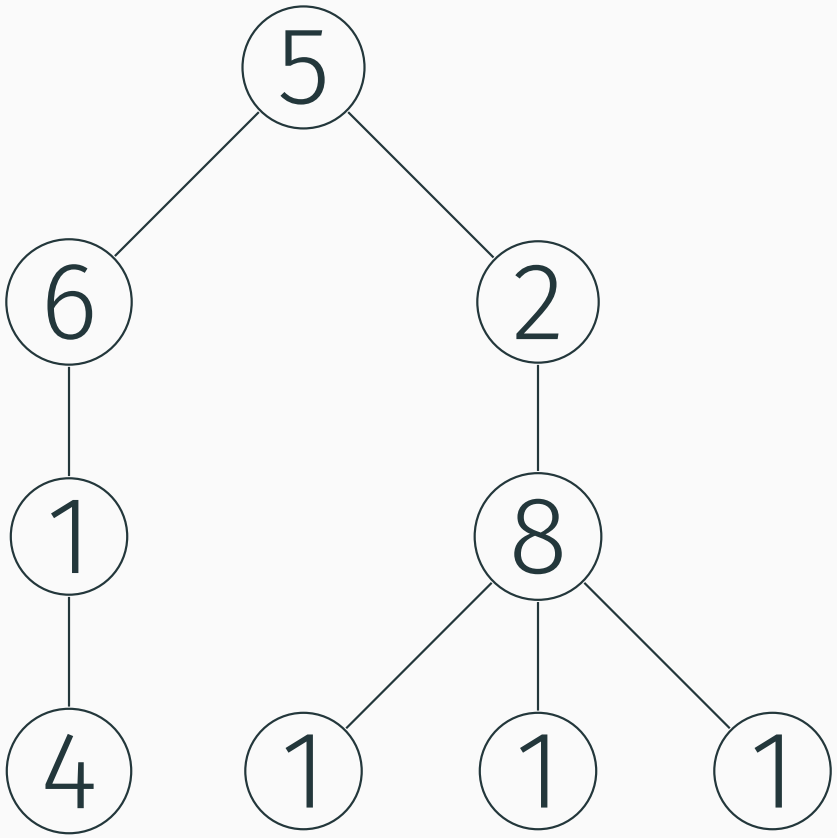
This is *NP*-hard in general, but polynomial time solvable on trees

Idea: for each node t in the tree, store the maximum weight independent set of the subtree rooted at t, which we denote as $D[t]$.

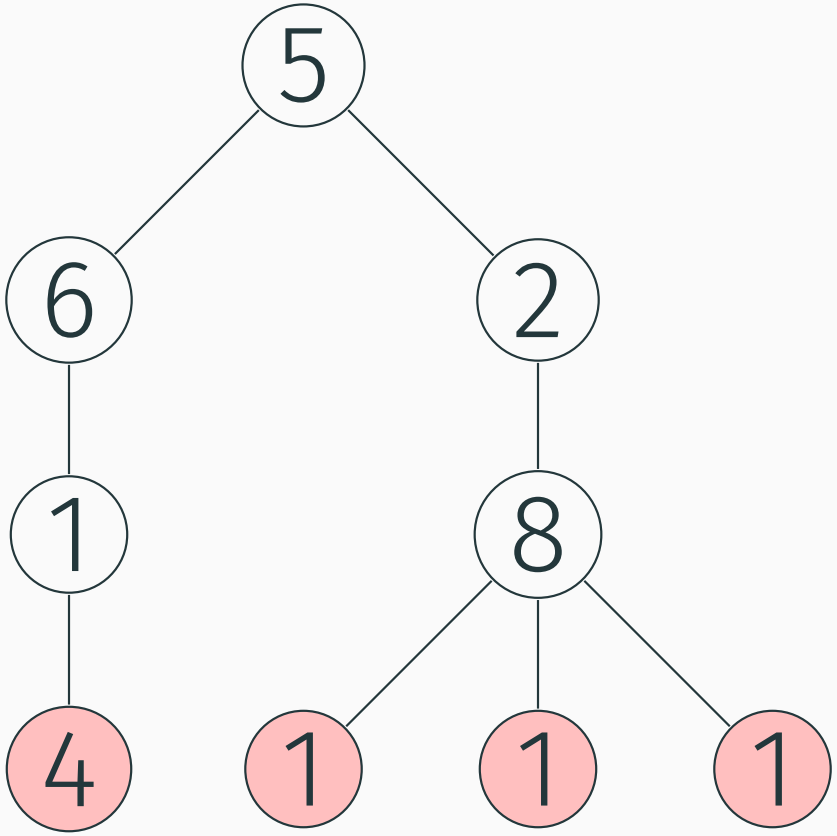
We have that

$$D[t] = \max \left(w(t) + \sum_{t' \text{ a grandchild of } t} D[t'], \sum_{t' \text{ a child of } t} D[t'] \right)$$

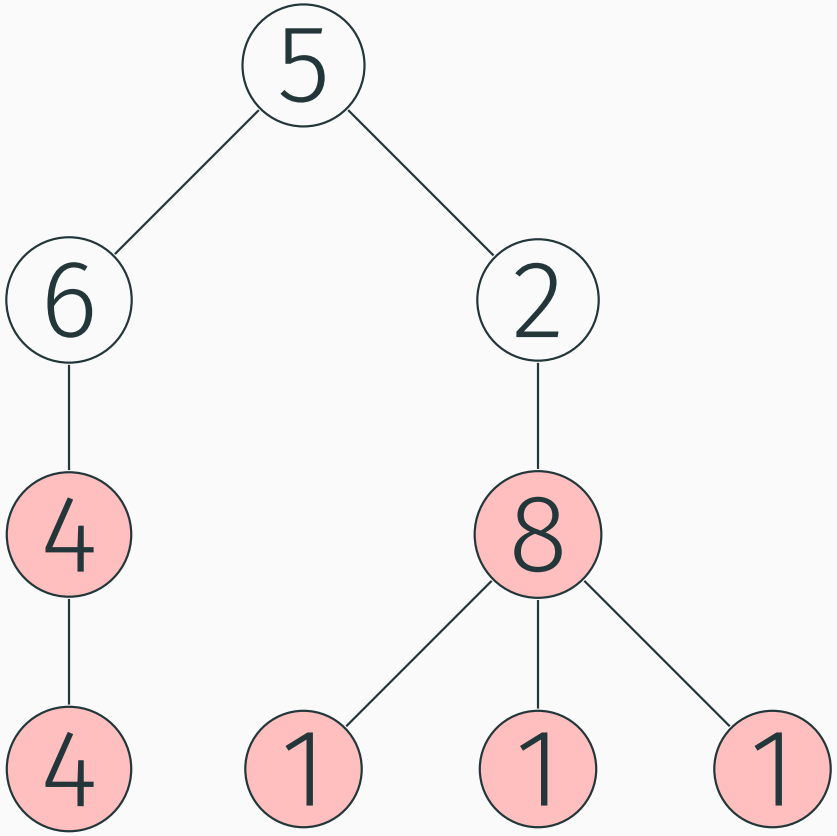
A different quick example



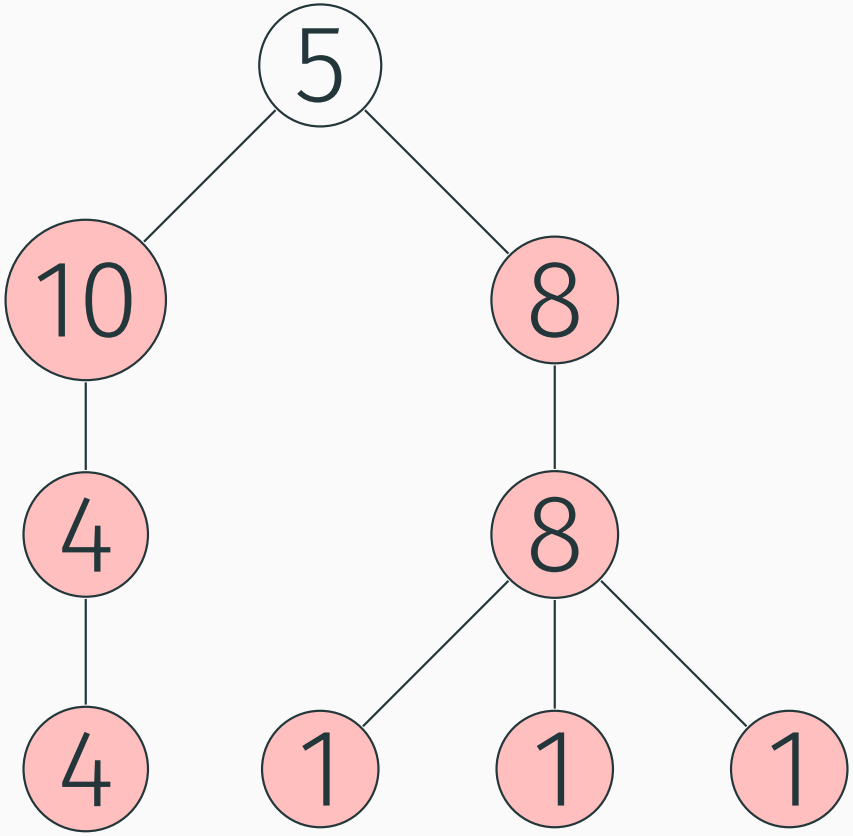
A different quick example



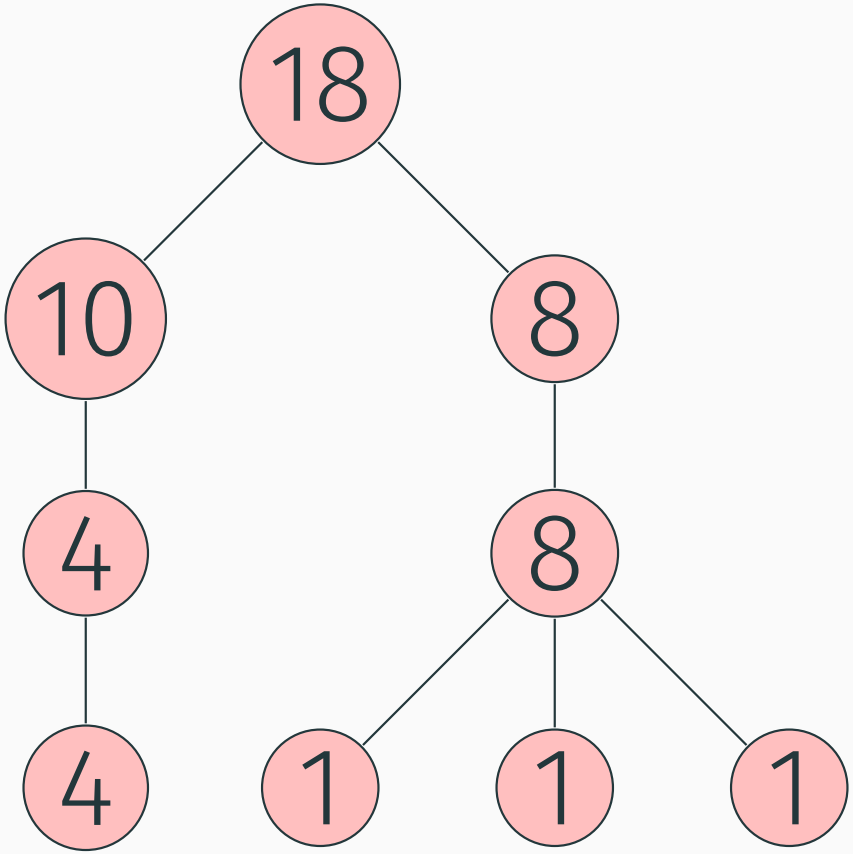
A different quick example



A different quick example



A different quick example



Dynamic programming with trees

This was easy on trees because subtrees rooted at the children of t only communicate with each other through t (or rather, each vertex is a separator)

For more general graphs, can we connect bags of vertices in a tree-like structure, where each bag is a separator on this tree, and bounding the size of these bags?

Tree decompositions

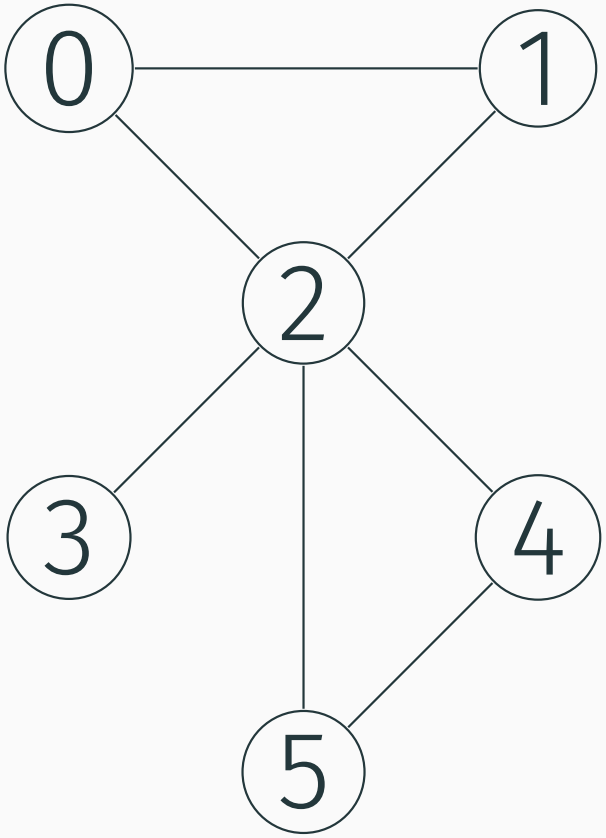
Given a graph G, define a tree T where each vertex t has a *bag*, X_t. This is a *tree decomposition* of G if:

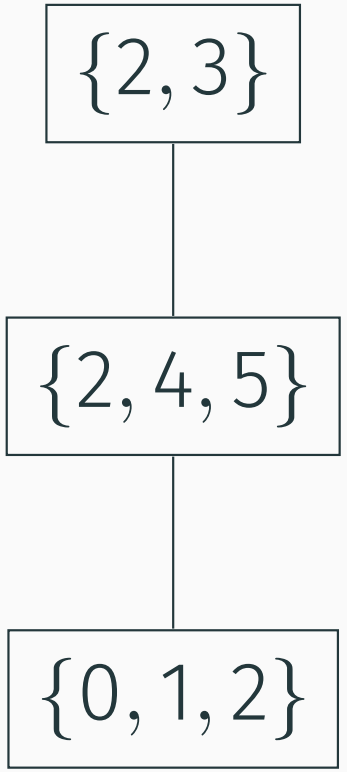
1. Every vertex of V is in at least one bag.
2. The set of nodes with v in their bag form a connected subtree in T for every vertex v.
3. Every edge (v, w) has a bag containing both v and w.

The *treewidth* of a graph G is the smallest maximum bag size over all tree decompositions of G, minus one. Trees and forests have treewidth 1.

Computing treewidth is NP-hard, but plenty of decent approximation algorithms and heuristics are available

Tree decomposition example





Nice tree decompositions

Given a tree decomposition, can construct a nice tree decomposition of the same width.

In a nice tree decomposition, the root bag is empty and each node fits in one of four cases:

- **Leaf nodes** have no children and an empty bag.
- **Forget nodes** have one child and their bag is obtained by ”forgetting” a vertex from the child bag.
- **Introduce nodes** have one child and their bag is obtained by ”introducing” a vertex from the child bag.
- **Join nodes** have two children, each with the same bag.

For a dynamic program based on tree decompositions, just need to consider these 4 cases.

Example: forget case

Two main possibilities here, depending on whether there's other vertices in the bag in the same part.

If not, we can never add anything to this part again - so need to check it has at least B_1 vertices

If so, we might still add things to the part, but we need to make sure our forgotten vertex is connected to something else in the current bag.

Treewidth for SCP

Store a table of DP entries of the form

$$D[t, p', \mathcal{Y}, (\ell_y)_{y\in\mathcal{Y}}, ((\mu_y, M_y)_{y\in\mathcal{Y}}, (\sim_y)_{y\in\mathcal{Y}}]$$

Key idea: $\mathcal{Y}$ partitions the current bag, describing how parts of the partition of V_t intersect the current bag. ℓ_y stores the number of vertices in this part, μ_y and M_y store the minimum/maximum vertex values in the part, and $\sim_y$ is an equivalence relation storing which vertices in y are connected within this part. p' counts the number of parts of the partition of V_t which do *not* intersect the current bag.

Then a given DP entry is true if there's a valid partition in the subgraph induced by vertices in V_t satisfying these properties.

Treewidth result

For each case in the nice tree decomposition, prove correctness and compute the running time of checking all states for each node.

From this, you get that Similar Connected Partition is solvable in XP-time parameterised by treewidth, with a fairly naive analysis based on checking all pairs of states from parent and child nodes and checking if they're compatible.

Definitely room to refine the running time here! Mainly by bounding how many states you need to consider for a node.

Future work

Implement this and assess its viability on real data

Extend to counting/sampling/enumeration

Consider removing limits on part size

Approximability for planar graphs