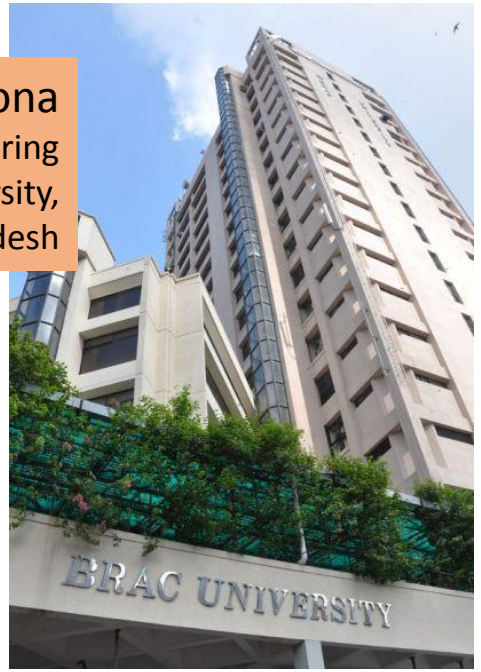
Efficiently Enumerating all Spanning Trees of a Plane 3-Tree

By Muhammad Nur Yanhaona

Research Team

Muhammad Nur Yanhaona
Department of Computer Science & Engineering
BRAC University,
Dhaka, Bangladesh



Asswad Sorkar Nooman
& Md Saidur Rahman
Graph Drawing and Information Visualization Laboratory
Department of Computer Science & Engineering
Bangladesh University of Engineering & Technology
Dhaka, Bangladesh

Image source: https://studybarta.com/brac-university/
https://bd.linkedin.com/in/sm-nahian-al-wasi-685758199?trk=public_profile_browsemap_profile-result-card_result-card_full-click

Presentation Outline

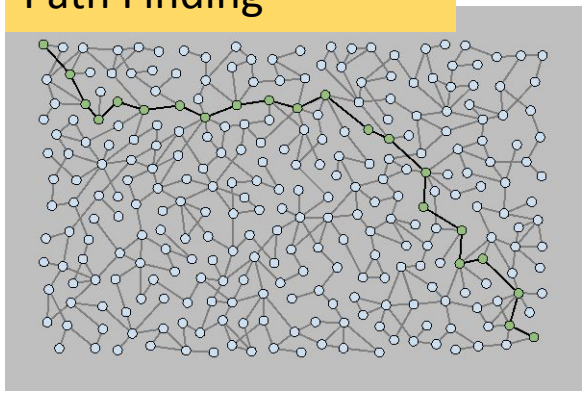
- Motivation
- Related Work
- Proposed Solution
  - General Approach
  - Inductive Algorithm
  - Dynamic Programming Algorithm
- Future Scope

Motivation

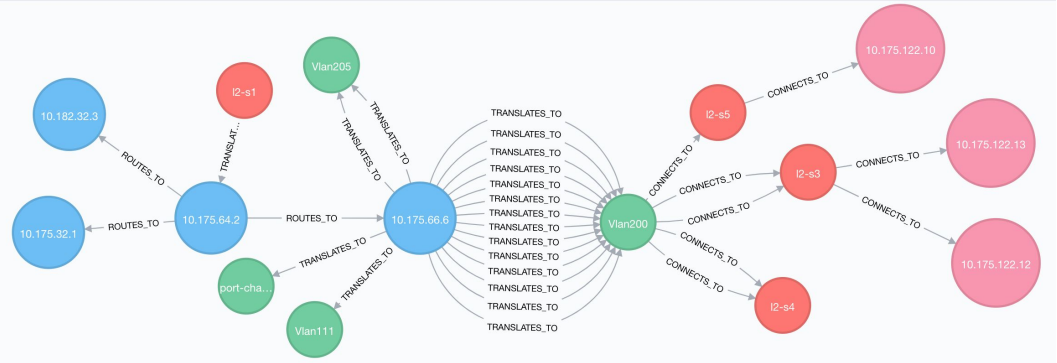
Importance of Spanning Trees

Spanning trees are essential in many algorithms related to path finding, network routing, topological graph embedding, etc. One seek spanning trees of different characteristics for different applications.

Path Finding



Routing



Graph Embedding

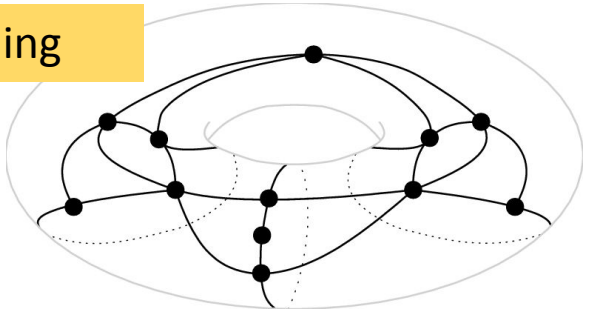


Image source: https://happycoding.io/tutorials/libgdx/pathfinding
https://maxdemarzi.com/2019/03/01/network-routing-in-neo4j/
https://www.researchgate.net/figure/An-embedding-of-a-graph-in-the-torus-Note-that-this-graph-can-also-be-embedded-in-the_fig1_37985042

Motivation

Importance of Spanning Tree Enumeration

- Spanning tree enumeration was a concern as early as Kirchoff's analysis of electrical circuits in 1847.
  - In many early papers, spanning tree enumeration is considered often an efficient solution to spanning tree counting problem which takes $O(n^3)$ using Kirchoff's matrix tree theorem for a graph with n vertices.
    - On trees of a graph and their generation by Hakimi, S, 1961
  - In present day parallel computers, enumeration as an alternative for counting via matrix determinant calculation is not that appealing.
- However, often individual spanning trees are needed, for example:
  - In topological synthesis of networks.
    - Generation and realization of trees and k-trees by Hakimi, S., Green, D., 1964
  - In construction of nets of polyhedra, multi-robots spanning tree routing.
    - Enumeration of Spanning Trees Using Edge Exchange with Minimal Partitioning by Naser Mohamed, 2014

Our work is focused on generating the individual trees as part of the enumeration process.

Related Work

- Char, J.: Generation of trees, two-trees, and storage of master forests. IEEE Transactions on Circuit Theory 15(3), 228–238 (1968). https://doi.org/10.1109/TCT.1968.1082817
- Rakshit, A., Sarma, S.S., Sen, R.K., Choudhury, A.: An efficient tree-generation algorithm. IETE Journal of Research 27(3), 105–109 (1981). https://doi.org/10.1080/03772063.1981.11452333

Time: O(n+m+t+t'), Space: O(nm)

- Gabow, H.N., Myers, E.W.: Finding all spanning trees of directed and undirected graphs. SIAM Journal on Computing 7(3), 280–287 (1978). https://doi.org/10.1137/0207024
- Matsui, T.: An algorithm for finding all the spanning trees in undirected graphs (1998)

Time: O(n+m+nt), Space: O(n+m)

- Kapoor, S., Ramesh, H.: Algorithms for enumerating all spanning trees of undirected and weighted graphs. SIAM Journal on Computing 24(2), 247–265 (1995). https://doi.org/10.1137/S009753979225030X
- Shioura, A., Tamura, A.: Efficiently scanning all spanning trees of an undirected graph. Journal of the Operations Research Society of Japan 38(3), 331–344 (1995). https://doi.org/10.15807/jorsj.38.331

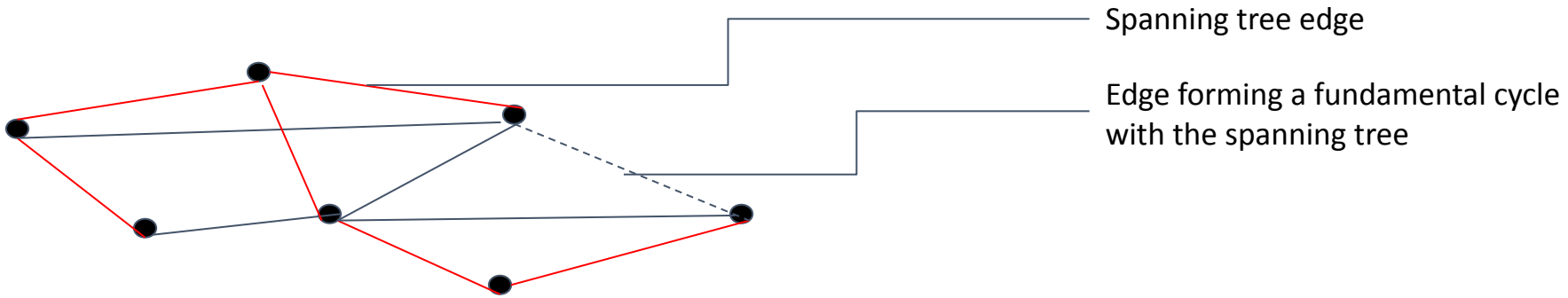
Time: O(n+m+t), Space: O(nm)

Do not output the trees.

Related Work

Analysis of the State of the Art

- The current best algorithms for spanning tree enumeration in general graphs utilizes the property that the set of all spanning trees of a graph form a connected search space.
  - .There is an edge between two spanning trees in the search space if we can switch between them by exchanging a pair of edges.
- Therefore, traversing that search space by forming its spanning tree will enumerate all spanning trees of the original graph.
- The principal part of the time complexity of these algorithms is related to discovering the edges between the nodes of the search space. The underlying logic for this operation is to find fundamental cycles of individual spanning trees.
- The space complexity arises from the need of maintaining an ordered list of edges per node to avoid reentering the same node in the search space during traversal.



Proposed Solution

General Approach

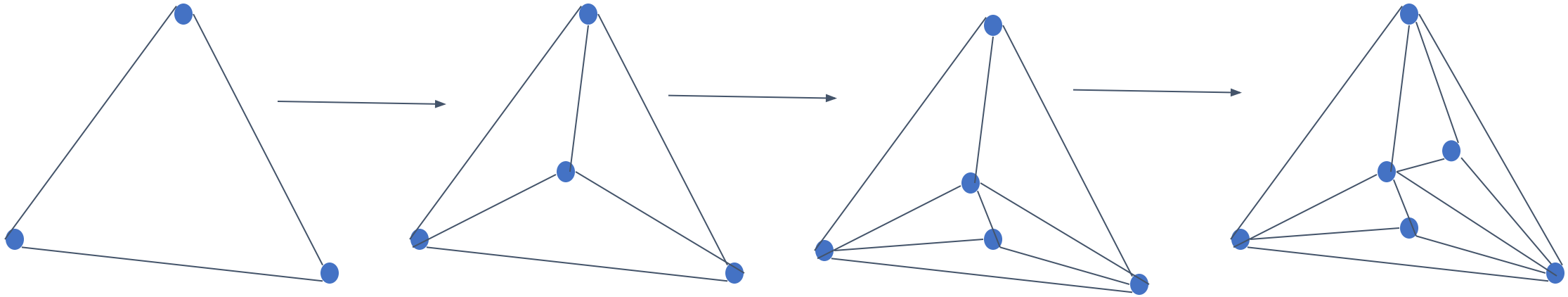
- Instead of approaching the spanning tree enumeration problem as a search space traversal problem, approach it as an inductive generation problem where:
  - There are a fixed number of fundamental extension operations.
  - The vertices of the input graph G_n with n vertices are given a certain order.
  - Then, spanning trees of larger intermediate induced subgraph G_i are generated by extending spanning trees of smaller induced subgraph G_{i-1} by applying those fundamental extension operations.

- Here proper choice of the fundamental extension operations is essential to ensure that all spanning trees are generated.
- A careful ordering of vertices is needed to avoid duplicate tree generation.

This approach has the potential to improve time and space complexities of spanning tree enumeration beyond the state of the art for specific graph classes.

In the paper, we adopt this approach for plane 3-trees.

Plane 3-Tree Definition

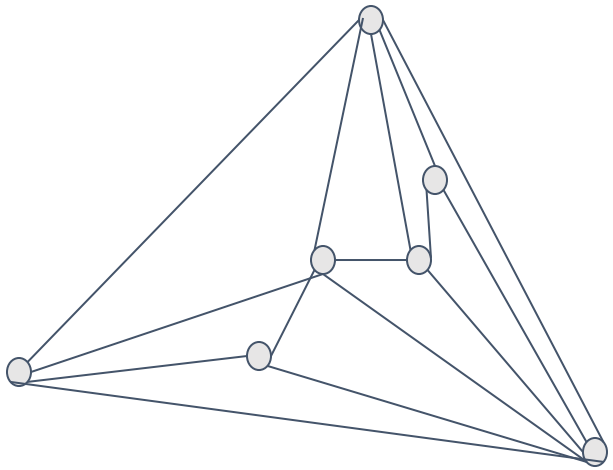


- A plane 3-tree in three vertex is a triangle.
- A plane 3-tree of n vertices is constructed by adding a new vertex of degree 3 to further triangulate a face of a plane 3-tree of n - 1 vertices.

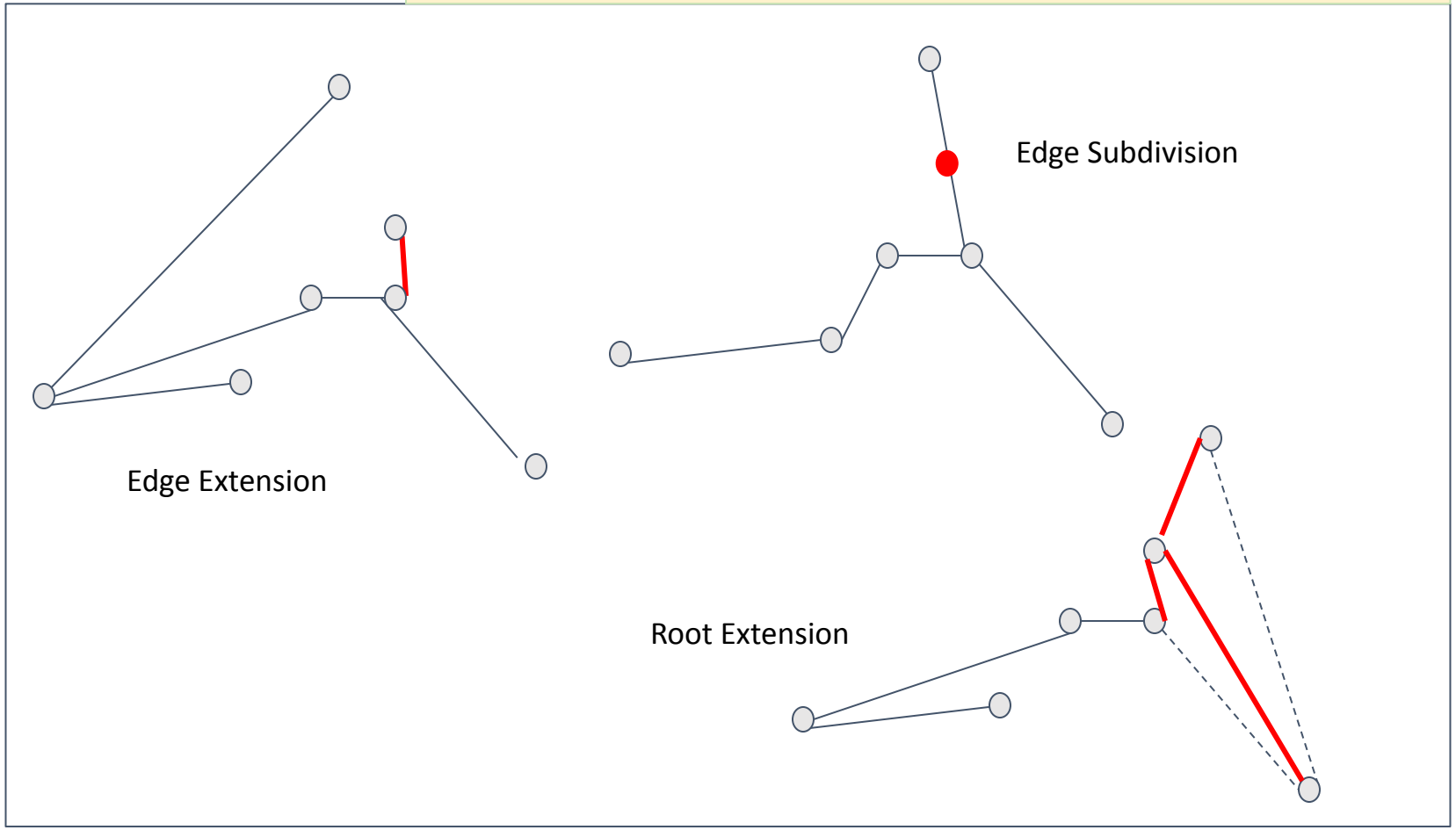
We will refer a plane 3-tree of n vertices as G_n, unless otherwise specified.

Inductive Algorithm

Fundamental Extension Operations



Extension Operations for Plane 3-tree

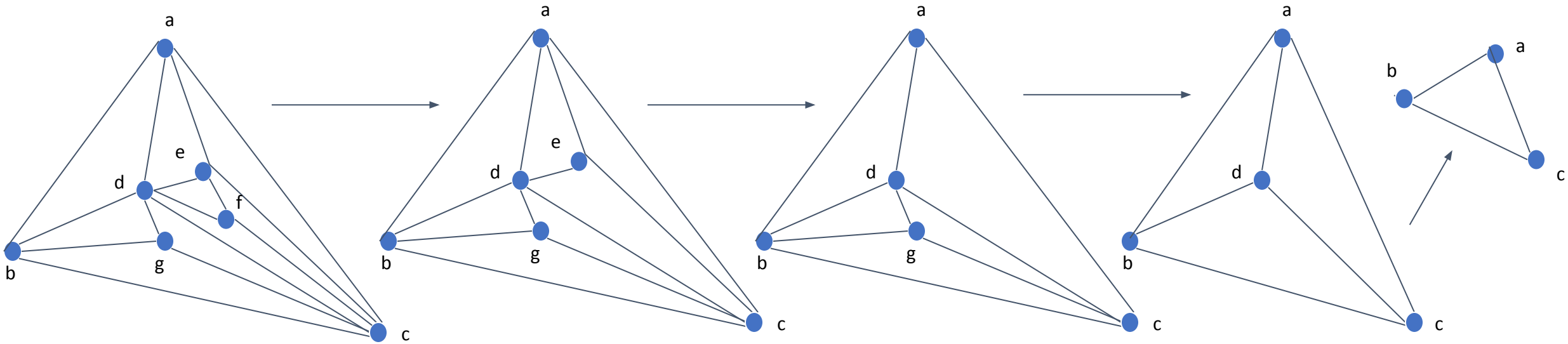


Note that the fundamental extension operations are applicable if and only if the spanning tree of an intermediate graph has a certain topology. That is, not all extensions can be used in all spanning trees.

Inductive Algorithm

Vertex Ordering of Extension Application

- If we repeatedly delete degree-3 vertices of a plane 3-tree and keep track of the faces from where each vertex was extracted during removal, we get an ordering of vertices except for the three vertices of the outer face.
- We call the reverse of that removal order, a dividing vertex sequence and corresponding face description, face vertex set sequence.
- After labeling the vertices of the outer face -2, -1, 0 arbitrarily, we label each vertex based on its position in the dividing vertex sequence. We call this label, the dividing order of a vertex.
- Spanning trees are constructed by adding vertices in intermediate spanning trees in increasing dividing order.



Dividing Vertex Sequence: d, g, e, f
Face Vertex Set Sequence: {a,b,c}, {b,c,d}, {a,c,d}, {c,d,e}
Sorting by Dividing Order: a, b, c, d, g, e, f

Inductive Algorithm

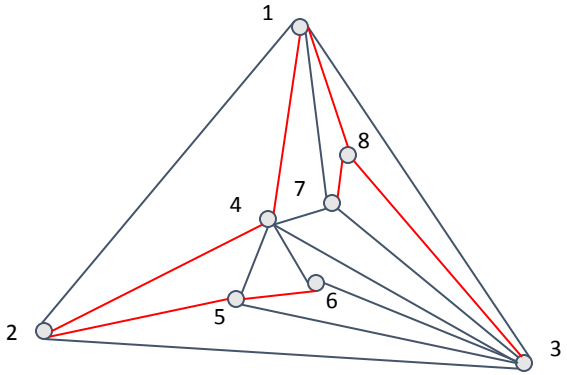
General Description

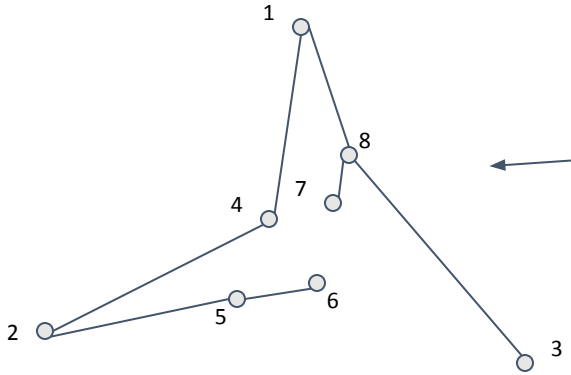
- Starts with a set of spanning trees of G_3
  - which are three paths of length two connecting the outer face vertices.
- Consider the other vertices in increasing dividing order.
- Inductively generate all spanning trees of G_i from all spanning trees of G_{i-1} by adding the i^{th} dividing vertex v_i with face vertex set $\{f_1, f_2, f_3\}$ as follows:
  - For each spanning tree T_{i-1}:
    - Add v_i to using the three possible edge extensions from $\{f_1, f_2, f_3\}$.
    - If two of the three vertices from $\{f_1, f_2, f_3\}$ are adjacent in T_{i-1} then also extend T_{i-1} using edge subdivisions for v_i in all applicable cases.
    - If $\{f_1, f_2, f_3\}$ forms a path of length two in T_{i-1} with the vertex with intermediate dividing order among the three in the middle of the path then also apply a root extension.
  - Add all newly generated T_i in this manner in a growing set of spanning trees for G_i.

In the paper, we proved that this strategy enumerate all spanning trees of Gn without producing duplicates.
However, there are efficiency concern.
Which we address in the next algorithm.

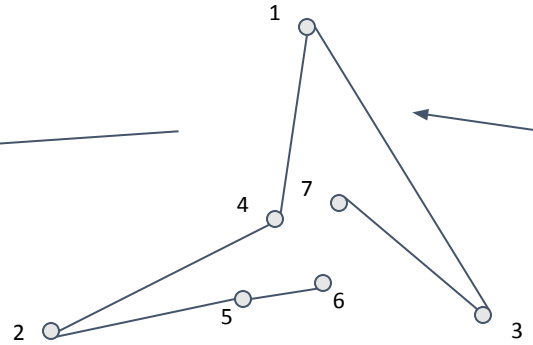
Inductive Algorithm

Illustration of Spanning Tree Construction

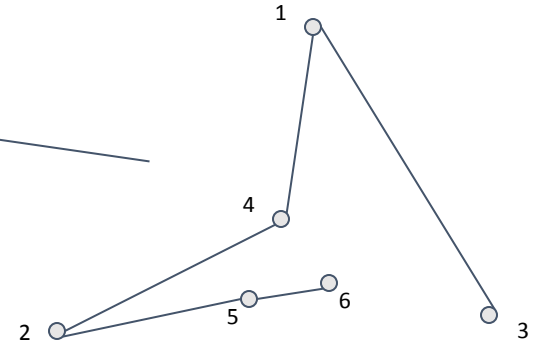




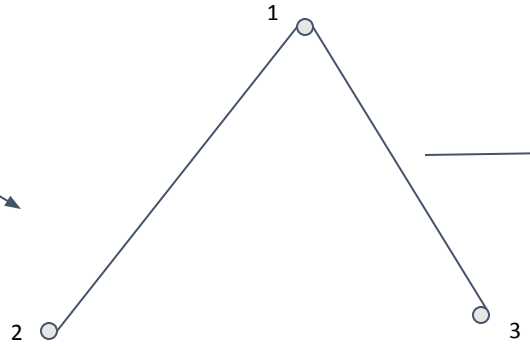
Root Extension for 8



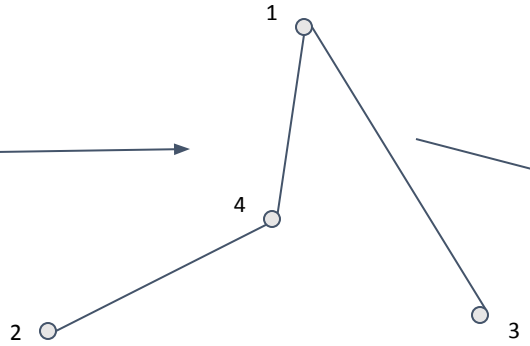
Edge Extension for 7



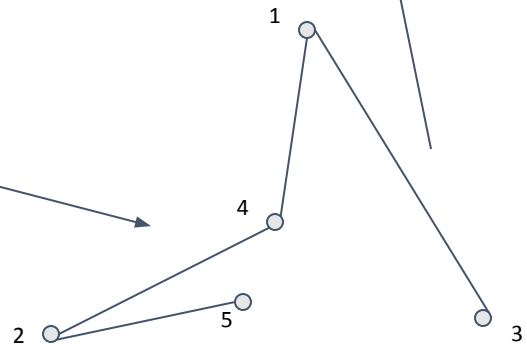
Edge Extension for 6



Starting Tree



Edge Subdivision for 4



Edge Extension for 5

Inductive Algorithm

Limitations and Analysis

- The algorithm requires all spanning trees of G_{n-1} in memory to generate all spanning trees of G_n. That gives the total space complexity of $O(nt)$ for a G_n with t spanning trees.
- Generating a T_i from a T_{i-1} requires an array copy operation, that makes the running time of the algorithm O(n + nt).
- Therefore, the algorithm is not better than the state of the art – it is just different.

However, the inductive algorithm has some key characteristics:

1. Vertices are added in all spanning trees following the same order.
2. There is no coordination/cross-checking between two spanning trees of G_{i-1} when generating spanning trees from G_i.
3. Extension applied for vertex with dividing order i can only affect the connectivity of vertices in its face vertex set during generate a spanning tree.

These characteristics lead to a much more efficient algorithm for spanning tree enumeration using dynamic programming.

Dynamic Programming for Spanning Tree Enumeration

General Strategy

- Instead of following the inductive process of generating all spanning trees for intermediate graph G_i from all spanning trees of immediate predecessor G_{i-1}, we consider the recursive process of adding the remaining vertices of G_n in all possible ways after a sequence of extensions for vertices of G_{i-1} have been made.
- Then, we convert the process of extending smaller spanning trees to larger spanning trees into an alternative process of mutating an existing spanning tree to another by changing what fundamental extension being applied for a particular vertex during the recursive process.

As the fundamental extension operations for a vertex v is applied to attach v with the three vertices in its face vertex set {f1, f2, f3} only, the possible mutation for v are few.
In fact, there are only 7 possible placements of v relative to its face vertex sets.

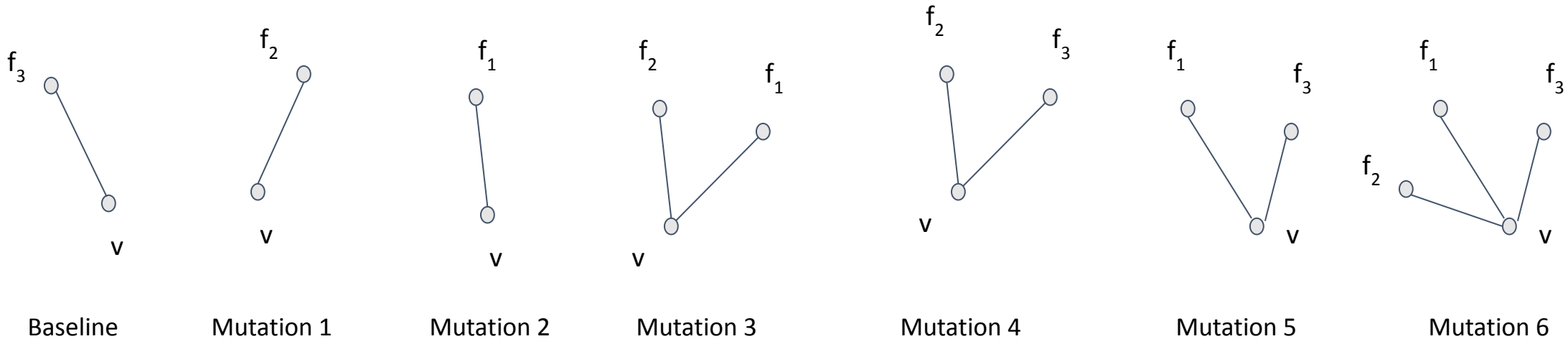
Allows us to enumerate all spanning trees without keeping more than one tree in the memory at a time.

Allows us to reverse and switch to a new mutation during folding and re-unrolling of recursion from specific point without extensive backtrace management .

Dynamic Algorithm

Placement Mutations for a Vertex

Suppose vertex v's face vertex is is {f_1, f_2, f_3} with the following condition holding for their dividing order, dividing_order(f_1) < dividing_order(f_2) < dividing_order(f_3). Then we define a baseline placement of v relative to its face vertex set and possible mutation of the baseline as follows:



Baseline | Mutation 1 | Mutation 2 | Mutation 3 | Mutation 4 | Mutation 5 | Mutation 6

- During recursion unfolding from a specific vertex, v_i, we apply all possible mutations from the baseline for v
- During folding back and returning to vi-1, we revert v to its baseline position.

Dynamic Algorithm

DP Relationship between Parent and Sub-problems.

Assume the vertices of G_n ordered by increasing dividing orders are $v_0, v_1, v_2, v_3, \ldots, v_n$. Then the set of spanning trees of G_n can be expressed using the following DP expression:

t_n = set of all spanning trees of G_n generated by adding v_2 in baseline arrangement

$$\bigcup_{i=1,\ \text{mutation } i \text{ is applicable}}^{6} \{\text{set of all spanning trees of } G_n \text{ generated by adding } v_2 \text{ in } i^{th} \text{ mutation arrangement}\}.$$

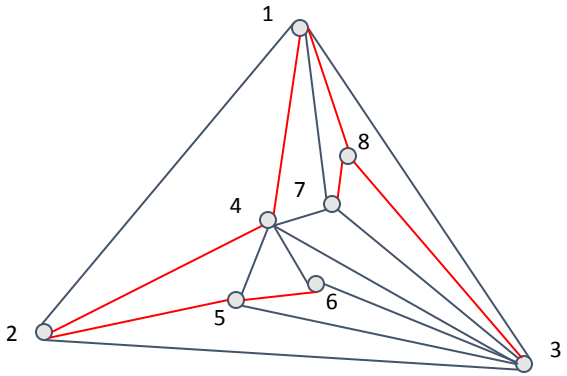
This strategy works due to the same reason inductive spanning tree generation process works:

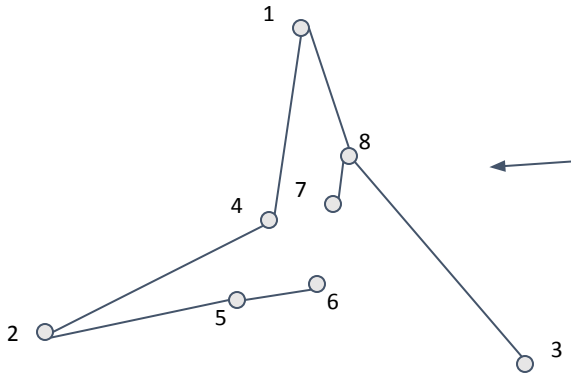
1. Two spanning trees T_1 and T_2 must differ in at least one position in the inductive extension process regarding what fundamental extension operation being used for the corresponding vertex.
2. The same spanning tree cannot be generated by two different sequences of fundamental extension operations.

Inductive Algorithm

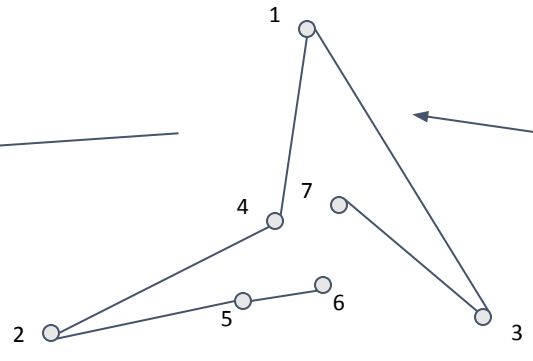
Illustration of Spanning Tree Construction

We will see how this same spanning tree is generated following the DP algorithm.

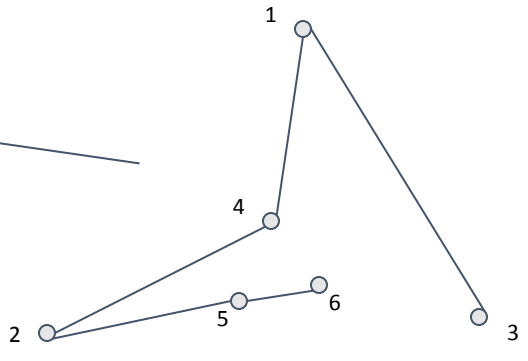




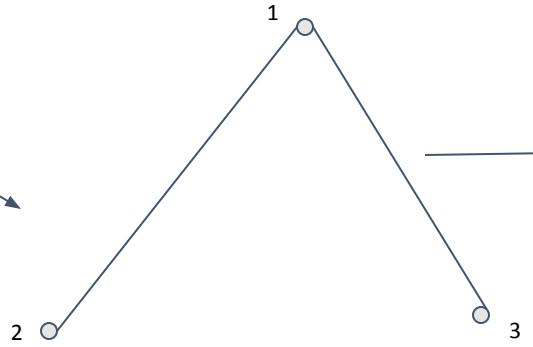
Root Extension for 8



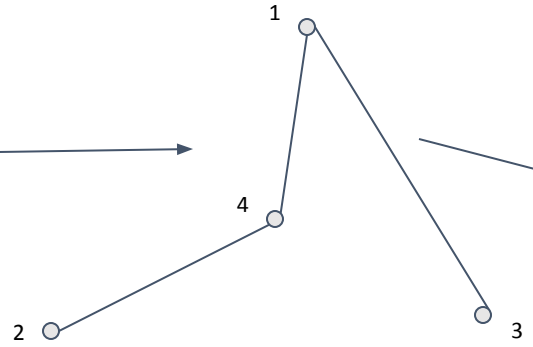
Edge Extension for 7



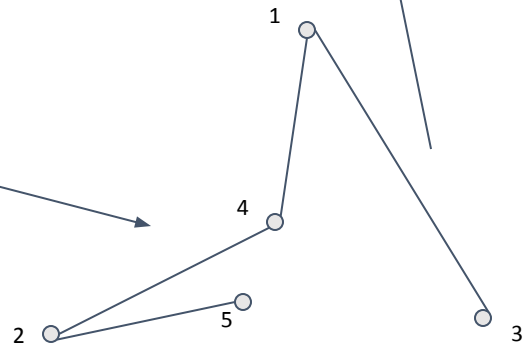
Edge Extension for 6



Starting Tree



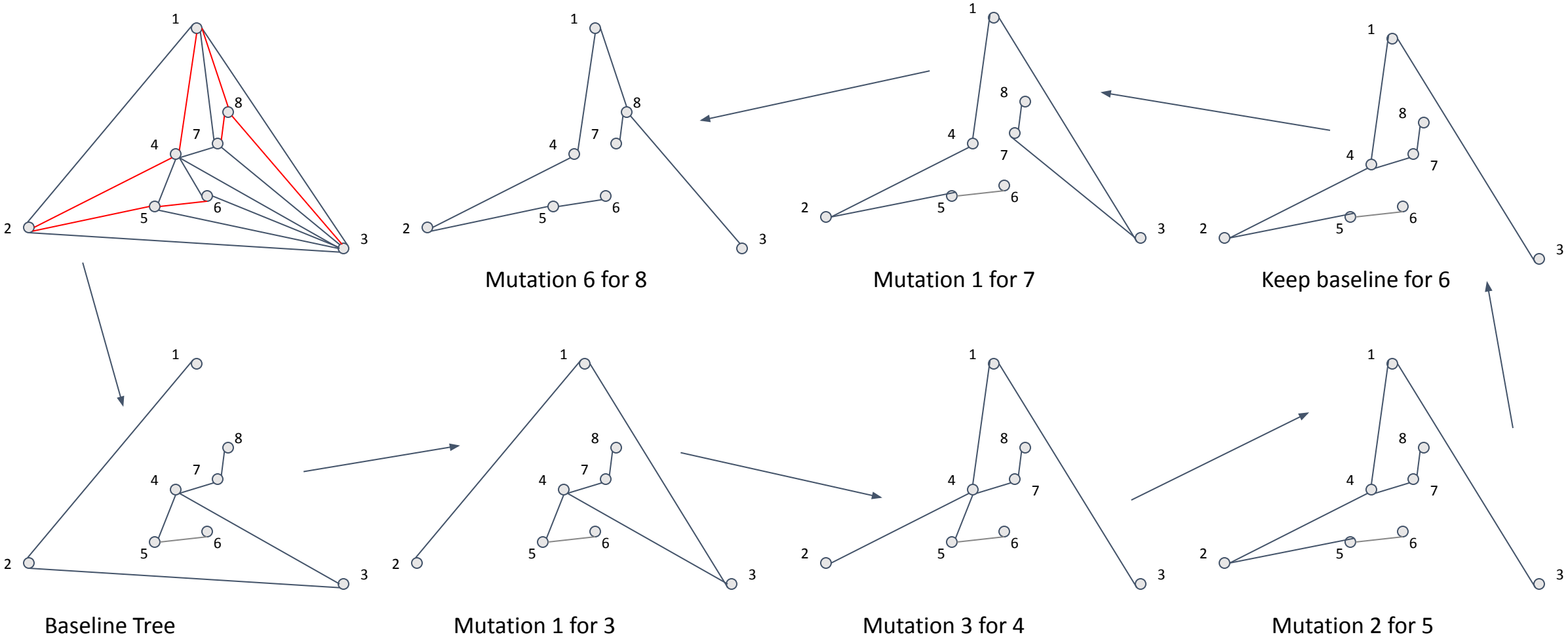
Edge Subdivision for 4



Edge Extension for 5

Dynamic Algorithm

Illustration of Generation of a Specific Spanning Tree



Notice that the DP algorithm will output all of these intermediate spanning trees before outputting the final spanning tree.

Dynamic Programming Algorithm

Performance Factors

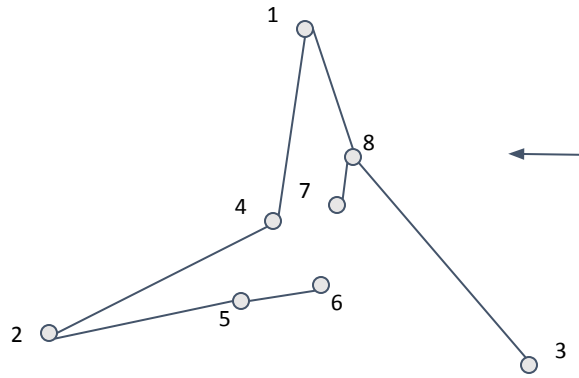
- The DP algorithm only keeps one tree in the memory along with the dividing vertex and the face vertex set sequences. Therefore the space complexity of the algorithm is just $O(n)$.
- The cost of generating the baseline tree is $O(n)$, afterwards the cost of repeated mutation and foldback of mutation when generating the other spanning trees depends on two things:
  - The cost of finding if a certain mutation is applicable in the current spanning tree.
    - As a mutation emulates a fundamental extension, just like an extension, a mutation is possible if the members of the face vertex sets of the current vertex forms a specific connection topology.
    - Applying the mutation and later cancelling it during recursion foldback.

We design a data structure to represent spanning trees that allows both of these operations to be done in constant time.

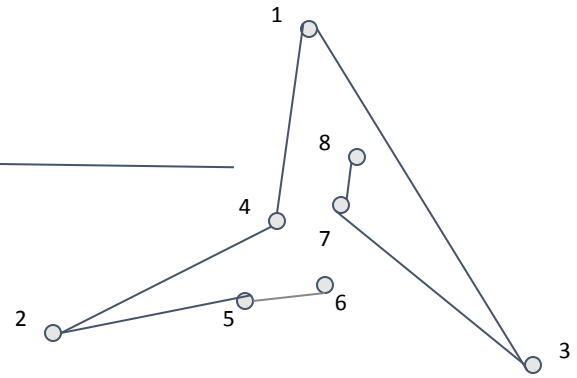
Dynamic Programming Algorithm

Spanning Tree Data Structure

- We maintain an array of length n - 1 to store a spanning tree of G_n.
- Here the i^{th} index of the array holds the dividing order of the vertex that is the other end of an edge connecting vertex with dividing order i - 1.
- Evaluating whether a certain mutation is possible on the current tree for the current vertex under consideration requires probing the array entries correspond to its face vertex sets.
- Applying a mutation or reversing it makes a constant number of changes in the array.
- As array index access is a constant time operation, spanning tree mutation is a constant time operation also.



Mutation 6 for 8



4	1	8	2	5	8	1

4	1	1	2	5	3	7

With this spanning tree data structure, enumerating all spanning trees of G_n having t spanning trees take $O(n + t)$ time.

Future Work

- We believe the approach we propose is generally useful for spanning tree enumeration of different graph classes.
- Regular graphs, k-trees, graphs with a small maximum degree, etc. can be interesting larger classes for future investigation using our approach.

Thanks!