Structural Analysis of Multiwinner Elections

Devansh Kumar Jha [*] Mohan Krishna [†] Labajyoti Das [‡]

April 22, 2024

1 Introduction

Computational social choice is a field at the intersection of social choice theory, theoretical computer science, and the analysis of multi-agent systems. It consists of the analysis of problems arising from the aggregation of preferences of a group of agents from a computational perspective.
Elections are a common phenomenon in real life. They are a particular example in this field of study where a group of agents are involved in collective decision making and for an election rule to be practically applicable, it has to be the case that we can compute the winner of an election in reasonable time.
In this report we present and study the computational aspects of the decision problem underlying an election. In particular we will provide a comprehensive framework for the computation of winning committees in multiwinner elections with special kind of preference profiles. Our report is based on the findings of (Peters et al., 2022).

2 Elections as a game

Practically conducted elections can be formalized into a rigid framework of a game and studied under game theory. This has wide ranging applications in the conduction of elections and participation in one. We will start by formally defining the notion of an election as a game and then ask the natural question - Who will win the election ? And what is the complexity of computation of the winner ?

2.1 Elections

Definition 2.1. Elections are defined as a tuple $\mathcal{G} = (C, N, (P_i)_{i \in N}, \mu)$ where -

- C is the *finite* set of **candidates**
- $N = \{1, 2, 3, ..., n\}$ is the set of **voters**
- For each voter $p_i \in P_i$ assigns a strict total order over C. All such total orders combine to form a **preference profile** $p \in P = P_1 \times P_2 \times ... \times P_n$ for the game. For each $i \in N$ we write $\succ_i$ to represent the **preference order** of i. If $a \succ_i b$ then we say that voter i prefers a to b.
- For each pair of voter and candidate the **score function** is mapping $\mu : N \times C \to \mathbb{Z}$ such that $a \succ_i b$ implies $\mu(i, a) \geq \mu(i, b)$. Intuitively, $\mu(i, a)$ indicates how well candidate a represents voter i.

Notation 2.2. For a given preference profile $p \in P$ and a candidate $a \in C$ and a voter $i \in N$ -

- $pos(i, a)$ denotes the position of candidate a in preference ordering p_i of voter i.
  Mathematically, $pos(i, a) = 1 + |\{b \in C : b \succ_i a\}|$.
- $top(i)$ denotes the most preferred candidate for voter i.

[*]Fourth Year Undergraduate, IIT Kanpur, dkjha20@iitk.ac.in
[†]Fourth Year Undergraduate, IIT Kanpur, mohank20@iitk.ac.in
[‡]Third Year Undergraduate, IIT Kanpur, labajyoti21@iitk.ac.in

- $second(i)$ denotes the second most preferred candidate for voter i.
- $bottom(i)$ denotes the least preferred candidate for voter i.

2.2 Singlewinner Election System

Once an election is defined then a **voting rule** needs to be applied so that a winner can be chosen for a given preference profile p and scoring function μ. There can be many voting rules for an election. If the voting rule chooses only one candidate as the winner then this voting rule alongwith the tuple G emits a **single winner election system**. We define two important voting rules here.

Definition 2.3 (Utility of Voters)**.** For a chosen candidate $a \in C$ and a score function μ the utility that a voter $i \in N$ gets if "a" is the winner of the election is equal to the score $\mu(i, a)$.

Definition 2.4 (Egalitarian Chamberlin-Courant Rule)**.** This rule elects all candidates such that the worst-off voter's utility is the maximum. More precisely, all candidates having the highest egalitarian score win where the egalitarian score for $a \in C$ is defined as

$$score_{\mu}^{min}(p, a) = \min_{i \in N} \mu(i, a)$$

Definition 2.5 (Utilitarian Chamberlin-Courant Rule)**.** This rule elects all candidates such that the sum of all voter's utility is the maximum. More precisely, all candidates having the highest utilitarian score win where the utilitarian score for $a \in C$ is defined as

$$score_{\mu}^{+}(p, a) = \sum_{i \in N} \mu(i, a)$$

In the above single winner setting it is easy to see that the problem of computing the winning candidates can be done in polynomial time using a simple brute force mechanism. In fact we state the following proposition without proof.

Proposition 2.6. Given a single winner election system $G = (C, N, (P_i)_{i \in N}, \mu)$ with Egalitarian Chamberlin-Courant Rule, the set of all winning candidates can be computed in time $poly(n, m)$ where $n = |N|$ and $m = |C|$. Same result also applies for Utilitarian Chamberlin-Courant Rule.

2.3 Multiwinner Election System

As the single winner system is easily tractable thus it is not of much theoretical interest. However, there exist voting rules where a committee $W \subseteq C$ is chosen as the winner. In such cases the problem of computing the winning committee becomes a lot more difficult. If the voting rule chooses a committee of candidates as the winner then this rule alongwith the tuple G emits **multi winner election system**. To study this system we start by extending our notation.

Notation 2.7. Given a committee $W \subseteq C$ and a preference profile $p \in P$ for a voter $i \in N$ -

- $top(i, W)$ denotes the most preferred candidate for voter i among the candidates in W.
- $second(i, W)$ denotes the second most preferred candidate for voter i among the candidates in W.
- $bottom(i, W)$ denotes the least preferred candidate for voter i among the candidates in W.

In a multi winner system we would also need a generalization of the previously defined notion of utility of a voter and the voting rules. In general the above introduced voting rules can be generalized as follows

Definition 2.8 (Utility of Voters)**.** For a chosen committee $W \subseteq C$ and a score function μ the utility that a voter $i \in N$ gets if W wins the election is equal to $\mu(i, top(i, W))$.

Definition 2.9 (Egalitarian Chamberlin-Courant Rule)**.** This rule elects all committees such that the worst-off voter's utility is the maximum. More precisely, all committees having the highest egalitarian score win where the egalitarian score for any $W \subseteq C$ is defined as

$$score_{\mu}^{min}(p, W) = \min_{i \in N} \mu(i, top(i, W))$$

Definition 2.10 (Utilitarian Chamberlin-Courant Rule)**.** This rule elects all committees such that the sum of all voter's utility is the maximum. More precisely, all committees having the highest utilitarian score win where the utilitarian score for any $W \subseteq C$ is defined as

$$score_{\mu}^{+}(p, W) = \sum_{i \in N} \mu(i, top(i, W))$$

It is immediately evident from the above setting that the problem is now not trivially solvable using simple methods. In fact we will see in due course that computation of winning committees is a hard problem. Here we define formally the decision problems associated with the optimization version for these rules. These problems are central to the study of multiwinner elections.

Definition 2.11 (EGALITARIAN CC)**.** An instance of the EGALITARIAN CC problem is given by a preference profile $p \in P$, a committee size k, $1 \leq k \leq |C|$, a scoring function $\mu : N \times C \to \mathbb{Z}$, and a bound $B \in \mathbb{Z}$. It is a "yes"-instance if there is a subset of candidates $W \subseteq C$ with $|W| = k$ such that $score_{\mu}^{min}(p, W) \geq B$ and a "no"-instance otherwise.

Definition 2.12 (UTILITARIAN CC)**.** An instance of the UTILITARIAN CC problem is given by a preference profile $p \in P$, a committee size k, $1 \leq k \leq |C|$, a scoring function $\mu : N \times C \to \mathbb{Z}$, and a bound $B \in \mathbb{Z}$. It is a "yes"-instance if there is a subset of candidates $W \subseteq C$ with $|W| = k$ such that $score_{\mu}^{+}(p, W) \geq B$ and a "no"-instance otherwise.

3 Computational aspect of Multiwinner Elections

3.1 Complexity for Egalitarian Chamberlin-Courant rule

The below result is a well known concept in social choice theory and is detailed in (Betzler et al., 2013). We will provide a brief proof here for the sake of completeness of the report.

Theorem 3.1. An instance of the EGALITARIAN CC PROBLEM is NP-complete.

For the above we will use the following problem for a NP-Completeness reduction. This problem will also be used for another similar reduction later for the Egalitarian CC rule.

Definition 3.2 (HITTING SET PROBLEM)**.** An instance of the Hitting Set problem is given by a ground set C, a family $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ of subsets of C, and a target cover size $k \in \mathbb{Z}_+$. It is a 'yes'-instance if there is a subset of vertices $W \subseteq C$ with $|W| \leq k$ such that $W \cap C_i \neq \emptyset$ for $i = 1, \ldots, n$, and a 'no'-instance otherwise.

The above problem is known to be NP-Complete from (Garey and Johnson, 1979). We will reduce this problem to our problem to proof the hardness of the EGALITARIAN CC problem.

Definition 3.3 (Positional Scoring Functions)**.** A scoring function is said to be positional if there exists a vector $\mathbf{s} = (s_1, s_2, ..., s_m) \in \mathbb{Z}^m$ where $m = |C|$ with $s_i \geq s_{i+1}$ for $1 \leq i < m$ such that $\mu(i, a) = s_{pos(i,a)}$. When this is the case we say that the scoring function is induced by vector $\mathbf{s}$.

Definition 3.4 (Borda Scoring Function)**.** The positional scoring function induced by $\mathbf{s} = (0, -1, ..., -(m + 1))$ is called the Borda Scoring Function.

Proof of Theorem 3.1.

$$(C, N, (p_i)_{i\in N}, \mu, k, B) \to (C, N, (p_i)_{i\in N}, \mu`, k, 0)$$

An instance of the EGALITARIAN CC with borda score function can be reduced to an instance of the UTILITARIAN CC with borda scoring and score bound $B = 0$. Formally, stating it consider

An instance of the problem of finding winning committees is given by $\mathcal{E} = (C, N, p, \mu, T, k, B)$ where T is the singly peaking tree and an instance of the tree hitting problem is given by $\mathcal{T} = (C', \mathcal{C}', T', k')$. We define a mapping $\mathcal{E} \to \mathcal{T}$ as follows.

$T' = T$, $k' = k$ and $C' = C$. We construct the set of subsets of C' given by $\mathcal{C}'$ by the following equation

Let W' is the tree hitting set got by running algorithm 3 on $\mathcal{T}$ with $|W| \leq k'$ then we construct a committee $W = W'$ for $\mathcal{E}$ which will be a winning committee due to the following which completes the reduction.

$$\forall i \in N \Big(\exists a \in W \text{ with } \mu(i, a) \geq B \implies \mu(i, top(i, W)) \geq B \Big) \implies score_{\mu}^{min}(p, W) \geq B$$

□

3.2 Complexity for Utilitarian Chamberlin-Courant rule

Theorem 3.5. An instance of the UTILITARIAN CC PROBLEM is NP-complete.

The above result can be obtained in a manner similar to the case of Egalitarian CC rule but we will provide a stronger result in Section 3.7 where we would prove the NP-Completeness of Utilitarian CC for a sub-class of multiwinner elections which automatically implies the above result.

3.3 Need for simplified subclasses of Multiwinner Elections

The problem of finding the winning committee for a multiwinner election in general is NP-Complete for both Egalitarian Chamberlin-Courant rule and Utilitarian Chamberlin-Courant rule. Thus, we would now like to find some 'nice' structural properties which can act as suitable simplifications to the general class of multiwinner elections. In the next section we will see a special kind of preference profile which was first introduced in (Demange, 1982) and that forms a very well studied subclass of multiwinner elections.

4 Preference profile Singly Peaked on a tree

4.1 Preliminaries

Definition 4.1 (Profile singly peaked on a tree)**.** A preference profile $p \in P$ is said to be singly peaked on a tree if there exists a tree $T = (C, E)$ where the vertices are given by the set C of candidates and edges are such that for every voter $i \in N$ and every pair of distinct candidates $a, b \in C$ if b lies on the unique path from $top(i)$ to a in T then $top(i) \succ_i b \succ_i a$.

Definition 4.2 (Single Peaked Profile)**.** The profile p is said to be single-peaked if p is single-peaked on some tree T that is a path.

Definition 4.3 (Top Initial Segment)**.** A subset of candidates $W \subseteq C$ is a top initial segment for a particular voter $i \in N$ if either $W = C$ or there exists a candidate $a \in C$ such that $W = \{b \in C \mid b \succ_i a\}$.

The definition 4.1 immediately implies the following proposition which basically shows the equivalence of the ideas of connectedness in the singly peaking tree and connectedness in the preference profile given by top initial segments.

Proposition 4.4. Let $p \in P$ be a preference profile and let T be a tree on the candidate set C (as the vertex set). The following properties are equivalent:

- p is single peaked on T.

- For every $W \subseteq C$ that is connected in T, $p|_W$ is single peaked on $T|_W$.
- For every $i \in N$ and every $a \in C$, the top initial segment $\{b \in C \mid b \succ_i a\}$ is connected in T.

Notation 4.5. Given a profile p, $\mathcal{T}(p)$ denote the set of all trees T such that p is single-peaked on T.

Once a class of problems is restricted a subclass we would first like to check if it is even tractable to find whether a problem belongs to this subclass. Infact the algorithm detailed below was first given in (Trick, 1989a) which shows that the decision problem $\mathcal{T}(p) = \emptyset$ or not is tractable in polynomial time.

4.2 The Trick's algorithm

We would start by introducing a few propositions, which would help to intuitively understand Trick's algorithm and understand why is it correct.

4.2.1 Building the algorithm

Proposition 4.6. Suppose P is single-peaked on T,and suppose a occurs as a bottom-most alternative, that is, bottom(i) = a for some i ∈ N. Then a is a leaf of T.

Proof. The preference ordering of i^{th} voter is: $C_{i_1} \succ_i C_{i_2} \succ_i \ldots \succ_i C_{i_N}$, $C_{i_k} \in C, k \subseteq [N]$. $C_{i_N} = a$. Hence. removing a, the set C $\setminus\{a\}$ is a top-initial segment, hence (by equivalence of properties of single-peaked trees) must be connected in T. This can only be possible if a is a leaf of T. □

Proposition 4.7. Suppose P is single-peaked on T, and suppose a ∈ C is a leaf of T, adjacent to $b \in C$. Let $i \in N$ be a voter. Then either:

- b = top(i) and b = second(i), or
- b $\succ_i$ a.

Proof. If $a = top(i)$, then $\{a, second(i)\}$ is a top-initial segment, hence connected in T. Since a is a leaf (degree=1), and b is it's (only) neighbour, b has to be second(i).
Otherwise, $a \neq top(i)$. Consider some $c \in C$ s.t $c = top(i)$. The path from c to a: Path(c,a) is connected (since it's a top-initial segment for voter i). Since b is a's only neighbour, it has to be present in Path(c,a). Note that i's preference decreases monotonically along Path(c,a), hence $b \succ_i a$. □

Intuitively, to construct tree T single-peaked on P, (given a candidate a) we would want to keep a track of all possible neighbours of a (from which we could choose one strategically). For the same, introduce the following notations(where B stands for neighbourhood set):

Notation 4.8.

$$\forall i \in N\text{: } B(i,a) = \begin{cases} c \in C : c \succ_i a, & \text{if } top(i) \neq a, \\ second(i), & \text{if } top(i) = a. \end{cases}$$

$$B(a) = \bigcap_{i \in N} B(i,a) \text{ ; Using proposition 4.2 on B(i,a).}$$

$$B(i,a) \subseteq B(a)$$

From the above discussions, the following is obvious:

Corollary 4.9. It is necessary for leaf a to be adjacent to some element in B(a).

Next, we show: Given profile P is single-peaked on a tree, there exists some tree T corresponding to any possible (a,b), where $b \in B(a)$. This is helpful both to construct a single-tree using Trick's algorithm and storing all possible trees (edges to all possible neighbours from a).

Proposition 4.10. Let P be a profile in which a occurs bottom-ranked. Suppose that $P|_{C\setminus\{a\}}$ is single-peaked on some T_{-a} with vertex set $C\setminus\{a\}$, and let T be a tree obtained from T_{-a} by attaching a as a leaf adjacent to some element $b \in B(a)$. Then P is single-peaked on T.

Proof. Note that if we are able to prove that all possible top-initial segments across all possible voters i are connected in T, we are done. Hence, pick any voter i, and any top-initial segment $c \subseteq C$ for i.
We need to show S is connected in T:

- If $a \notin S$, then since $P|_{C\setminus\{a\}}$ is single-peaked on T_{-a} , S(=S $\setminus\{a\}$) is connected in T.
- S = $\{a\}$ case is trivial. Consider $a \in S, S \neq \{a\}$. S $\setminus\{a\}$ is connected in T_{-a}. Hence if any of the possible a's neighbours $b \in B(a)$ is in S $\setminus\{a\}$, then S is connected in T (and we are done).
  Since $B(i,a) \subseteq B(a)$ we have $b \in B(i,a)$. We use the following two cases to show that $b \in S$:
  - If top(i) = a , then b is forced to be second(i). Since S is top-initial segment, top(i) and second(i) $\in S$. Hence, b is in S. Therefore, S is connected.
  - If $top(i) \neq a$, B(i,a) = $\{c : c \succ_i a\}$ => $b \succ_i a$. Since S is top-initial segment (ending at leaf a), we must have $b \in S$.

□

4.2.2 Final sketch of the algorithm

Intuitively, we would like to add upward edges starting from potential-leaves (and inductively construct the full tree). Using the above propositions, at every iteration we construct a leaf-set (bottom-ranked candidates from current candidate-set). For every element in this leaf-set, we add an edge to some to potential neighbour. Henceforth, we delete candidates in the leaf-set and start again with a smaller candidate-set. Base case is where number of candidates is 2, where we add an edge arbitrarily, or 1 where we do nothing.

Algorithm 1: Algorithm to construct a Singly Peaking Tree

Input : A preference profile p
Output: The singly peaking tree T

```
W ← ∅;
while T is not empty do
    Let a be a leaf vertex in T;
    Let b be a vertex adjacent to a in T;
    if there exists C_i ∈ 𝒞 such that C_i = {a} then
        Add a to W;
    end if
    Remove a from T;
    Remove all copies of {a} from 𝒞;
end while
return W
```

Algorithm 2: Decide whether a profile is single-peaked on a tree

```
1 T ← (C, φ), the empty graph on C //Digraph C_1 ← C, r ← 1 while |C_r| ≥ 3
2 do- L_r ← {bottom(i, C_r) : i ∈ N} for each candidate a ∈ L_r do-
3     B(a) ← ⋂_{i∈N} B(i, C_r, a) if B(a) = φ then-
4         return fail : P is not single-peaked on any tree else-
5             select b ∈ B(a) arbitrarily //All add an edge between a and b in T
6     C_{r+1} ← C_r \ L_r r ← r + 1
7 if |C_r| = 2 then
8     add an edge between the two candidates in C_r to T
9 return P is single-peaked on T
```

The algorithm returns a tree, since we add $|C| - 1$ edges (one in each for loop iteration, for each candidate) and T is connected (since at the end of algorithm, every vertex has an edge to C_r, C_r being singleton or connected set of size 2).

4.2.3 Correctness of the algorithm

Theorem 4.11 (Trick's Theorem)**.** The above algorithm 1 correctly constructs a tree T on which a preference profile p is singly peaked if $\mathcal{T}(p) \neq \emptyset$ and returns failure if $\mathcal{T}(p) = \emptyset$.

Proof. For correctness, we use induction on leaves/bottom-ranked candidates. Base cases are trivial.
Inductive hypothesis: Given algorithm is on step i, it is is correct on step i+1 (on smaller candidate-set). In any iteration if B(a) is empty, then by corollary 4.4 P(on remaining candidate set) is not single-peaked. Hence P (on full candidate set) is not single-peaked(by Proposition 4.5) Else after every iteration we are in effect running the algorithm again on $P|_{C_2}$, and we use Proposition 7.4 along with inductive hypothesis to deduce that algorithm is indeed correct. □

This algorithm will be studied again and tweeked a little bit so that we can construct a more useful data structure from the same algorithm which will help us construct more useful trees from the set $\mathcal{T}(p)$ rather than just constructing arbitrary trees.

4.3 Egalitarian Chamberlin-Courant on Arbitrary Trees

Theorem 4.12. For preference profiles that are single-peaked on a tree T, we can find a winning committee under the Egalitarian Chamberlin-Courant rule in time $poly(n, m)$.

For the proof of the above theorem we introduce a variant of the hitting set problem first.

Definition 4.13 (TREE HITTING SET PROBLEM)**.** An instance of the Tree Hitting Set problem is given by a tree T on a vertex set C, a family $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ of subsets of C such that each C_i is connected in T, and a target cover size $k \in \mathbb{Z}_+$. It is a 'yes'-instance if there is a subset of vertices $W \subseteq C$ with $|W| \leq k$ such that $W \cap C_i \neq \emptyset$ for $i = 1, \ldots, n$, and a 'no'-instance otherwise.

We state and provide a brief proof of the below result from (Guo and Niedermeier, 2006).

Theorem 4.14. TREE HITTING SET PROBLEM can be solved in time $poly(|\mathcal{C}|, |C|)$.

Proof. For a leaf vertex $a \in C$ having a unique neighbor $b \in C$ we have $a \in C_i \implies C_i = \{a\}$ or $b \in C_i$ as every C_i is connected in T. This observation can be used to build a simple greedy algorithm as shown in Algorithm 3 which returns a tree hitting set of minimal size. □

Proof of Theorem 4.12.

$$\mathcal{E} = (C, N, p, \mu, T, k, B) \rightarrow \mathcal{T} = (C', \mathcal{C}', T', k')$$

Algorithm 3: Algorithm to construct a Tree Hitting Set

Input : A class of subsets $\mathcal{C}$ of C and a tree T
Output: The tree hitting set W

$W \leftarrow \emptyset$;
while *T is not empty* **do**
  Let a be a leaf vertex in T;
  Let b be a vertex adjacent to a in T;
  if *there exists $C_i \in \mathcal{C}$ such that $C_i = \{a\}$* **then**
    Add a to W;
  end if
  Remove a from T;
  Remove all copies of $\{a\}$ from $\mathcal{C}$;
end while
return W

An instance of the problem of finding winning committees is given by $\mathcal{E}$ where T is the singly peaking tree and an instance of the tree hitting problem is given by $\mathcal{T}$. We define a mapping $\mathcal{E} \to \mathcal{T}$ as follows. $T' = T$, $k' = k$ and $C' = C$. We construct the set of subsets of C' given by $\mathcal{C}'$ by the following equation

$$\mathcal{C}' = \left\{ \{a \in C \mid \mu(i,a) \geq B\} \;\middle|\; \forall i \in N \right\}$$

Let W' is the tree hitting set got by running algorithm 3 on $\mathcal{T}$ with $|W'| \leq k'$ then we construct a committee $W = W'$ for $\mathcal{E}$ which will be a winning committee due to the following which completes the reduction.

$$\forall i \in N \left(\exists a \in W \text{ with } \mu(i,a) \geq B \implies \mu(i, top(i,W)) \geq B \right) \implies score_{\mu}^{min}(p,W) \geq B$$

□

4.4 Utilitarian Chamberlin-Courant on Arbitrary Trees

Theorem 4.15. Given a preference profile p that is single-peaked on a tree T, a target committee size k, and a target score B, it is NP-COMPLETE to decide whether there exists a committee of size k with score at least B under the Utilitarian Chamberlin-Courant rule.

Rather than proving this theorem we will prove a much stronger result by showing that the hardness holds even for a very restricted subclass of singly peaked profile multiwinner elections. This in some sense will show that we are still far away from getting tractability.

Theorem 4.16. Given a preference profile $\mathcal{P}$ that is single-peaked on a tree $\mathcal{T}$, a target committee size k, and a target score B, it is NP-COMPLETE to decide whether there exists a committee of size k with score at least B under the Utilitarian Chamberlin-Courant rule with the Borda scoring function. Hardness holds even restricted to profiles single-peaked on a tree with diameter 4 and pathwidth 2.

For the reduction required to be done for this problem we will be using the well known Vertex cover problem already well known to be NP-Complete from (Karp, 1972).

Definition 4.17 (VERTEX COVER PROBLEM)**.** An instance of the VERTEX COVER problem is given by an undirected graph $G = (V, E)$ and $t \in \mathbb{Z}_+$. It is a 'yes'-instance if there exists a subset $S \subseteq V$ of vertices such that $|S| \leq t$ and $(u, v) \in E \implies u \in S$ or $v \in S$, and a 'no'-instance otherwise.

Proof of theorem 4.16.

$$(G = (V, E), t) \to (C, N, (p_i)_{i \in N}, \mu, k, B, T)$$

The tuple $(G = (V, E), t)$ denotes an instance of the vertex cover problem and the other tuple denotes various components of our problem with T denoting the singly peaking tree that we assume to exist for

the given profile p.
Define $M = 5mn$ and construct $k = n + t$ and $B = 5m(n - t) + 2m$. Also construct μ to be the Borda Scoring Function. Here $n = |V|$ and $m = |E|$. The tree T is as shown in the figure 1. Now only the construction for C, N and $(p_i)_{i \in N}$ is left which is done as follows.

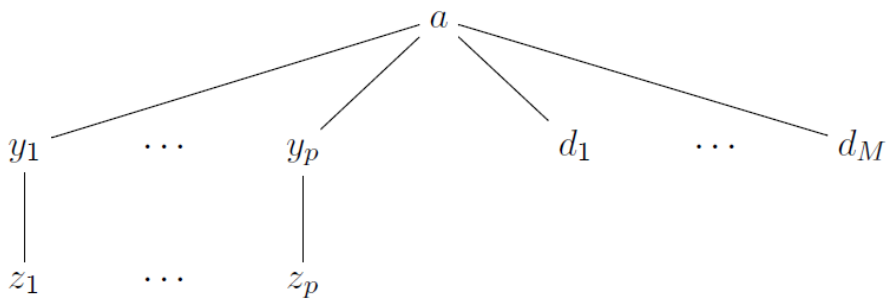

Figure 1: Singly peaking tree for construction for theorem 4.16

$$C = \{a\} \cup \{y_1, y_2, ..., y_n\} \cup \{z_1, z_2, ..., z_n\} \cup \{d_1, d_2, ..., d_M\}$$

For every vertex in $u_i \in V$ we make two candidates y_i and z_i for $1 \leq i \leq n$ and also make a lot of dummy candidates given by $D = \{d_1, d_2, ..., d_M\}$.

$$N = N_1 \cup N_2 \cup N_3$$

- N_1 has $5m$ identical voters for each $u_i \in V$: they rank y_i first, z_i second, and a third followed by any order that satisfies the constructed tree T.
- N_2 has 1 voter for each $e_j \in E$: they rank a first, followed by y_{j_1} and y_{j_2} followed by $d_1, ..., d_M$ followed by any order that satisfies the constructed tree T.. Here j_1 and j_2 are respectively the endpoints of the edge e_j.
- N_3 has M identical voters for each $u_i \in V$: they rank z_i first, y_i second, and a third followed by any order that satisfies the constructed tree T.

Intuitively the construction of voters is such that it generates huge penalties if any candidate in Z is not selected and also enforces selection of the other t candidates from Y. Below is a brief argument to show that this construction works.

- $\implies$ If any candidate z_i is not selected in final committee then there exist atleast M voters in N_3 which impose a penalty of atleast 1 each so the total penalty is atleast $M > B$ which is a contradiction. Similarly, if the number of candidates from Y in winning committee is $< t$ then atleast $5m(n-t+1)$ voters from N_1 give a penalty of 1 each thus the total penalty $> B$ which is a contradiction. Now as $k = n + t$ thus no more candidates can be selected. Assume that the selected candidates from Y does not correspond to a vertex cover in G then there is atleast 1 voter in N_2 which gives a penalty of atleast $(M + 3) > B$ which is again a contradiction.
- $\impliedby$ If we start from a "yes" instance of the vertex cover problem with vertex cover S then we can take a winning committee $W = Z \cup \{y_i | i \in S\}$ then voters in N_3 and and $5mt$ voters in N_1 give no penalty. A total of $5m(n - t)$ voters give 1 penalty each in N_1 and the maximum penalty that any voter in N_2 gives is 2 so total cost $\leq 5m(n - t) + 2m \implies \leq B$.

□

4.5 Utilitarian Chamberlin-Courant on Trees with Few Leaves

Theorem 4.18. Given a profile p with $|C| = m$ and $|N| = n$, a tree T with λ leaves such that p is single-peaked on T, and a target committee size k, we can find a winning committee under the utilitarian Chamberlin-Courant rule in time $poly(n, m^{\lambda}, k^{\lambda})$.

For the above theorem's proof we will build a dynamic programming algorithm which is a generalization to the algorithm from (Betzler et al., 2013) used to find winning committees when preference profile is singly peaked on a path. For this we first define a term as follows.

Definition 4.19. A set $A \subseteq C$ is said to be an antichain if no two elements are comparable with respect to $\succ$ using the information encoded in the singly peaking tree T. An example is shown in 2.

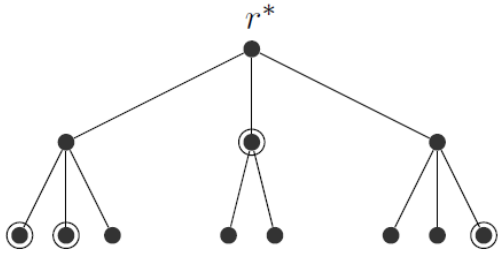

Figure 2: An Antichain

Proof of theorem 4.18. For any $r \in C$ we define T_r to be the subtree rooted at r in T. The vertex set of T_r is $C_r = \{r\} \cup \{a | r \succ a\}$ and $N_r = \{i \in N | top(i) \in C_r\}$ are the related voters. If p_r is the restriction of the preference profile on C_r then for all $r \in C$ and $1 \leq l \leq |C_r|$ we take $M(r, l)$ to be the maximum score of a committee in the restricted elections in T_r.

$$M(r,l) = max\left\{score_{\mu}^{+}(p_r, W) : W \subseteq C_r \text{ with } |W| = l \text{ and } r \in W\right\}$$

To compute this value we consider $\mathcal{T}_r$ to be the set of all antichains in T_r and consider all possible divisions of each of the antichains in components with atleast 1 candidate.

$$M(A,L) = \sum_{j=1}^{s} M(r_j, l_j) + \sum_{i \in N_r'} \mu(i, top(i, A \cup \{r\}))$$

In a selected antichain $A\{r_1, r_2, ..., r_s\} \in \mathcal{T}_r \setminus \{\{r\}\}$ of size $(l-1)$ considering all possible divisions of it $M(A, L)$ denote the score of the committee with antichain A and division L. We would need to take maximum of all possible cases here denoted by the next equation.

$$M(r,l) = max_{A \in \mathcal{T}_r \setminus \{\{r\}\}, L \in \mathcal{L}_{l-1}^{A}} M(A, L)$$

Here $\mathcal{L}_{l-1}^{A}$ denote the set of all divisions of an antichain A of size $(l-1)$ into its constituents.

□

4.6 Utilitarian Chamberlin-Courant on Trees with Few Internal Vertices

Theorem 4.20. Given a profile P with $|C| = $ m and $|N| = $ n, a tree $\mathcal{T} \in \mathcal{T}(\mathcal{P})$ with η internal vertices such that $\mathcal{P}$ is single-peaked on $\mathcal{T}$, and a target committee size $k \geq 1$, we can find a winning committee of size k for $\mathcal{P}$ under the Chamberlin-Courant rule with the Borda scoring function in time $poly(n, m, (k+1)^{\eta})$.

Definition 4.21 (Plurality of a candidate)**.** Given a candidate $a \in C$ the plurality of the candidate is given by $plu(a) = |\{i \in N | top(i) = a\}|$. This denotes number of voters represented by candidate $a \in C$.

Proof of theorem 4.20. If $C^o \subseteq C$ denote the set of all internal vertices in T and $lvs(c)$ denote the set of leaf candidates in $C \setminus C^o$ which are adjacent to $c \in C^o$ then the following algorithm works

- Select a strict ordering $\triangleleft$ over the candidates which will be used to break ties during the entire run.
- For every $c \in C^o$ make a guess for the pair $(x(c), l(c))$ where $x(c) \in \{0, 1\}$ denotes whether c is selected and $0 \leq l(c) \leq |lvs(c)|$ denotes how many candidates in $lvs(c)$ are in the committee.
- Construct a committee where all $c \in C^o$ with $x(c) = 1$ are added and $l(c)$ leaf candidates in $lvs(c)$ are added in order of the plurality of the candidates breaking ties using $\triangleleft$.
- Compute the score for the chosen committee and iterate the process over the complete set of guesses possible for $\{x(c), l(c)\}$.

Now it remains to prove that this simple strategy would indeed work. The proof of correctness of this algorithm involves majorly contradiction proves.
The reason why this is the best committee is because as we have chosen the leaf candidates based on the plurality, it is always better to include the candidate with higher plurality in the committee. Also, iterating over all possible cases automatically considers the best allocation for internal vertices.

□

5 Framework for Computation of Winning Committees

In order to obtain "nice" trees (for the algorithms described above), let's try to create a data structure which compactly stores all possible trees in $\mathcal{T}$(p) (which are singly-peaked on T). We will call this structure "attachment digraph" D=(C,A) on a candidate set C, which stores all possible arcs in set A. After recognizing some special properties of attachment digraph, we would develop algorithms to find nice trees (minimum number of leaves/internal vertices) by systematically "choosing" desired arcs from D. Attachment Digraph will be constructed by tweaking the Trick's algorithm for constructing a singly-peaked tree T on profile P.

5.1 Building Attachment Digraph

In Trick's algorithm, we consider all possible choices of neighbours B(a) and add all corresponding arcs to the digraph. The purpose of adding arcs (instead of edges) is to get some nice properties.

Algorithm 4: Build attachment digraph D = (C,A) of P

```
1 D ← (C, A), the empty digraph on C C_1 ← C, r ← 1 while |C_r| ≥ 3
2   do- L_r ← {bottom(i, C_r) : i ∈ N} for each candidate a ∈ L_r do-
3       B(a) ← ⋂_{i∈N} B(i, C_r, a) if B(a) = φ then-
4           return fail : P is not single-peaked on any tree else-
5               for each b ∈ B(a), add an arc (a,b) to A
6   C_{r+1} ← C_r \ L_r r ← r + 1
7 if |C_r| = 2 then
8   add an arc between the two candidates in C_r to A, arbitrarily directed
9 return D
```

Time complexity: $O(|N|.|C|^2)$.
Consider the following illustration. Suppose C = $\{a, b, c, d, e, f, g, h, i, j, k\}$ and let P be the profile with voters N = $\{1, 2, 3\}$ such that:

$$k \succ_1 f \succ_1 \succ_1 e \succ_1 d \succ_1 g \succ_1 h \succ_1 c \succ_1 i \succ_1 j \succ_1 b \succ_1 a$$
$$d \succ_2 c \succ_2 b \succ_2 e \succ_2 a \succ_2 f \succ_2 g \succ_2 h \succ_2 i \succ_2 j \succ_2 k$$
$$g \succ_3 f \succ_3 h \succ_3 i \succ_3 e \succ_3 d \succ_3 c \succ_3 b \succ_3 a \succ_3 j \succ_3 k$$

Running above algorithm, we get $L_1 = \{a, k\}$, $L_2 = \{b, j\}$. $L_3 = \{c, i\}$, $L_4 = \{d, h\}$ and $L_5 = \{e, g\}$. Note the various possible combinations of possible edges (grey). Note that if at any instant some candidate is top(i) for any i, it will always have exactly 1 outgoing edge (since it's B set will be a singleton). We would use this property later on.

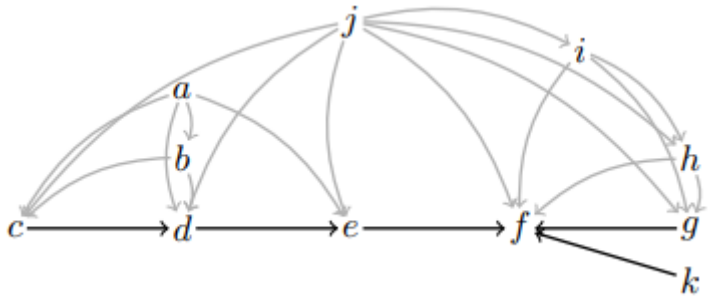

Figure 3: Example for construction of Attachment Digraph

5.2 Properties of Attachment Digraphs

Throughout this subsection, we have profile P fixed, C_r is the candidate set, L_r is the leaf-set and B(a) is neighbourhood set in the r^{th} iteration of algorithm to construct attachment digraph.

Proposition 5.1. Let $a \in C$ be a candidate with $a \in L_r$. Then $B(a)L_r = \phi$. Hence, for every arc $(a, b) \in A$ with $a \in L_r$ and $b \in L_s$, we have that $s > r$.

Proof. For L_r to be defined (within the while-loop), we require $|C_r| \geq 3$. Note that all the elements in L_r set are bottom-ranked (w.r.t candidate set C_r). Consider any element $b \in B(a)$. For i^{th} voter, then by definition either a = top(i,C_r) and b = second(i,C_r), or $b \succ_i a$. We deduce that b cannot be bottom-ranked in either cases. The first case is handled by the fact that $|C_r| \geq 3$, so b can't be both second and bottom-ranked. For the second case, if b is bottom-ranked, then $b \succ_i a$ isn't possible. Therefore, $B(a)L_r = \phi$. Second statement is proved as follows: $\because (a, b) \in A$, $b \in B(a)$. $B(a) \subseteq C_r = c\backslash(L_1 \cup L_2 \cup \ldots L_{r-1}$ and $b \in L_s$. Hence $s > r$, since $s \neq r$ by above arguments. □

Theorem 5.2. Every attachment digraph D = (C,A) is acyclic and has exactly one sink.

Proof. Let R-1 denote the number of while-loop iterations of algorithm to construct attachment digraph.

- Note that $L_1, \ldots, L_R$ is a partition of C. By above proposition, we can construct a topological ordering of D (since there are arcs only from elements of L_r to elements of L_s, s¿r). Hence, D is acyclic.
- $\exists$ at least 1 sink. No vertex in $C\backslash L_R$ is a sink , $\because \exists$ outgoing arc to some element in L_s, s¿r. Hence, we only need to consider possible elements in L_R for sinks.
- If $|L_R| = 1$, we have the unique sink. Else $|L_R| = 2$. Since we add an arc between the 2 candidates, we have exactly 1 sink (on which the arc's head points).

□

We have attachment digraph D = (C,A), and in order to construct nice trees, we need to extract $|C| - 1$ arcs from A s.t. we get a connected graph (tree). For the same, we choose exactly 1 outgoing arc for all candidates except the sink. Hence we are motivated to define attachment function:

Definition 5.3. $f : C\backslash\{t\} \to C$. f specifies one outgoing arc $(a, f(a))$ for each $a \in C\backslash\{t\}$.

$$T(f) = \{\{a, f(a)\} : a \in C\backslash t\}\} \tag{1}$$

Theorem 5.4. P is single-peaked on a tree T **iff** the set of edges of T is obtained by picking exactly one outgoing arc for each non-sink vertex of D and converting it into an undirected edge. $T \in \mathcal{T}(P)$ **iff** T=T(f).

Proof.
- If T=T(f), T is a possible output of Trick's algorithm, by making proper choices of $b \in B(a)$, where $a \in L_r$ in each iteration.
- Converse: Induction on $|C|$ ($\because C_{r+1} = C_r \backslash L_r$), L_1 consists of bottom-ranked candidates (leaves in induced sub-graph). Base case: $|C| \leq 2$, where we have unique tree, hence we get unique f. Otherwise, for the bottom-ranked candidates, (from the algorithm) we obtain unique neighbours. Hence we have a tuple (a,f(a)), where $a \in L_r$. Since L_r are the leaves of tree $T|_{C_r}$, $T|_{C_{r+1}}$ is also a tree, s.t. $P|_{C_{r+1}}$ is single-peaked on it. By induction hypothesis, $T|C_{r+1} = T(f')$ for some attachment function f' defined for $D|_{C_{r+1}}$. By induction, we have T=T(f).

□

From above proof, we can observe that the the mapping from f to T(f) is one-one. Hence, we have the following corollary:

Corollary 5.5. The number of trees in $\mathcal{T}(P)$ is equal to the product of the out-degrees of the non-sink vertices of D. Hence we can compute $|\mathcal{T}(P)|$ in polynomial time.

5.3 Circumtransitivity

Intuitively, a circumtransitive digraph consists of an outer (free) part which is transitively attached to inner (forced) part. The formal definition is as follows:

Definition 5.6. Given directed acyclic graph D = (C,A). Partition C into sets $C^{\rightarrow}$ of forced vertices and $C^{\Rightarrow}$ of free vertices, such that:

1. Every forced vertex $a \in C^{\rightarrow}$ has out-degree at most 1. If $(a, b) \in A$ then $b \in C^{\rightarrow}$ as well.
2. Every free vertex $a \in C^{\Rightarrow}$, has out-degree at least 2. If $a, b \in C^{\Rightarrow}, c \in C^{\rightarrow}$ s.t. $(a, b), (b, c) \in A$, then $(a, c) \in A$ as well.

Note that for each forced vertex a, the value of f(a) is uniquely determined.

Theorem 5.7. Every attachment digraph (C,A) is circumtransitive.

Proof. We use the fact that if any candidate a is top-most ranked for any voter, then B(a) can be at-most 1 (hence out-degree can be at-most 1); and vice versa (i.e. if out-degree is at-most 1, then the candidate is top-most ranked for some voter). Therefore, forced vertex should be top-most ranked for some i, and free vertex cannot be top-most ranked.

Let R-1 be the total number of while-loop iterations of algorithm to construct attachment digraph, hence we have a partition of candidate set C. Denote degree of candidate a by d(a).

1. Forced: Let $a \in C^{\rightarrow}$.
   - If d(a)=0, a is vacuously a forced vertex.
   - Consider d(a) = 1. If $(a, b) \in A$ s.t. $d(b) \in \{0, 1\}$, then we are done. $a \in L_r, b \in L_s$.
   - If possible, assume $d(b) \geq 2$. This means that b isn't top-most ranked for any voter (by above); and secondly $r < s < R - 1$ since neither r nor s are sinks. Consider $(b, c) \in A$, $c \in C_r$.
   - $\because c \in B(b) \Rightarrow c \succ_i b \ \forall$ i. $B(i, C_r, A) = \{x \in C_r : x \succ_i a \ \forall \ i \in N$, since it isn't possible that top(i,C_r) = a ($\because$ then second (i,C_r) = b by definition, and $c \succ_i b$ isn't possible).
   - Since B(a) = $\{b\}$, therefore $b \succ_i a \ \forall \ i \in N$. $c \succ_i b$, $b \succ_i a$; hence $c \succ_i a \ \forall \ i \in N$.
   - By definition, this means that $c \in B(a)$.
     Hence $B(a) = \{b, c\}$, a contradiction, since a has degree at-most 1.

- Hence, b should also have degree at most 1.

2. Free: Consider a , b , c $\in C$, with a , b $\in C^{\Rightarrow}$ and $(a,b),(b,c) \in A$. $a \in L_r$, $b \in L_s$ where $r, s < R$. The inequality is strict since a,b (whose degree is ≥ 2 cannot be in L_R (whose elements have degree at most 1). By argument used in start of the proof, $top(i, C_r) \neq a, b \; \forall \; i \in N$.
$(a,b),(b,c) \in A \Rightarrow b \in B(i, C_r, a)$ and $c \in B(i, C_r, b) \; \forall i \in N$. Therefore, $b \succ_i a$ and $c \succ_i b$ for all i $\in N$. By transitivity, $c \succ_i a \; \forall i \in N$. So, $c \in B(a)$. $\therefore (a,c) \in A$.

□

Theorem 5.8. Suppose $|C| \geq 3$. For every free vertex $a \in C^{\Rightarrow}$ of the attachment digraph D = (C,A), there are two forced vertices $b, c \in C^{\rightarrow}$ with $(a,b),(a,c),(b,c) \in A$.

Proof. We complete the proof in 4 steps. $a \in C^{\Rightarrow}$ is a free vertex:

1. There is a forced vertex $b \in C^{\rightarrow}$ with $(a,b) \in A$.

   Proof. Since the directed acyclic graph D has a unique sink t, we have a directed path $a = c_1 \rightarrow c_2 \rightarrow \ldots \rightarrow c_p = t$. Consider the minimum-length path possible. Note that c_2 must be forced, since otherwise there would exist an arc (c_1, c_3) by transitivity of free-vertex. This contradicts the fact that we chose a minimum-length path. □

2. There are at least two forced vertices $b, c \in C^{\rightarrow}$ with $(a,b),(a,c) \in A$.

   Proof. We know that degree of a(free vertex) is less than 2. If possible, consider that a forced vertex 'a' can have outgoing arc to only one other forced vertex. Consider the maximum possible value of r such that L_r contains such an a. In that case, $\exists$ free vertex $b \in C^{\Rightarrow}$ s.t. $(a,b) \in A$. Since b is a free-vertex, $\exists$ forced vertices x , y $\in C^{\rightarrow}$ s.t. $(b,x),(b,y) \in A$ and using circumtransitivity $\exists (a,x),(a,y) \in C$. But this contradicts the fact that a has outgoing arc to only one other forced vertex. Therefore, the claim 2 holds true. □

3. The set $\{b \in C^{\rightarrow} : (a,b) \in A\}$ induces a subtree in $\mathcal{G}(D)$.

   Proof. Consider $a \in L_r$, and $(a,b),(a,c) \in A$ where $b, c \in C^{\rightarrow}$. $|B(a)| > 1$ and $|B(i, C_r, a)| > 1$. Hence, a cannot be top-most preference of any voter i; $b \succ_i a, c \succ_i a$. Hence, consider the top-initial segment of i wrt a, call it W. Note that $D|_{C^{\rightarrow}}$ is connected since the graph is like a linked-list (ending at sink). Hence $\mathcal{G}(D|_{C^{\rightarrow}})$ is a subtree of any possible tree $T \in \mathcal{T}(P)$. There is a unique path Q from b to c in this subtree (and hence, in original tree T). Call this C_Q. Since P is singly-peaked on T, W is connected in T and $C_Q \subseteq W$. Since we're focused only on the r^{th} iteration, C_Q contains only the elements in C_r. Hence $C_Q \subseteq B(i, C_r, a)$. Intersection across all voters, we get $C_Q \subseteq B(a)$. Since $C_Q \subseteq C^{\rightarrow}$, we have $C_Q \subseteq \{x \in C^{\rightarrow}\} : (a,x) \in A$. Hence, by changing notations, $\{b \in C^{\rightarrow}\} : (a,b) \in A$ is connected in $\mathcal{G}(D|_{C^{\rightarrow}}$ □

4. There are two forced vertices b , c $\in C^{\rightarrow}$ with $(a,b),(b,c),(a,c) \in A$.

   Proof. The set $\{b \in C^{\rightarrow}\} : (a,b) \in A$ is connected and contains at least 2 members. Hence, there must exist some vertices b,c s.t. $(b,c) \in A$. □

□

6 Recognition Algorithms: Finding Nice Trees

6.1 Minimum Number of Internal Vertices

Here our intuition is that the graph of forced vertices remains constant throughout all trees in the set.

Algorithm 5: Find T ϵ T(P) with fewest internal vertices

Let $D = (C, A)$ be the attachment digraph of P.
Let $C^{\rightarrow}$, $C^{\Rightarrow}$ be the sets of forced and free vertices in D.
Let t be the sink vertex of D.
$f \leftarrow \emptyset$, be the attachment function under construction.
For each $a \in \rightarrow C \setminus \{t\}$ **do**
$f(a) \leftarrow b$ where b is the unique $b \in C$ with $(a, b) \in A$.
if $\|C^{\rightarrow}\| = 2$
then pick some $c \in C^{\rightarrow}$
For each a $\in C^{\Rightarrow}$ **do**
$f(a) \leftarrow c$.
else if $\|C^{\rightarrow}\| > 2$ **then**
for each $a \in C^{\Rightarrow}$ **do**
find c $\in$ C$^{\rightarrow}$ such that (a,c) $\in A$ and c is internal in $G(D - C^{\rightarrow})$
f(a) $\leftarrow c$.
Return $T^* = T(f)$.

Theorem 6.1. Algorithm 3 runs in polynomial time and returns a tree $T \in T(P)$ with the minimum number of internal vertices among trees in $T(P)$.

Proof: The fact that our algorithm is polynomial is immediate from the description of the algorithm. For starters the fact that we are choosing from the possible arcs from the attachment digraph already makes it so that our output tree is a valid one.

Now suppose the size of the forced vertices is 2 then the output graph is simply a star in which case the leaves are truly maximised.

If it is greater than 2 then it is already clear from the algorithm that any vertex that is a leaf in the graph of forced vertices is a leaf in the main tree as well and the rest of all free vertices are leaves in the algorithm itself, and hence this algorithm maximizes the number of leaves.

6.2 Minimum Diameter

Theorem 6.2. Algorithm 3 returns a tree $T \in T(P)$ with the minimum diameter among trees in $T(P)$.

Proof: We have already stated that in case the size of the forced vertices is 2 in that case the tree is a star and hence the diameter is minimum.

Otherwise, let us just take the longest path as going from c_1, c_2, ..., c_k. From the algorithm, it is clear that only c_1 and c_k have the potential to be free vertices. Now, we have that c_2 is an internal vertex in the graph of forced vertices then we can always replace c_1 with a forced vertex $c_1^{'}$ not already in the longest path and similarly for the case of c_{k-1}. This means that the entire longest path can always be replaced with a similar path from the graph of forced vertices which would always remain consistent in all trees and hence minimum.

6.2.1 Minimum Number of Leaves

Now we present an algorithm that gives us a tree with the minimum number of leaves. The intuition is to convert the problem to that of finding the maximum matching so that the internal nodes get maximized.

Algorithm 6: Find T ϵ T(P) with fewest leaves

Let $D = (C, A)$ be the attachment digraph of P. $f \leftarrow \emptyset$, the attachment function under construction.
Let t be the sink vertex of D, and let $s \in C^{\rightarrow}$ be a forced vertex with a unique outgoing arc $(s, t) \in A$ (such a vertex exists by circumtransitivity of our trees).
$f(s) \leftarrow t$.
Construct a bipartite graph H with vertex set $L \cup R$ where $L = \{l_a : a \in C \setminus \{s, t\}$ and $R = \{r_a : a \in C\}$, and edge set $E_H = \{\{l_a, r_c\} : (a, c) \in A\}$.
Find a maximum matching $M \subseteq E_H$ in H.
For each $a \in C \setminus \{s, t\}$ **do**
  if l_a is matched in M, i.e. $\{l_a, r_c\} \in M$ for some $c \in C$ **then**
    f(a) $\leftarrow c$
  else
    take any $c \in C$ with $(a, c) \in A$
    f(a) $\leftarrow c$
Return$T^* = T(f)$.

Before going into the proof of correctness let's look at how we define internal vertices

Definition 6.3. A non-sink vertex a is *internal* if the attachment function f has some b which maps to a otherwise if a is sink then it is an *internal* vertex if there are at least two vertices c, d which map to a.

Theorem 6.4. Algorithm 4 runs in polynomial time and returns a tree $T \in T(P)$ with the minimum number of leaves among trees in $T(P)$

Proof: We claim that if a vertex is matched as r_c during the algorithm if and only if it will be an internal vertex in the tree .

Let us look at the if side of things. If c is not a sink then c getting matched as r_c makes it internal otherwise if c is a sink then it already has an s and the algorithm will add another a which will be matched to c in the attachment function f making it internal.

The only if direction is obvious because if it is not the case then we can say that there will be a matching which will have a higher number of internal vertices contradicting the maximality of this algorithm.

Suppose there is a tree with higher internal vertices than our tree. Then for that tree we can create similar $\{l_a , r_c\}$ pairs for each vertex c and a being one of the vertices from which the function f maps to c. And thus we have a matching that is higher than the maximum matching M contradicting its maximality.

6.3 Minimum Max-Degree

Now we look at an algorithm that enables us to find minimum max-degree. Here we simply use the concepts of flow networks to get our tree.

Algorithm 7: Decide whether there is T ϵ T(P) with maximum degree at most k

Let $D = (C, A)$ be the attachment digraph of P.
Let t be the sink vertex of D.
Let $L = \{l_a : a \in C \setminus \{t\}\}$ and $R = \{r_a : a \in C\}$, and construct a flow network H on vertex set $\{x, z\} \cup L \cup R$ with arc set
$E_H = \{(x, l_a) : a \in C \setminus \{t\}\} \cup \{(l_a, r_b) : a, b \in C, (a, b) \in A\} \cup \{(r_a, z) : a \in C\}$, and capacities cap(x, l_a) $= 1$ for all $a \in C \setminus \{t\}$, cap$(l_a, r_b) = 1$ for all $(a, b) \in A$, cap$(r_a, z) = k - 1$ for all $a \in C \setminus \{t\}$, and cap$(r_t, z) = k$.
Find a maximum flow in H.
$f \leftarrow \emptyset$, the attachment function under construction.
If the size of the maximum flow is $|C| - 1$ **then**
  for each (a, b) $\in A$ such that the arc (l_a, r_b) is saturated,
    set $f(a) \leftarrow b$.
  Return $T^* = T(f)$.
Else return there is no $T^* \in T(P)$ with maximum degree at most k.

Theorem 6.5. Algorithm 5 runs in polynomial time and returns a tree $T \in T(P)$ with maximum degree at most k if one exists

Proof: Here we simply want to show that max degree of the graph produced from this algorithm does not exceed k. So our problem reduces to deciding weather there exist f such that

$$1 + |f^{-1}(a)| \leq k \quad \text{(i.e., } |f^{-1}(a)| \leq k-1) \quad \text{for each } a \in C \setminus \{t\} \text{ and } |f^{-1}(t)| \leq k$$

Let f be a function that satisfies this equation. Suppose we send one unit of flow from the super-source x along each of its $\|C\| - 1$ outgoing links. For each a $\in C \setminus \{t\}$, send the one unit of flow that arrives to l_a towards $r_{f(a)}$. Finally, for each $b \in C$, send the flow that arrives into r_b towards the super-sink z. This flow satisfies the capacity constraints because f satisfies the given equation.

Conversely, we say that if we have a flow of $\|C\| - 1$ then we also have a function satisfying the given equation. This is very much clear from the algorithm, if f(a) = b then there must be one unit of flow from $l_a$to r_b, and hence f satisfies the equation due to the constraints of the algorithm on the vertices.

6.4 Minimum Pathwidth

For this part, we have to first understand a lemma that sets a restriction on the width of path decomposition.

Lemma 6.6. For every tree T = (C, E), there exists a path decomposition S_1, . . . , S_r of T of minimum width w such that, for every edge $e \in E$, there is an endpoint $a \in e$ for which there exists a bag S_i with$a \in S_i$ such that $|Si| \leq w$

Proof: To prove this we simply take any arbitrary path decomposition $S_1, S_2, ...S_r$, and then for each edge say $\{a, b\}$ we find an a minimum i such that S_i has both a and b. Now if i is 1 then we simply create an S^* which is simply S_i without b then size of this new set would be at most w. Then we append this set before S_1.

However, if i is greater than 1 then one of a or b, say b is not in S_{i-1} again we take the same S^* as before and we put it between S_i and S_{i-1}. Obviously, this new sequence of S_i will also be a new path decomposition.

Now we give our algorithm,

Algorithm 8: Find a tree $T = T(P)$ of minimum pathwidth

1: Let $D = (C, A)$ be the attachment digraph of P
2: Let $S_1, \ldots, S_k$ be a path decomposition of $G(D)$ of minimum width w that satisfies the condition of Lemma 5.6
3: $f \leftarrow \varnothing$, the attachment function under construction
4: **for** $a \in C \setminus \{t\}$ **do**
5: **if** $a \in C^{\rightarrow}$ **then**
6: $f(a) \leftarrow b$, for the unique $b \in C$ with $(a, b) \in A$
7: **else If** $a \in C^{\Rightarrow}$
8: Let C_1, C_2 be two forced vertices such that $(C_1, C_2), (a, c_1), (a, c_2) \in A$ (these exist by the circumtransitivity property)
9: Since $\{C_1, C_2\}$ is an edge of the graph of forced vertices, by the condition of Lemma 5.6, there is a bag S_j with $c_i \in S_j$ and $|S_j| < w$, for some $i \in \{1, 2\}$
10: $f(a) \leftarrow c_i$
11: Make a new bag $S_a = S_j \cup \{a\}$ and place it to the right of S_j in the sequence of the path decomposition
12: **end if**
13: **end for**
14: return $T^* = T(f)$

Theorem 6.7. Algorithm 6 returns a tree $T \in T(P)$ with minimum pathwidth among trees in T(P) in polynomial time

Proof: Firstly, we claim that the output sequence is a path decomposition. Here each free vertex a becomes a leaf of the output tree and only occurs in the bag S_a. Since it is a leaf it has only a single neighbor and that is f(a). From the algorithm, it is clear that a and f(a) both are in S_a. Since a only occurs in a single bag the sequence is also correct.

We have already seen the fact that the graph of only forced vertices is a common occurrence in all trees . Also, the size of the bag s will be at most w+1 Since the algorithm identifies a graph of padwidth equal to the graph of only forced vertices this is indeed minimum.

6.5 Other Graph Types from above framework

To conclude this section we now discuss how we recognize trees of certain types.

Paths: To recognize paths some algorithms worth mentioning are by Doignon and Falmagne (1994) and Escoffier et al which run in time O(mn). In our study algorithm 4 used to find tree with minimum leaves can be used for this purpose. Algorithm 5 can also be used to find trees with max degree 2. However these require attachment diagraph which take $O(m^2n)$ time to run and hence slower.

Star: We have already seen that the singly peaked tree is a star if every voter ranks a particular candidate either first or second.This can be easily checked in O(N) time.

Caterpillar: A caterpillar graph has a padwidth of exactly 1 . Algorithm 6 can easily check this. Further, if the graph of only forced vertices is a caterpillar then the full graph is also one. Otherwise, it is not. This is because free vertices only become leaves to internal vertices in the algorithm.

Subdivision of a star: A tree is a subdivision of a star if at most one vertex has a degree of 3 or higher. We can find a subdivision of a star in T(P), should one exist, by adapting Algorithm 5 we guess the center of the subdivision of the star, and then assign suitable upper bounds on the vertex degrees by appropriately setting the capacity constraints in the flow network

7 Conclusions

Conclude in your words.

References

N. Betzler, A. Slinko, and J. Uhlmann. On the computation of fully proportional representation. In *Journal of Artificial Intelligence Research, 47(1)*, pages 475–519, 2013.

G. Demange. Single-peaked orders on a tree. In *Mathematical Social Sciences*, pages 389–396, 1982.

M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. In *W. H. Freeman and Company*, 1979.

J. Guo and R. Niedermeier. Exact algorithms and applications for tree-like weighted set cover. In *Journal of Discrete Algorithms*, pages 608–622, 2006.

R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

D. Peters, L. Yu, H. Chan, and E. Elkind. Preferences single-peaked on a tree: Multiwinner elections and structural results. In *Journal of Artificial Intelligence Research*, January 2022.

M. A. Trick. Induced subtrees of a tree and the set packing problem. In *IMA Preprint Series 377, University of Minnesota*, 1989a.