FPT:

Reference: Parametrized Algorithms

Fixed. Parameter tractible

Goal:= Refine the complexity analysis of a computational problem to find tractability within NP & beyond. So far we measure the complexity of the problem in terms of the size of the input. In real life we also have information on structure of the input. The idea is to understand which parameter of the structure is to be blamed for the combinatorial explosion. Any algorithm with running time $f(k) \cdot n^c$, for a constant c is called a fixed parameter algorithm (FPT).

In parametrized algorithm, k simply a relevant secondary measurement that encapsulate

some aspect of the input instance. eg size of solution sought after or number describing the structure of the input ...

Not all parameters yeild FPT. Eg k-colorability. There are problems that are naturally FPT eg SAT (with number of variables as parameter).

Formal definition A parametrized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed finite alphabet. For any instance (x,k) k is called the parameter.

Defn :: A parametrized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called fixed parameter tractible (FPT) if there is an algorithm A, a computible func $f: \mathbb{N} \to \mathbb{N}$, and a constant c s.t given $(x,k) \in \Sigma^* \times \mathbb{N}$, the algorithm correctly decides whether $(x,k) \in L$ in time bounded by $f(k) \cdot |x|^c$. The complexity class containing all fixed-point-tractable problems is called FPT.

A parametrized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is called slice-wise polynomial (XP) if there is an algo A & two computable functions f, g s.t given (x, k). the algorithm correctly decides if $(x,k) \in L$ in time $f(k) \cdot |x|^{g(k)}$. The complexity class containing all slice-wise poly problem is called XP.

Bounded - Search Tree

Bounded Search tree (or simply branching) is one of the simplest and most commonly used technique in parametrized complexity. Source of high running time is branching behaviour of algorithm. The idea is to keep this branching as a function of the parameter.

It is often convienient to think of a branching as "guessing" the right branch, we now show how to apply the idea of BST to vertex cover.

Theorem VERTEX COVER of size at-most k, is solvable in $O(2^k \cdot |V(G)|)$.

Proof

We construct a binary tree of height 'k' as follows.

- Label the root by (ϕ, G).
- Choose an edge $(v, w) \in E$.
- In any vertex cover V', either $v \in V'$ or $w \in V'$

So we create two children, one labelled by $(\{v\}, G \setminus \{v\})$ & the other by $(\{w\}, G \setminus \{w\})$.

Then continue in each of these newly created nodes in the same way.

→ If the graph has a vertex cover $\leq k$

→ If the graph has a vertex cover $\leq k$ then we can find it in a tree of height $\leq k$.

Shrinking the search tree

For this we will assume that the graph we consider has vertices with degree > 2.

Note that if we only have vertices with degree ≤ 2, it only consists of paths, cycles & isolated vertices. If such a graph has $> 2k$ edges, it cannot have a VC of size k. Solution of such graphs can be found in polytime.

We now describe our recursive branching algo.

1) For any vertex v, either it must be in the vertex cover or all its neighbours.

2) Vertex cover becomes trivial when the maximum degree of graph is atmost 1.

3) We first find a vertex $v \in V(G)$ of maximum degree in G.

If v is degree -1, we know how to solve it in poly time. Otherwise $|N(v)| \geq 2$. We recursively branch on 2 cases by considering.

v, $N(v)$. in the vertex cover.

In the branch where v is in the vertex cover we can delete v & reduce the param by 1. In the second branch, we add $N[v]$ to the vertex cover, delete $N[v]$ & decrease k by $|N(v)|$.

The running time of Algo is bounded by.

(Num of nodes in search tree) $\times$ (time for each node)

Clearly time taken in each node is $n^{O(1)}$.
$\Rightarrow$ The algo takes $T(k) \cdot n^{O(1)}$, where $T(k)$ is the size of tree.

The size of the tree can be obtained by solving the following recurrence relation for k.

$$T(i) = \begin{cases} T(i-1) + T(i-2) & \text{if } i \geq 2. \\ 1 & \text{otherwise.} \end{cases}$$

$$\tau(k) = T(k). \leq 1.6181^k.$$

Now if we only consider vertices of degree atleast 3, each time we branch, we get

$$T(k) = \begin{cases} T(k-1) + T(k-3) & k \geq 3 \\ 1 & \text{otherwise}). \end{cases}$$

$$\Rightarrow \tau(k) = T(k) = 1.4656^k.$$

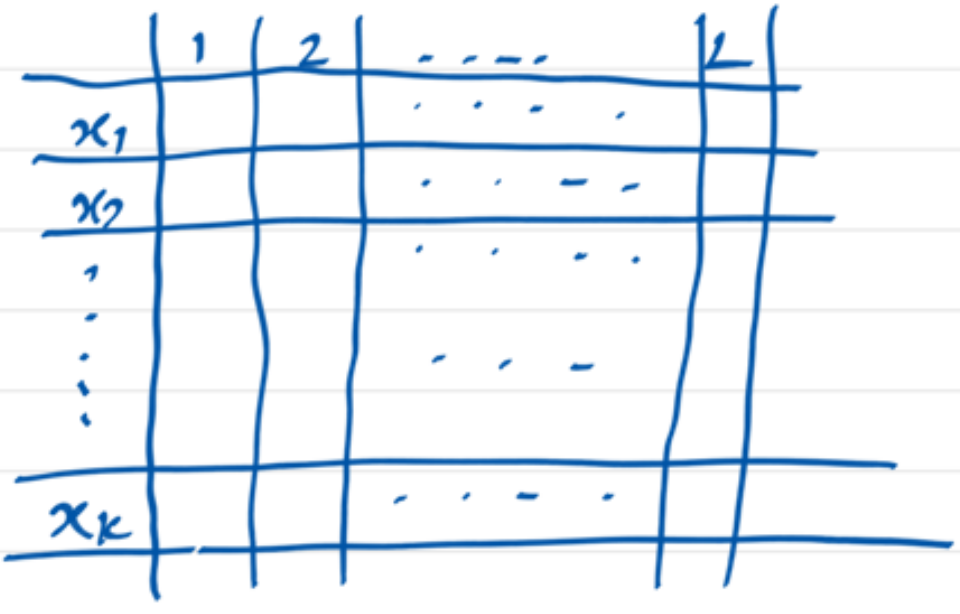
We will see how to solve another related problem called the closest string problem.

CLOSEST STRING

Problem We are given k strings $x_1, \ldots, x_k$ each string over an alphabet Σ and of length L and an integer d. The question is whether there is a string y of length L over Σ s.t $d_H(y, x_i) \leq d$.

$d_H(y)$ → is the hamming distance b/w x & y. i.e. the number of positions where x & y differ.

Given a set of k strings, each of length L we can think of these strings as a $k \times L$ character matrix.



↑ columns.

We call a column bad if it contains atleast two different symbols from alphabet Σ & good otherwise. For any good column, we can set value in 'y' (& center string) to be that value. ie for any good column 'j', we let

$$y[j] = x_1[j] = \cdots = x_k[j].$$

Rule

(1) Delete all good columns.

<u>lemma</u> for every yes instance, of the CLOSEST string, the corresponding $k \times \ell$ matrix contains at most $k \cdot d$ bad columns.

<u>Proof</u> Let y be a center string. For every bad column j, there is a string x_{i_j} s.t $x_{i_j}[j] \neq y[j]$. Since y differs by at most d positions in every string, there are atmost $k \cdot d$ bad columns.

Rule (2) If there are more than kd bad columns then conclude that we are dealing with a no-instance.

Thm:- CLOSEST String can be solved in time $O(kd \cdot (d+1)^d)$.

Proof:- First we apply Rule (1) & (2). Unless we have resolved the instance already, we are left with k-strings $x_1, x_2, \ldots, x_k$, each of length $L \leq kd$. The algorithm is as follows.

(1) we set $z = x_1$ & $l = d$ initially.

(2) If z is the center string we return it.

(3) If $l = 0$, we return am as NO.

(4) Otherwise [$L > 0$], there is atleast 1 string x_i s.t $d_H(x_i, z) > d$. Let P be some arbitrary set of $d+1$ positions of x_i that differ from z.

if y is a center string then it can differ in atmost d-positions w.r.t x_i. So there is atleast one position $p \in \mathbb{P}$ s.t $y[p] = x_i[p]$. Now we branch into $d+1$ branches, corresp to each position in $\mathbb{P}$ & in such a branch, we set $z[p] = x_i[p]$, $p \in \mathbb{P}$. We also reduce the value of l. We now repeat the process for $(z_p, l-1)$. The gendral idea here is z is initially set to x_1 and hence differs from y (a center string if it exists) in atmost d-position. The algorithm attempts to toggle bits of z, d times to take it closer to y.

The tree we construct branches as $(d+1)$ & has height d hence a size $O((d+1)^d)$. We need $O(kd)$ steps to check if z is a center string & to find x_i.

Iterated Compression

Goal := Introduce Iterated compression as an FPT-technique.

The main technique used in Iterated compression is to employ a so-called compression routine. A compression routine is an algorithm that given a problem instance and a corresponding solution, either calculates a smaller solution or provides a certificate for "no-solution". The main point of the iterative compression technique is that if the compression routine runs in FPT time then so does the whole algorithm

We will next look at Iterative Compression for the parametrized vertex cover problem.

Theorem: Parametrized Vertex cover problem can be solved in time $\mathcal{O}(2^k \cdot n^{\mathcal{O}(1)})$ using iterated compression technique.

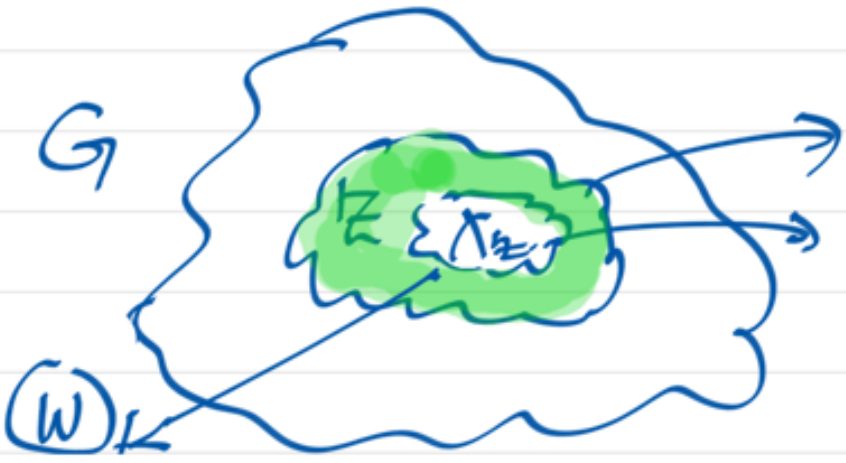
Proof: The idea is to first come up with a compression algorithm.

Algorithm - Compress - Vertex cover

Input : Graph G, Vertex cover Z, Param - k

output : compressed vertex cover X of size $\leq k$ or False if none exists.

Idea: We branch in all possible ways an optimal solution X can intersect with Z. Let $X_Z \subseteq Z$ be one such guess.



We are searching for X → vertex cover of G s.t $X \cap Z = X_Z$.

Let $W = Z \setminus X_Z$. If $G(W)$ contains any edge then the guess X_Z is wrong. Otherwise the vertex cover X has to include X_Z and the neighborhood of W. If $|X_Z \cup N(W)| \leq k$ then we let $X = X_Z \cup N(W)$ & return the same. Otherwise we conclude that there is no k-vertex cover. Such an algorithm runs in time $2^{|Z|} \cdot n^{O(1)}$.

Iterative Compression algorithm for Vertex Cover

Let G be the given graph & k be the given parameter. We assume an arbitrary ordering on the vertices of G say $\{v_1, v_2, \ldots, v_n\}$. Let G_i be the subgraph induced by first i-vertices. We proceed iteratively and show how to build vertex cover for graphs $G_1, \ldots, G_n$. The vertex cover $X_1, \ldots, X_n$ so built will all be of size $\leq k$.

Initially $X_1 = v_1$. clearly it is vertex cover of G_1. We also let $Z_1 = X_1$.

For case where $i \geq 1$, we proceed as follows.

Let $Z_i = X_{i-1} \cup \{v_i\}$. Clearly if X_{i-1} is a vertex cover of G_{i-1}, then Z_i is vertex cov of G_i. We then call the compression algo

$X_i = \text{COMPRESS-VC}(G_i, Z_i)$.

if a vertex cover $\leq k$ exists for Z_i, then X_i is one such Vertex-Cover.

Tree width :-

The tree width of a graph is one of the most frequently used technique in FPT. algos. Tree-width measures how well the structure of a graph can be captured by a tree like structural decomposition.

Consider the weighted independant set on Trees. We are given a tree where each vertex is assigned a non-negative weight. The task is to find the max wt of an independant set in T.

Solution using dynamic program

- Let r be root of the tree T.
- for every v of T, let T_v be the subtree of T rooted at v. For each v, we define the following.
  - Let A[v] → max possible weight of an independant set in T_v.
  - Let B[v] → Max possible weight of an independant set not including v.

Goal is to find $A[r]$. We do this recursively in bottom-up manner.

For leaves, $A[v] = w(v)$ & $B[v] = 0$. Otherwise let $v_1 \ldots v_q$ be the children of v.

$$B[v] = \sum_{i=1}^{q} A[v_i].$$

$$A[v] = \text{Max}\{ B[v], w[v] + \sum_{i=1}^{q} B[v_i] \}.$$

We will now try to lift this idea to general graphs.

Path and Tree-decomposition

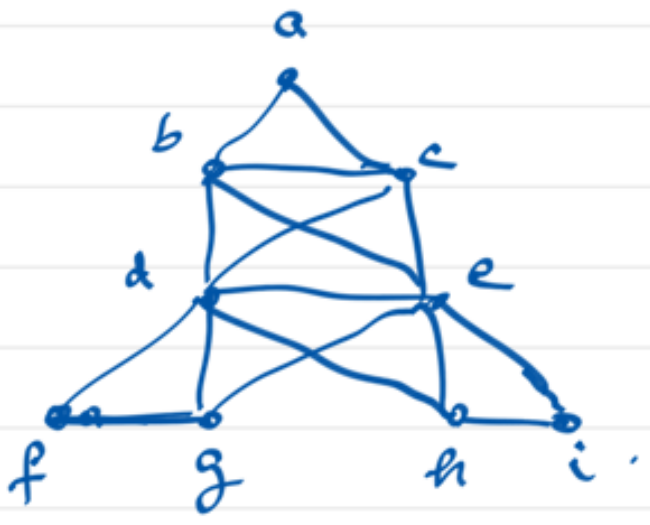
Path decomposition: A path decomposition of a graph G is a sequence $P = (X_1, \ldots X_r)$ of bags where $X_i \subseteq V(G)$ for each $i \in \{1, \ldots, r\}$, the following conditions hold.

(1) $\bigcup_{i=1}^{r} X_i = V(G)$.

(2) $\forall (u,v) \in E(G)$, $\exists i$: $u, v \in X_i$.

(3) $\forall\, u \in V(G)$, if $u \in X_i \cap X_k$ for some $i \le k$, then $u \in X_j$ for each $j : i \le j \le k$

eg.



Path decomposition

The width of a path decomposition $(X_1, \ldots, X_r)$ is $\max_{1 \le i \le r} |X_i| - 1$. The path width of a graph G, denoted $PW(G)$ is the min possible width of a path decomposition of G.

Some definitions

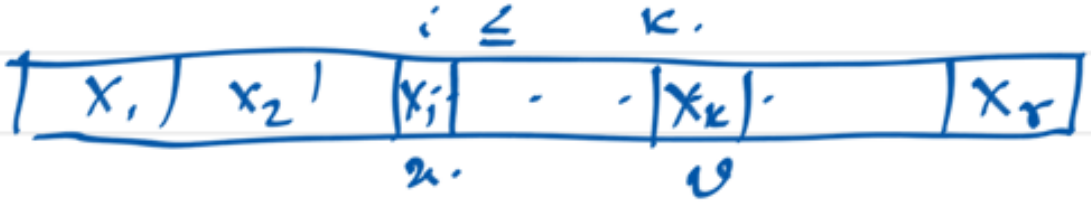
The most crucial property of a path decompositions is that they define a sequence of seperators in the graph.

We say (A, B) is a seperation of a graph G if $A \cup B = V(G)$, $A \setminus B$ & $B \setminus A$ have no edges btw them. $A \cap B$ is called the seperator of this seperation, $|A \cap B|$ is called the order of the seperation.

For any subset $A \subseteq V(G)$, we define border of A denoted $\partial(A)$, as the set of those vertices of A that have a neighbour in $V(G) \setminus A$.

Lemma: Let $(X_1, \ldots, X_r)$ be a path decomposition of a graph G. Then for every $j \in \{1, \ldots, r-1\}$, it holds that $\partial(\bigcup_{i=1}^{j} X_i) \subseteq X_j \cap X_{j+1}$. In other words, $(\bigcup_{i=1}^{j} X_i, \bigcup_{i=j+1}^{r} X_i)$ is a separation of G, with separator $X_j \cap X_{j+1}$.

Proof: Let us fix j & let $(A, B) = (\bigcup_{i=1}^{j} X_i, \bigcup_{i=j+1}^{r} X_i)$. We will first show that $\partial(\bigcup_{i=1}^{j} X_i) = \partial(A) \subseteq X_j \cap X_{j+1}$. Targeting a contradiction suppose there is a $u \in \partial(A)$ such that $u \notin X_j \cap X_{j+1}$. This means that $\exists (u, v)$ edge s.t $u \in A$ & $v \notin A$. But $u \notin X_j \cap X_{j+1}$. Now let i be the largest index such that $u \in X_i$ & k be the smallest index such that $v \in X_k$.



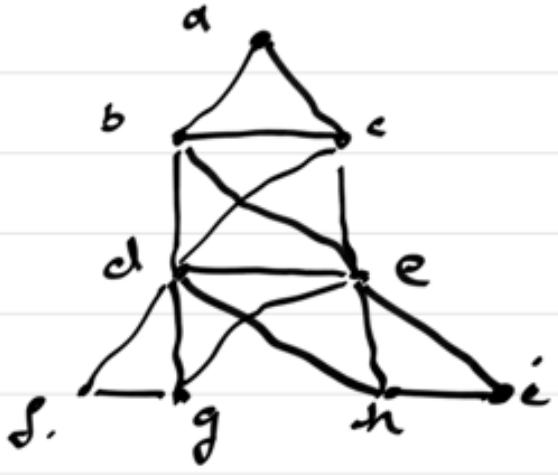
Since $u \in A$, $u \notin X_j \cap X_{j+1} \Rightarrow i \leq j$ similarly $k \geq j+1$. Hence $i < k$. But notice that there should be an 'l' s.t $u, v \in X_l$. $\Rightarrow l \leq i < k \leq l$ a contradiction.

Canonical path decomposition

We say a path decomposition $P = (X_1, \ldots, X_r)$ of a graph G is nice if:

- $X_1 = X_r = \emptyset$.
- $\forall i \in [1 .. r-1]$, there is either a vertex $v \notin X_i$ s.t. $X_{i+1} = X_i \cup \{v\}$ or there is a vertex $w \in X_i$ s.t. $X_{i+1} = X_i \setminus \{w\}$.

C_9 :=



Notations :-

Bags of the form $X_{i+1} = X_i \cup \{v\}$ shall be called introduce bags. Similarly, bags of the form $X_{i+1} = X_i \setminus \{v\}$ shall be called forget bag (or node)

Every path decomposition can be turned into a nice path decomposition.

Lemma :- If a graph G admits a path decomp of width at most p, then it also admits a nice path decomposition of width at-most p.

Tree - decomposition :- A tree decomposition is a generalization of path - decomposition.

Formally, a tree decomposition of a graph G is a pair $T_d = (T, \{X_t\}_{t \in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a bag such that following conditions hold.

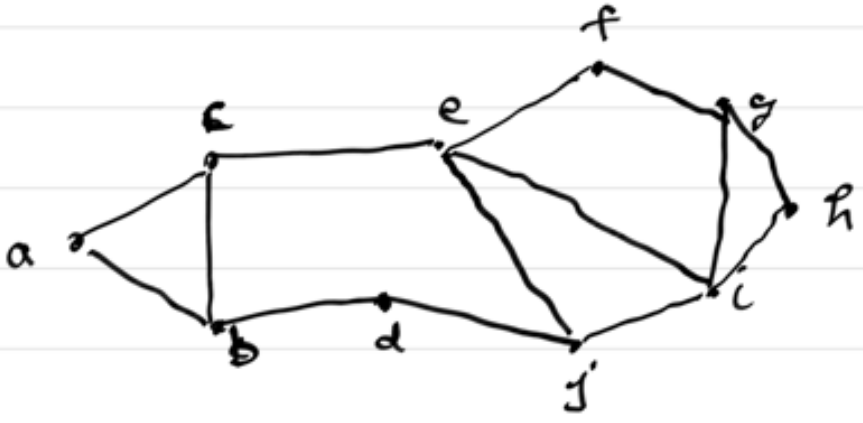
1) $\bigcup_{t \in V(T)} X_T = V(G)$

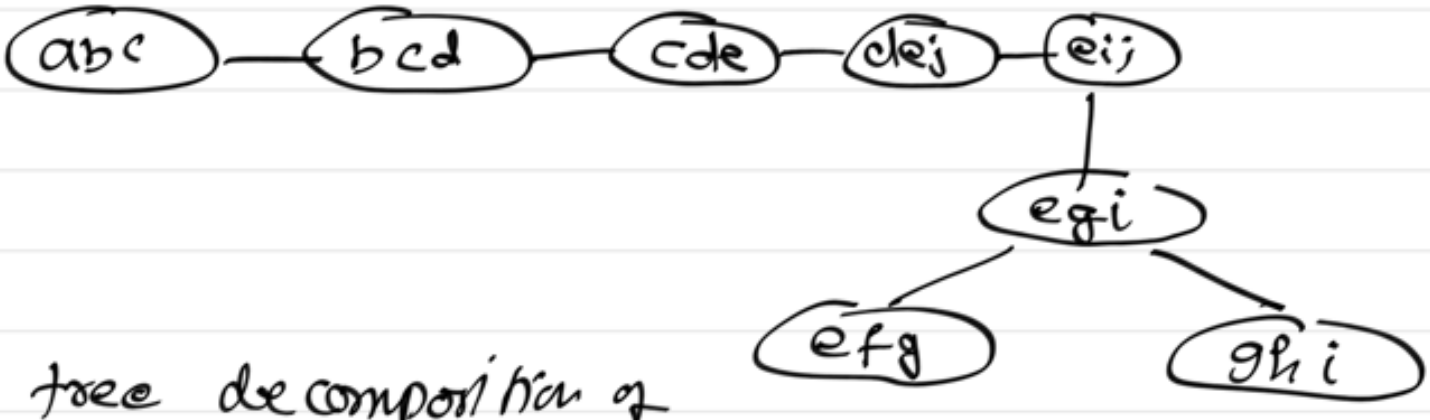
(2) $\forall (u,v) \in E(G), \exists t \in T$ s.t $u, v \in X_t$.

(3) $\forall x \in V(G)$, the set $T_x = \{t \in V(T) / x \in X_t\}$. induces a connected sub-tree of T.

The width of tree decomposition $\mathcal{D} = (T, \{X_t\}_{t \in T})$ $= \max_{t \in T} (|X_t| - 1)$, i.e maximum size of its bags - 1.

Example





tree decomposition of width 2.

Tree width is the min width of all possible decomposition

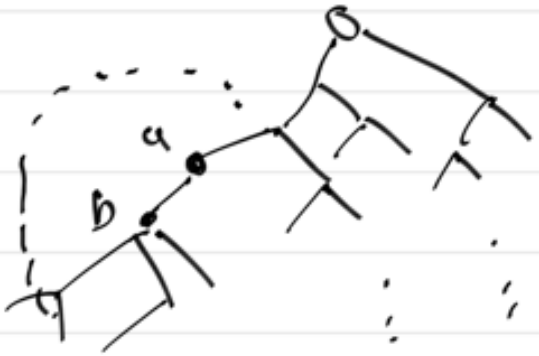
Similar to path decomposition, tree decomposition induces a separation property on the graph. This is captured in the following lemma.

Lemma := Let $(T, \{X_t\}_{t \in V(T)})$ be a tree-decomposition of a graph G and let (a,b) be any edge in T. The forest $T-(a,b)$ obtained from T by deleting the edge (a,b) consists of two components T_a & T_b. Let

$A = \bigcup_{t \in V(T_a)} X_t$ & $B = \bigcup_{t \in V(T_b)} X_t$. Then,

$\partial(A), \partial(B) \subseteq X_a \cap X_b$. Equivalently (A,B) is a separation of G with separator $X_a \cap X_b$.

Proof := Idea is similar to the case of path decomposition.



Nice Tree Decomposition :-

We will say that a rooted tree decomposition $(T, \{X_t\}_{t \in V(T)})$ is nice if the following conditions hold.

- $X_r = \phi$ & $X_\ell = \phi$, for every leaf ℓ in T.
- Every non leaf node of T is one of the following types.

1) Introduce Node : a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$. for some $v \notin X_{t'}$.

2) Forget Node: A node t with exactly one child t' such that $X_t = X_{t'} \setminus \{w\}$. for some vertex $w \in X_{t'}$.

3) Join Node: a node t with two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$.

Lemma If a graph G admits a tree decomposition of width at most k, then it also admits a nice tree decomposition of width at most k. Moreover given a tree decomposition $\mathcal{T} = (T, (X_t)_{t \in V(T)})$ of G of width at most k, one can compute a nice tree decomposition of width at most k that has at most $O(k \cdot |V(G)|)$ nodes in polytime.

Dynamic programming on graphs of bounded tree-width.

Weighted Independent Set

Let G be the given graph and let w be the weight function. Let

$$\mathcal{T} = (T, (X_t)_{t \in V(T)})$$ be a tree

decomposition, we will assume that it is a nice tree decomposition. For any node $t \in V(T)$, we let V_t to be the union of all bags present in the subtree rooted in t.

For any node $t \in V(T)$, we know from the previous lemma that $\partial(V_t) \subseteq X_t$. We will show how to extend weighted Independent set Algo that we saw earlier for trees to graphs.

Let I_1, I_2 be any two independent set of G such that $I_1 \cap X_t = I_2 \cap X_t$ for some $t \in V(T)$. Suppose $w(I_1 \cap V_t) > w(I_2 \cap V_t)$. We can obtain I_2' from I_2 by replacing $I_2 \cap V_t$ by $I_1 \cap V_t$. Since X_t seperates $V_t \setminus X_t$ from rest of the graph I_2' is still an independent set. Further $w(I_1 \cap V_t) > w(I_2 \cap V_t) \Rightarrow w(I_2') > w(I_2)$.

We will now attempt to mimic the bottom-up dynamic programming that we performed on trees.

For every node t & every $S \subseteq X$, let

$c[t, S]$ = Max possible weight of set $\hat{S}$ s.t $S \subseteq \hat{S} \subseteq V_t$, $\hat{S} \cap V_t = S$ & $\hat{S}$ is indep-set.

If no such $\hat{S}$ exists, we set $c[t, S] = -\infty$.
$c[r, \phi]$ is the value that we are looking for.

We will now show how to compute $c[\cdot, \cdot]$.

Leaf node := If t is a leaf node, then we have only one value $c[t, \phi] = 0$.

Introduce Node := Suppose t is an introduce node with only child t' s.t $X_t = X_{t'} \cup \{v\}$. Let S be any subset of X_t. If S is not an independent set, we let $c[t, S] = -\infty$. Otherwise, we let

$$c[t, S] = \begin{cases} c[t', S] & \text{if } v \notin S. \\ c[t', S \setminus \{v\}] + w(v) & \text{otherwise} \end{cases}$$

Forget Node := Suppose t is a forget node with only child t' s.t $X_t = X_{t'} \setminus \{v\}$. Let S be any subset of X_t. If S is not an Indep set, we let $c[t, S] = -\infty$ otherwise

$$c[t, S] = \max\{c[t', S], c[t', S \cup \{v\}]\}.$$

Join node :- Suppose t is a join node with two children t_1 & t_2 s.t $X_t = X_{t_1} = X_{t_2}$. For any indep subset $S \subseteq X_t$, we at

$$C[t, S] = C[t_1, S] + C[t_2, S] - w(S).$$

We are working on a tree decomposition of width at-most k! i.e. $|X_t| \leq k+1 \; \forall t \in V(T)$. Thus at any node we compute at most $2^{|X_t|} \leq 2^{k+1}$ values for $V[t, S]$. Any computation in each node can be done in poly(n), resulting in

Thm Let G be a n-vertex graph, with weights on its vertices. Let $T = (T, (X_t)_{t \in T})$ be its tree decomp. with width at most k. Then the weighted Ind set problem can be solved in $2^k \cdot \text{poly}(n)$.

Tree width & Monadic second order Logic

Syntax & Semantics of MSO$_1$ Logic

MSO Logic

→ Let $x, y, z, x_1, x_2 \ldots$ be variables that denote vertices

→ Let $X, X_1, X_2, X_3 \ldots$ be variables that denote subsets of vertices.

→ Let $E(x_1, x_2)$ denote the fact that $x_1 \to x_2$ is an edge.

Monadic second order formulas are those that can be constructed using the following.

- $x \in X$
- $x_1 = x_2$
- $E(x_1, x_2)$
- $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg \phi_1$
- $\exists x\, \phi$, $\forall x\, \phi$.
- $\exists X\, \phi$, $\forall X\, \phi$.

Example 3- colorability.

3 Colorability $= \exists X_1, X_2, X_3 .$ Partition (X_1, X_2, X_3)
$\wedge$ indp $(X_1) \wedge$ indp $(X_2) \wedge$ indp (X_3).

Partition $(X_1, X_2, X_3) = \forall v \; (v \in X_1 \wedge v \notin X_2 \wedge v \notin X_3) \vee$
$(v \notin X_1 \wedge v \in X_2 \wedge v \notin X_3) \vee$
$(v \notin X_1 \wedge v \notin X_2 \wedge v \in X_3)$

Indp $(X) = \forall u, v, \; (u, v \in X) \Rightarrow (\neg E(u, v))$.

MSO_2 formulas are those in which we can quantify over Edges as well.

Courcelle's theorem

Thm Assume that φ is a formula of MSO_2 and G is an n-vertex graph. Suppose that a tree decomposition of G of width t is provided. Then there is an algorithm that verifies whether φ is satisfied in G in time $f(\|\varphi\|, t) \cdot n$, for some computable function f.

In order to solve vertex cover problem, given a graph G and integer k, we can quantify existentially over k vertex variables and write an MSO formula. However such a formula is linearly dependent on k, Applying Courcelle's theorem gives us $f(k,t) \cdot n$ algorithm. The following theorem comes to our rescue.

Thm Let φ be an MSO formula with p free monadic variables $X_1, \ldots X_p$. & let α be an affine function over $x_1, \ldots x_p$. Assume we are given a n-vertex graph and a tree decomposition of width t. Then there is an algorithm that in $f(\|\varphi\|, t) \cdot n$ finds the min & max values of $\alpha(|X_1|, |X_2|, \ldots, |X_p|)$ for sets $X_1, \ldots X_p$ that satisfies φ.

[optimized courcelles theorem]

Computing tree width :=

Finding a small tree decomposition of a given graph is NP-hard problem. what about Parameterized tree-width?

Thm:= There exists an algorithm that given a n-vertex graph G & parameter k, runs in time $k^{O(k^3)} \cdot n$ and either constructs a tree decomposition of G of width at-most k or concludes that $tw(G) > k$.

Kernelization :-

Goal :- To introduce Kernelization. Kernelization is a systematic approach to study polynomial time pre-processing algorithms. The idea of this method is to reduce the given problem instance to an equivalent smaller sized instance in time polynomial in the input size.

Defn (Kernelization, Kernel). A Kernelization Algo. or simply a Kernel, for a parameter problem Q is an algorithm A that, given an Instance (I, k) of Q and returns an equivalent instance (I', k') of Q. Moreover $size_A(k) \leq g(k)$

$$size_A(k) = \sup\{|I'| + k' : (I', k') = A(I, k), I \in \Sigma^*\}.$$

if $g(k)$ is bounded polynomially, we say Q admits polynomial kernel.

Kernelization + decision procedure $\Rightarrow$ FPT

but what about the otherdirection?

Lemma If a parametrized problem Q is FPT then it admits a kernelization algorithm.

Proof: Since Q is FPT, there is an algo $\mathcal{A}$ deciding if $(I,k) \in Q$ in time $f(k) \cdot |I|^c$ for some computable function f and a constant c. We obtain a kernelization algorithm for Q as follows.

1) Given (I,k), Run $\mathcal{A}$ for time $|I|^{c+1}$ & if it terminates return the answer.

2) If it does not terminate within $|I|^{c+1}$ steps, return (I,k) itself.

why does this work. we have

$$f(k) \cdot |I|^c > |I|^{c+1}$$

$$\Rightarrow \quad f(k) > |I|.$$

Some Simple Kernels :-

Vertex Cover :- In Vertex Cover problem, we are given a graph G and a positive integer k as input. The objective is to check whether there exists a vertex cover of size atmost k.

We employ the following reduction rules to obtain the kernel.

(Red-1) If G contains an isolated vertex v, delete v from G. The new instance is $(G-v, k)$

(Isolated edges do not cover any edges).

(Red-2): If there is a vertex v of degree at least $k+1$, then delete v from G and decrement k by 1.

(If G contains v of degree $> k$, then v must be in every vertex cover of size $\leq k$)

Observe that if graph has max deg d then a vertex cover of size k can cover at most $k \cdot d$ edges.

lemma:= If (G, k) is a yes-instance and none of the reduction rules Red-1, Red-2 is applicable then $|V(G)| \leq k^2 + k$ & $|E(G)| \leq k^2$.

Proof:=

① There are no isolated edges.

② Yes instance $\Rightarrow$ vertex cover of @-most k further deg is $\leq k$. $\Rightarrow |E(G)| \leq k^2$

③. Every vertex is either in vertex cover or end point of an edge $\Rightarrow |V(G)| \leq k^2 + k$.

Now let (G, k) be an input instance s.t Red-1, Red-2 are not applicable. Then if G has $> k^2 + k$ vertices or $> k^2$ edges then we are dealing with a no-instance.

Thm:= Vertex Cover admits a kernel with $O(k^2)$ vertices & $O(k^2)$ edges.

Crown Decomposition

Crown decomposition is a general kernelization technique that can be used to obtain kernels for many problems.

For disjoint vertex subset V & W of graph G, a matching M is called a matching of V into W if every edge of M connects a vertex of V & a vertex of W. further every vertex of V is an endpoint of some edge of M. (we also say M saturates V).

Defn (Crown decomposition). A crown decomposition of a graph G is a partioning of $V(G)$ into three parts C, H and R such that

(1) C is non-empty
(2) C is an independent set
(3) There are no edges b/w C & R.
(4) Let E' be the set of edges b/w C & H. Then E' contains a matching of H into C

We will use the following results for proving Crown lemma.

Thm (Konig): In every undirected bi-partite graph the size of maximal matching = size of min vertex-cover.

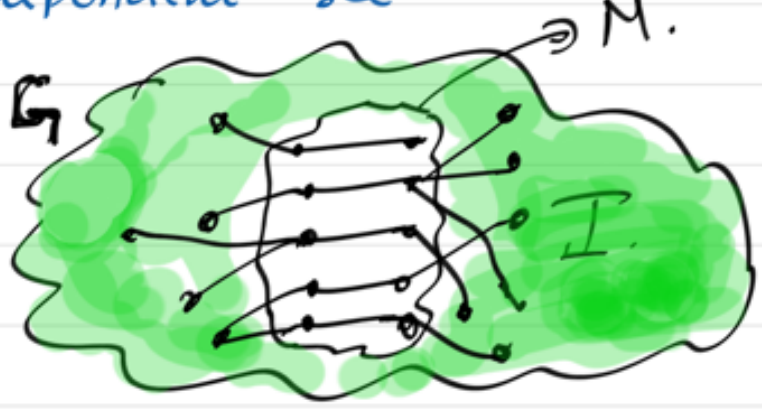
Thm (Halls):= Let G be an undirected bipartite graph with bi-partition (V_1, V_2). The graph G has a matching saturating V_1 iff $\forall X \subseteq V_1$, we have $|N(X)| \geq |X|$.

Thm: (Hopcroft-Karp). Let G be an undirected bipartite graph with bipartition V_1 & V_2 on n vertices & m-edges. Then we can find a maximal matching & min vertex cover in $O(m\sqrt{n})$.

Crown lemma: Let G be a graph w/o isolated vertices and with at least $3k+1$ vertices. There is a poly time algo that either

- finds a matching of size $k+1$
- finds a crown decomposition.

Proof: We first find an inclusion-maximal matching M in G. If size of M is $k+1$, we are done. We hence assume $|M| \leq k$. Let V_M be end points of M. Clearly $|V_M| \leq 2k$. Since M is Maximal matching $I = V(G) \setminus V_M$ is an independent set.



Now consider the bipartite graph G_{I,V_M} formed by edges of G between V_M & I.

We compute the min sized vertex cover X and max sized matching M' of the bipartite graph G_{I,V_M} in poly time (Kong-Hopcroft).

we assume $M' \leq k$ otherwise we are done.
By Konig's theorem $|X| = |M'| \Rightarrow |X| \leq k$.

lemma :- $X \cap V_M \neq \phi$.

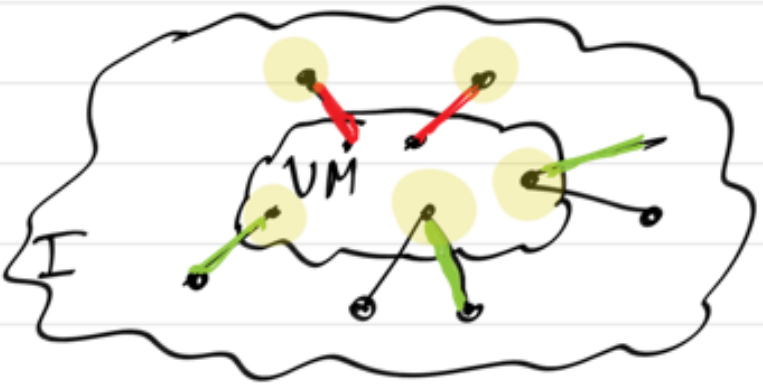
Proof :- Suppose $X \cap V_M = \phi$ then $X \subseteq I$.
In this case $\underline{X = I}$, why. Let $w \in I \setminus X$. Since G_I has no isolated vertices $\exists (w, z)$ an edge in G_{I, V_M}. since G_{I, V_M} is bipartite, $z \in V_M$. However X is a vertex cover $\Rightarrow w \in X$ a contradiction.

Now $|I| = |X| \leq k$. we know $|V_M| \geq 2k$
$\Rightarrow |I| + |V_M| \leq k + 2k = 3k$. a contradiction $\square$.

Hence we assume $X \cap V_M \neq \phi$. We obtain a crown decomposition as follows.

(1) Since $|X| = |M'|$, every edge of the matching M' has exactly one endpoint in X.

(2) Let M^* denote subset of M' s.t every edge from M^* has one end point in $X \cap V_M$. Let V_{M^*} denote set of endpoints in M^*



— ⇒ M^*

= ⇒ M'

● ⇒ X.

we define

- $H = X \cap V_M = X \cap V_{M^*}$.
- $C = V_{M^*} \cap I$
- $R = V(G) \setminus C \cup H = V(G) \setminus V_{M^*}$.

why is this a Gallai decomposition.

- Clearly C is an Independent set since $C \subseteq I$.
- By construction M^* is a matching of H into C.
- H seperates C & R since X is a vertex cover of G_{I, V_M}.

Application to vertex cover

Let the VC instance be (G, k). we will assume it has no isolated vertices. If $|V(G)| > 3k$, we can apply Crown Comma to obtain C, H, R. or a matching $\geq k+1$ in former case, the reduced instance is

$$(G - H, k - |H|),$$

why for every vertex-cover X, it contains atleast $|H|$ vertices of $H \cup C$ to cover edges of matching M (of H into C).

Thm: Vertex Cover admits a kernel with at most $3k$ vertices.

Lower Bound for Kernelization

GOAL:- Study the conditions and assumptions under which existence of a polynomial kernel for a particular FPT problem can be refuted !!!

The best known answer to the question of "whether we can refute the existence of a polynomial kernel for a particular FPT problem" comes via the framework of compositionality.

Compositionality :=

General idea is as follows. Consider the "Longest Path Problem" where given a graph G, we have to find a simple path of size k. Assume that this problem admits a kernelization algo that reduces no. of vertices to k^3. Suppose some one gave us k^2 instances of longest path problem (G_1, k) .... (G_{k^2}, k). Let H be $\bigcup_{i=1}^{k^2} G_i$ i.e. disjoint union of all graphs. The longest path problem on (H, k) = boolean OR of the answers to $(G_1, k) \vee \ldots\ldots (G_{k^2}, k)$.

But by kernelization $(M, k) \xrightarrow{\text{kernel}} (M', k')$. where $|M'| \leq k^3$ & $k' \leq k^3$. But this $\Rightarrow$ loss of information.

DISTILLATION :=

Defn := We say that a language $L \in$ co-NP/poly if there is a TM M & $(\alpha_n)_{n=0,1,2,\ldots}$ called advice string s.t.

- Machine M when given input x of length n has access to string α_n & has to decide if $x \in L$ in co-NP time.
  - if $x \in L$, the machine should accept x in all its run & if $x \notin L$ then reject in atleast one execution.
  - $|\alpha(n)| = \text{poly}(n)$.

Defn := Let $L, R \subseteq \Sigma^*$ be two languages. An OR-distillation of L into R is an algorithm that given a sequence $x_1, \ldots, x_t$ runs in time $\text{poly}(\sum_{i=1}^{t} |x_i|)$, outputs $y \in \Sigma^*$ s.t

- $|y| \leq \text{poly}(\text{Max}(x_1, \ldots, x_t))$
- $y \in R$ iff $\exists i$ s.t $x_i \in L$.

Theorem Let $L, R \subseteq \Sigma^*$ be two languages; if $\exists$ an OR-distillation of L into R then $L \in$ co-NP/poly.

Consequence of the theorem.

Corollary If an NP-hard lang $L \subseteq \Sigma^*$ admits an OR distillation into some language $R \subseteq \Sigma^*$ then $NP \subseteq$ co-NP/poly.

Proof: for any $L' \in NP$, $L' \Rightarrow L$ since L is NP-hard. Now $L \in$ co-NP/poly by above thm.

Proof: W.log, let $\Sigma = \{0,1\}$. Let A be the OR-distillation algo. Let $p(\cdot)$ be the poly. upper bound. We will assume $p(\cdot)$ is non dec.

By defn. A runs on sequence of strings of length n & outputs a string of length k ($k \leq p(n)$). Thus the algorithm maps the set $D = (\Sigma^{\leq n})^t$ of input strings into the set $\Sigma^{\leq k}$, for some t. We will assume $t = k+1$.

Let $A = L \cap \Sigma^{\leq n}$ & $\bar{A} = \Sigma^{\leq n} \setminus L$.
similarly Let $B = R \cap \Sigma^{\leq k}$ & $\bar{B} = \Sigma^{\leq k} \setminus R$.

By assumption, $\mathcal{A}'$ is an OR-distillation & hence maps $(\bar{A})^E$ into $\bar{B}$

For any string $x \in \Sigma^{\leq n}$ & $y \in \Sigma^{\leq k}$, we say that x is covered by y if $\exists (x_1, \ldots x_t)$ s.t
(a) $x = x_i$ for some i.
(b) $\mathcal{A}'(x_1 \ldots x_t) = y$.

We lift this to set X & Y. we say X is covered by Y if $\forall x \in X$, $\exists y \in Y$ s.t y covers x.

Claim: There exists a set $Y \subseteq \bar{B}$ s.t
(a) $|Y| \leq n+1$
(b) Y covers $\bar{A}$.

Suppose the claim do true, we let α_n to be the encoding of Y. $|Y| \leq n+1$ & each $y \in Y$ is at most poly(n) in size.

Now given x of length n, the required algo guesses $(x_1 \ldots x_t)$ and computes $\mathcal{A}(x_1 \ldots, x_t)$.

It then checks whether the result is contained in the advice string α_n. If indeed $x \notin L$, then at least for one guess, we will compute a string belonging to Y. This certifies $x \notin L$. □.

Claim: There exists a set $Y \subseteq \bar{B}$ s.t

(a) $|Y| \leq n+1$

(b) Y covers $\bar{A}$.

Proof:- Idea is to construct $\{y_1, y_2, \ldots\} \in \bar{B}$ till the set covers $\bar{A}$. Let S_i be the set of strings not covered by Y_i. $S_0 = \bar{A}$. Inductively the following property is maintained. $|S_i| \leq \frac{|\bar{A}|}{2^i}$.

Recall A maps $(S_{i-1})^t \to \bar{B}$.

Now $|\bar{B}| \leq |\Sigma^{\leq k}| < 2^{k+1}$. Hence by Pigeon hole principle, $\exists y \in \bar{B}$ such that

$$\left| A^{-1}(y) \cap (S_{i-1})^t \right| \geq \frac{S_{i-1}^t}{2^{k+1}} = \left(\frac{S_{i-1}}{2}\right)^t.$$

we set $y_i = y$. Observe that every string

from $(A^{-1}(y) \cap (S_{i-1})^t)$ is covered by y.

clearly $A^{-1}(y) \cap S_{i-1}^t \subseteq (S_{i-1} \setminus S_i)^t$.

$$\Rightarrow \left(\frac{S_{i-1}}{2}\right)^t \leq |A^{-1}(y) \cap S_{i-1}^t| \leq |(S_{i-1} \setminus S_i)^t| = |S_{i-1} \setminus S_i|^t.$$

This implies that

$$(S_{i-1} \setminus S_i) \geq \left(\frac{S_{i-1}}{2}\right).$$

$$\Rightarrow S_i \leq \frac{|S_{i-1}|}{2}. \quad \Rightarrow S_i \leq \frac{|A|}{2^i}.$$

$\Rightarrow$ the construction will terminate after at most $n+1$ steps. □.

Composition :-

Defn: An equivalence relation R on set Σ^* is called a polynomial $\equiv$ relation if the following conditions are satisfied.

1) $\exists$ an algo that given $x, y \in \Sigma^*$. resolves whether $x \equiv_R y$ in poly $(|x| + |y|)$ time

2) The relation R restricted to the set $\Sigma^{=n}$ has at-most $\underbrace{p(n)}_{\text{Poly.}}$ equivalence class.

Defn: Let $L \subseteq \Sigma^*$ be a language & $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized lang. We say L - cross-composes into Q if $\exists$ a poly $\equiv$ relation R. & an algo A (called cross composition) satisfying the following condition.

A takes $x_1, \ldots, x_t \in \Sigma^*$ that are equiv w.r.t R, runs in time $\sum_{i=1}^{t} |x_i|$ & outputs $(y, k) \in \Sigma^* \times \mathbb{N}$. s.t

(a) $k \leq$ Poly $(\text{Max}\{x_1, \ldots x_t\} + \log(t))$

(b) $(y, k) \in Q$ iff $\exists i$ s.t $x_i \in L$.

Defn:= A poly-compression of a parametrized language $Q \subseteq \Sigma^* \times \mathbb{N}$ in $R \subseteq \Sigma^*$ is an algo that takes as input $(x,k) \in \Sigma^* \times \mathbb{N}$, comp in time poly $(|x| + k)$ & returns y s.t.

(1) $y \leq$ poly(k).
(2) $y \in R$ iff $(x,k) \in Q$.

Thm:= Assume that an NP-hard lang L cross-composes into a parametrized lang Q. Then Q does not admit a poly compression unless $NP \subseteq$ co-NP/poly.

Proof:= Let A be cross-composition of L into Q. Let $\equiv$ be the poly equivalence relation used by A.

Let us assume that Q admits a poly compression C into some lang R,
Let $OR(R)$ be a language consisting of strings of the form $d_1 \# d_2 \# \ldots \# d_q$ s.t $\exists i$ s.t $d_i \in R$

we will under the assumptions construct an OR-distillation of L onto $OR(R)$.

This would imply $NP \subseteq coNP/poly$.

Let $x_1 \ldots x_t$ be the sequence of inpt string & $n = \max\{x_1, \ldots x_t\}$.

Examine the string pairwise & remove duplicates. Number of different strings over Σ' of length at-most n is bounded by $|\Sigma|^{n+1}$. Hence we assume $\log(t) = O(n)$.

Now partition the strings into equivalence class they belong to $[c_1, \ldots c_q]$ w.r.t to $\equiv$. (Note q is $\leq poly(n)$.).

Now for each $j = 1 \ldots q$, apply the cross composition of A to obtain (c_j, k_j) s.t

(a) $k_j \leq poly(n)$. $\left[\begin{array}{l}\max\{x_1 \ldots x_m\} \leq O(n) \\ \log(t) \leq O(n)\end{array}\right]$.

(b) $c_j, k_j \in Q$ iff $\exists x \in c_j$ s.t $x \in L$.

Now apply the assumed compression Algo C, on each (c_j, l_{ij}), obtaining the string d_j.

$\therefore k_j \leq \text{poly}(n)$, $d_j' \leq \text{poly}(n)$.

Finally let $d = d_1 \# d_2 \# \ldots \# d_q$. From the construction, it is immediate that $d \in OR(R)$ iff $\exists i$ s.t $x_i \in L$.

Further each $|d_i| \leq \text{poly}(n)$ & $q \leq \text{poly}(n)$
$\Rightarrow |d| \leq \text{poly}(n)$.

□.

Corollary:= Longest path has no poly kernel unless $NP \subseteq co\text{-}NP/poly$.

Proof:= Let $\equiv$ be graphs with same no of vertices. (clearly poly equiv relation)

It is easy to see Hamiltonian path cross composes into longest path.

Given $G_1, \ldots, G_t$ s.t $|G_1| = |G_2| = \ldots = |G_t|$
Let $G = G_1 \cup \ldots \cup G_t$. & $k = n$.