FPT ALGORITHMS, PART I:

KERNELIZATION ALGORITHMS

Problems considered:

Vertex cover
Max Satisfiability
d-Hitting Set
Max Leaves Spanning Tree
An improved kernel for vertex cover

Introduction / Motivation

- Kernelization is a technique to obtain FPT algorithms.
- Kernelization also gives a theoretical framework for theoretically evaluating preprocessing algorithms.
- Kernelization algorithms are related to approximation algorithms.

ALGORITHM 1: Vertex Cover, Buss' kernel

• A *vertex cover* in a graph $G = (V, E)$ is a set $S \subseteq V$ such that every edge of G is incident with at least one vertex from S.
It is a k-VC if $|S| = k$.

k-Vertex Cover (k-VC):
INSTANCE: A graph G and integer k.
PARAMETER: k
QUESTION: Does G have a vertex cover S with $|S| \leq k$?

- An *isolated vertex* has degree zero.

Observation (1)

Let (G, k) be a k-VC instance, and let v be an isolated vertex. Then the k-VC instance $(G - v, k)$ is equivalent.

Observation (2)

Let (G, k) be a k-VC instance. If $v \in V(G)$ has degree at least $k + 1$, the k-VC instance $(G - v, k - 1)$ is equivalent.

Observation (3)

Let G be a graph with maximum degree k that admits a vertex cover with at most k vertices. Then $|E(G)| \leq k^2$.

Theorem

Let (G, k) be a k-VC instance. In polynomial time we can obtain an equivalent k-VC instance (G', k') with $|E(G')| \leq O(k^2)$.

Proof: Iteratively remove isolated vertices and vertices with degree at least $k + 1$, decreasing the parameter by one in the second case. By Observation 2, the resulting instance (G', k') is equivalent to the original instance.
By Observation 3, if $|E(G')| > k'^2$, we may return a trivial small NO-instance, which is again equivalent.
If $|E(G')| \leq k'^2$, we return (G', k'), which satisfies the bound since $k' \leq k$. □

Theorem

Let (G, k) be a k-VC instance. In polynomial time we can obtain an equivalent k-VC instance (G', k') with $|E(G')| \leq O(k^2)$.

• This preprocessing algorithm is easily extended to an FPT-algorithm:

Let (G, k) be a k-VC instance on n vertices. Preprocessing yields equivalent instance (G', k'), with (roughly speaking!) at most $k'^2 \leq k^2$ vertices. Consider all vertex subsets of size at most k' of G': if one of these is a VC, return YES. If not, return NO.

Complexity:

• Preprocessing takes polynomial time $n^{O(1)}$.
• There are at most $\binom{k'^2}{k'} \in O(k^{2k})$ vertex sets of G' to test.
• Testing whether vertex sets are vertex covers can be done in polynomial time $k'^{O(1)}$.

Total complexity: $n^{O(1)} + k^{2k} k^{O(1)} \approx n^{O(1)} + O(k^{2k})$.

- This preprocessing algorithm used a *parameter dependent preprocessing rule*: not so nice (not immediately applicable to optimization problem).

- Preprocessing algorithms of this type (*kernelization* algorithms) always give FPT algorithms with nice ‘additive’ complexities.

Kernelization: definition

An algorithm A for a parameterized problem (Q, κ) is a *kernelization algorithm* if for every instance X, in *polynomial time* (in $|X|$), A returns an *equivalent instance* X' with $|X'| \leq f(\kappa(X))$, for some function $f : \mathbb{N} \to \mathbb{N}$.

This is also called an $f(\kappa)$-*kernel*.

- Usually, $\kappa(X') \leq \kappa(X)$. This property is sometimes added to the definition.

The above algorithm for k-VC is a kernelization algorithm that returns an instance G' with $m = |E(G)| \in O(k^2)$ and $n = |V(G)| \in O(k^2)$.

We will sometimes (sloppily) ignore log factors, and call this an $O(k^2)$-kernel; note however that at least $m \log n = \Theta(k^2 \log k)$ bits may be needed to encode G'.

For graph problems, *vertex kernels* are important: e.g. suppose a graph G' is returned with $|E(G')| \leq k^2$ and $|V(G')| \leq ck$: this is an $O(k^2)$-(size) kernel, but a ck-*vertex kernel*.

Edge kernels are defined similarly.

Theorem

A problem P admits an FPT algorithm $\Leftrightarrow$ there is a kernelization algorithm for P.

- The $\Leftarrow$ direction is important: kernelization algorithms give FPT algorithms.
- However, the $\Rightarrow$ direction is just of theoretical importance: here no actual preprocessing is done, so the kernelization algorithm is practically irrelevant.
- We are usually only interested in *polynomial* kernels (where $f(\kappa)$ is polynomial).
- Polynomial kernelization algorithms are only known for parameterized problems obtained from optimization problems with the *standard parameterization*.

ALGORITHM 2: Maximum Satisfiability

Example:

$(x \vee \neg y \vee z) \wedge (\neg x \vee a)$ is a boolean formula in *CNF* consisting of two *clauses*, where x, y, z, a are the variables, which can occur as *positive or negative literals* (x resp. $\neg x$).

k-Max Sat:
INSTANCE: a boolean CNF-formula $F = \bigwedge_{i=1}^{m} C_i$ and integer k.
PARAMETER: k.
QUESTION: Does there exist a variable assignment satisfying at least k clauses?

• The *size* of a CNF-formula is the sum of clause lengths (# literals); we ignore log-factors again.

Trivial clauses

- A clause in F is *trivial* if it contains both a positive and negative literal in the same variable.

Observation

Trivial clauses are satisfied in any truth assignment.

Observation

Let F_n be obtained from formula F by removing all t trivial clauses, let $k' = k - t$. Then (F_n, k') and (F, k) are equivalent.

Long clauses

- For instance (F_n, k'), a clause in F_n is *long* if it contains at least k' literals, and *short* otherwise.

Observation

If F_n contains at least k' long clauses, (F_n, k') is a YES-instance.

Proof: Satisfy long clauses one by one by setting one of their literals appropriately. After x clauses have been satisfied, every (non-trivial) long clause contains still at least $k' - x$ free variables, so we can continue until at least k' clauses are satisfied. □

Observation

Let F_s be obtained from formula F_n by removing all $l \leq k'$ long clauses, and $k'' = k' - l$. Then (F_s, k'') and (F_n, k') are equivalent.

Proof: Clearly a truth assignment for F_n satisfying at least k' clauses satisfies at least $k' - l$ clauses of F_s.
In a truth assignment for F_s satisfying $k' - l$ clauses, all variables except at most $k' - l$ are free to be changed. This allows to satisfy the remaining l long clauses. □

Observation

If (F_s, k'') contains at least $2k''$ clauses, it is a YES-instance.

Proof: Take an arbitrary truth assignment T and its complement $\bar{T}$ obtained by flipping all variables. Every clause of F_s is satisfied in T or in $\bar{T}$ (or in both). □

An $O(k^2)$-kernel for MaxSat

An $O(k^2)$-kernelization algorithm on instance (F, k):

(1) Remove all t trivial clauses to obtain F_n, set $k' = k - t$.
(2) If (F_n, k') has at least k' long clauses, return YES.
(3) Remove all l long clauses to obtain F_s, set $k'' = k' - l$.
(4) If (F_s, k'') contains at least $2k''$ clauses, return YES.
(5) kernel (F_s, k'') now contains at most $2k''$ clauses with at most k' literals, so has size $O(k' \cdot k'') = O(k^2)$.

ALGORITHM 3: *d*-Hitting Set

- Vertex Cover is equivalent to 2-Hitting Set:

d-Hitting Set:
INSTANCE: A hypergraph $H = (V, E)$. with $|e| \leq d$ for all $e \in E$.
SOLUTION: A subset $S \subseteq V$ that intersects every $e \in E$ (a *hitting set*).
OBJECTIVE: Minimize $|S|$.

k-d-Hitting Set:
INSTANCE: A hypergraph $H = (V, E)$. with $|e| \leq d$ for all $e \in E$, and integer k.
PARAMETER: k.
QUESTION: Does H have a hitting set S with $|S| \leq k$?

- The kernel for k-VC can be generalized to k-d-Hitting Set: for every *fixed* d, a $O(k^d)$-kernel exists.

Sunflowers

- Let $H = (V, E)$ be a hypergraph. A *k-sunflower* in H consists of a set $S = \{e_1, \ldots, e_k\} \subseteq E$ and *core* $C \subseteq V$ such that for all $i \neq j$, $e_i \cap e_j = C$.
- Hypergraph $H = (V, E)$ is *d-uniform* if $|e| = d$ for all $e \in E$.

Lemma (Sunflower Lemma)

Let $H = (V, E)$ be a d-uniform hypergraph with more than $(k-1)^d d!$ edges. Then H has a k-sunflower (which can be found in polynomial time).

Proof of the Sunflower Lemma:

By induction on d.

If $d = 1$, then H has more than $k - 1$ (disjoint) edges, which gives a k-sunflower.

If $d \geq 2$, then we use the following induction hypothesis:

- Every $(d-1)$-uniform hypergraph with more than $(k-1)^{d-1}(d-1)!$ edges contains a k-sunflower.

Let $F = \{f_1, \ldots, f_l\}$ be a *maximal* set of disjoint hyperedges in H.
If $l \geq k$, then F is a sunflower with core $\emptyset$.
Otherwise, let $W = f_1 \cup \ldots \cup f_l$, which has $|W| \leq (k-1)d$.

Proof, continued:

Let $W = f_1 \cup \ldots \cup f_l$, which has $|W| \leq (k-1)d$.
H contains more than $(k-1)^d d!$ edges, and every edge of H is covered by W.
Thus there is an element $w \in W$ that hits more than

$$\frac{(k-1)^d d!}{(k-1)d} = (k-1)^{d-1}(d-1)!$$

edges.
Taking all of these edges and removing w from them yields a $(d-1)$-uniform hypergraph H' with more than $(k-1)^{d-1}(d-1)!$ edges. By induction, H' contains a k-sunflower S. Let C be its core.
Taking the corresponding edges in H yields a k-sunflower in H, with core $C \cup \{w\}$. □

- The above proof is easily translated to a polynomial time algorithm that constructs a k-sunflower.

A kernel for k-d-Hitting Set

Let F be a $(k+1)$-sunflower with core C in hypergraph H, and let S be a hitting set of H.

- If $S \cap C = \emptyset$, then C instead hits all 'petals' of F, so $|S| \geq k+1$.
- Therefore, H has a hitting set of size $k \Leftrightarrow$ the hypergraph H' with edge set $(E(H) \setminus F) \cup \{C\}$ has a hitting set of size k.
- Reduction rule: replace (H, k) by (H', k).

By the sunflower lemma, a *reduced* hypergraph H contains
- at most $(k-1)$ edges of size 1,
- at most $(k-1)^2 2!$ edges of size 2,
- ...
- at most $(k-1)^d d!$ edges of size d,

So it contains at most $(k-1)^d d! d$ edges in total.

Theorem

The above algorithm is a $(k-1)^d d! d$-edge *kernelization for* k-d-*Hitting Set.*

ALGORITHM 4: Maximum Leaves Spanning Tree

• A subgraph H of a graph G is *spanning* if $V(H) = V(G)$.

• A graph H is a *tree* if it is connected and has no cycles.

• A *leaf* of a graph (tree) is a vertex v with degree 1. $d(v)$ denotes the degree of v.

Max-Leaves Spanning Tree:
INSTANCE: A connected graph G.
SOLUTION: a spanning tree T of G.
OBJECTIVE: maximize the number of leaves of T.

By k-Leaf Spanning Tree or *k-LST* we denote the standard parameterization of this problem.

Reduction Rules

Observation (Degree 2 Rule)

Let (G, k) *be a* k*-LST instance, and let* $uv \in E(G)$ *with* $d(u) = d(v) = 2$*. If* $G - uv$ *is connected, then* $(G - uv, k)$ *is an equivalent instance.*

- A *bridge* in a connected graph G is an edge uv such that $G - uv$ is disconnected.

Observation (Bridge Rule)

Let (G, k) *be a* k*-LST instance, and let* $uv \in E(G)$ *with* $d(u) \geq d(v) \geq 2$*. If* uv *is a bridge, then* contracting uv *gives an equivalent instance* (G', k)*.*

- Conclusion: a *reduced instance* (G, k) contains no adjacent vertices of degree 2, and no bridges between degree ≥ 2 vertices.

Theorem

A connected simple graph G on n vertices, with no adjacent vertices of degree 2 and no bridges between two non-leaves, contains a spanning tree with at least $n/5$ leaves.

Proof: For any (possibly non-spanning) tree subgraph T of G we define

- $n(T) = |V(T)|$,
- $l(T)$ is the number of leaves of T, and
- $d(T)$ is the number of *dead leaves*, which are leaves of T with no neighbors outside of T.

A tree T with $4l(T) + d(T) \geq n(T)$ exists: w.l.o.g. G contains a vertex v with $d(v) \geq 3$; consider v and all its neighbors.

Proof, continued:

Given a tree T with $4l(T) + d(T) \geq n(T)$, a larger tree T' with $4l(T') + d(T') \geq n(T')$ exists if:

(A) T contains a vertex with $d \geq 2$ neighbors not in T, or a non-leaf with one neighbor not in T.
($\Delta l \geq d - 1$, $\Delta n \leq d$, resp. $\Delta d = 1 = \Delta n$, $\Delta l = 0$.)

(B) If (A) does not apply but there is a $v \in V(G) \setminus V(T)$ with either at least two neighbors in T, or $d(v) = 1$.
($\Delta d \geq 1$, $\Delta n = 1$.)

(C) If there is a $v \in V(G) \setminus V(T)$ with exactly one neighbor in T and $d = d(v) \geq 3$.
($\Delta l \geq d - 2$, $\Delta n \leq d$.)

Proof, continued:

Given a tree T with $4l(T) + d(T) \geq n(T)$, a larger tree T' with $4l(T') + d(T') \geq n(T')$ exists if:

(D) If (B) and (C) do not apply but T is not yet spanning: there is a $u \in V(T)$ with neighbor in T.

- $d(u) = 2$ (by (B) and (C)), and
- u has a neighbor $v \in V(G) \setminus V(T)$ (by (B)).

$d(v) \neq 2$ (no degree 2 neighbors) and $d(v) \neq 1$ (u and its other neighbor $w \neq v$ would form a bridge uw), so $d = d(v) \geq 3$, and v has no neighbors in T (by (C)).

Therefore: $\Delta l \geq d - 2$, $\Delta n \leq d + 1$.

We conclude that a spanning tree T with $4l(T) + d(T) \geq n(T)$ exists. In a spanning tree, $d(T) = l(T)$, and $n(T) = n$, so $l(T) \geq n/5$. □

A $5k$-vertex kernel for k-Leaf Spanning Tree

The following algorithm gives a $5k$-*vertex*-kernel for a k-LST instance (G, k):

- Apply the degree 2 rule and bridge rule until an equivalent, irreducible instance (G', k) is obtained.

If $|V(G')| \geq 5k$, it is a YES instance (Theorem 5). Otherwise (G', k) is the kernel.

ALGORITHM 5: a $2k$-vertex kernel for Vertex Cover

- Nemhauser-Trotter
- Linear Programming
- Crown Decompositions

Vertex Cover - Integer Program

Goal: find a minimum vertex cover for graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$.

The 0/1 variable x_i indicates whether v_i is chosen in the vertex cover.

Integer linear programming formulation of Vertex Cover:

VC-IP:

$$\begin{aligned} \min \quad & \textstyle\sum_{i=1}^{n} x_i & \\ \text{s.t.} \quad & x_i + x_j \geq 1 & \forall v_i v_j \in E \\ & x_i \in \{0, 1\} & \forall i \in \{1, \ldots, n\} \end{aligned}$$

Vertex Cover - Relaxed

Goal: find a minimum vertex cover for graph $G = (V, E)$, with $V = \{v_1, \ldots, v_n\}$.

The 0/1 variable x_i indicates whether v_i is chosen in the vertex cover.

Half-integer linear programming relaxation of Vertex Cover:

VC-Rel:

$$\begin{array}{lll} \min & \sum_{i=1}^{n} x_i & \\ \text{s.t.} & x_i + x_j \geq 1 & \forall v_i v_j \in E \\ & x_i \in \{0, \frac{1}{2}, 1\} & \forall i \in \{1, \ldots, n\} \end{array}$$

- An optimal solution to VC-Rel can be found in polynomial time.

(Q1) How does this give a $2k$-vertex kernel?

(Q2) How exactly can VC-Rel be solved in polynomial time?

Properties of a vertex partition

Given an optimal solution x to VC-Rel on graph $G = (V, E)$, partition V as follows:

$C_0 = \{v_i : x_i = 1\}$
$I_0 = \{v_i : x_i = 0\}$
$V_0 = \{v_i : x_i = \frac{1}{2}\}$

Lemma

Let vertex partition $\{C_0, I_0, V_0\}$ be deduced from an optimal VC-Rel solution. Then:
(1) If D is a VC for $G[V_0]$, then $D \cup C_0$ is a VC for G.
(2) $G[V_0]$ has no VC of size less than $|V_0|/2$.
(3) There is a minimum VC C of G with $C_0 \subseteq C$.

Observation (11)

Vertices in I_0 only have neighbors in C_0.

Proof of Property (1):

$C_0 = \{v_i : x_i = 1\}$
$I_0 = \{v_i : x_i = 0\}$
$V_0 = \{v_i : x_i = \frac{1}{2}\}$

Property (1): If D is a VC for $G[V_0]$, then $D \cup C_0$ is a VC for G.

Proof: Consider an edge not covered by D, so
$v_i v_j \in E(G) \setminus E(G[V_0])$.
If it has at least one end vertex in C_0 it is clearly covered by $D \cup C_0$.
Since edges with one end vertex in I_0 have their other end vertex in C_0 (Observation 11), we have considered all types of edges. □

Proof of Property (2):

$C_0 = \{v_i : x_i = 1\}$
$I_0 = \{v_i : x_i = 0\}$
$V_0 = \{v_i : x_i = \frac{1}{2}\}$

Property (2): $G[V_0]$ has no VC of size less than $|V_0|/2$.

Proof: If C^* is a VC of $G[V_0]$ with $|C^*| < |V_0|/2$, then by (1), setting $y_i = 1$ for all $v_i \in C^* \cup C_0$ is a VC of G with $\sum_i y_i = |C_0| + |C^*| < |C_0| + \frac{1}{2}|V_0|$, contradicting the optimality of x. □

Proof of Property (3):

$C_0 = \{v_i : x_i = 1\}$
$I_0 = \{v_i : x_i = 0\}$
$V_0 = \{v_i : x_i = \frac{1}{2}\}$

Property (3): There is a minimum VC C of G with $C_0 \subseteq C$.

Proof: Let S be a minimum VC of G.
We first show that $|S \cap I_0| \geq |C_0 \setminus S|$:

Construct a new solution y to VC-Rel as follows:

- $y_i = \frac{1}{2}$ if $v_i \in (S \cap I_0) \cup (C_0 \setminus S)$.
- $y_i = x_i$ otherwise.

Claim
y is a feasible solution to VC-Rel.

Construct a new solution y to VC-Rel as follows:

- $y_i = \frac{1}{2}$ if $v_i \in (S \cap I_0) \cup (C_0 \setminus S)$.
- $y_i = x_i$ otherwise.

Claim

y is a feasible solution to VC-Rel.

Proof: Consider an edge $v_i v_j$.
If $\{v_i, v_j\} \subseteq V_0 \cup C_0$ then $x_i + x_j \geq \frac{1}{2} + \frac{1}{2}$.
So w.l.o.g $v_i \in I_0$. Then $v_j \in C_0$ (Observation 11). If $v_j \in S$ then $y_j = 1$.
Otherwise, since S is a VC, $v_i \in S$, so $x_i + x_j \geq \frac{1}{2} + \frac{1}{2}$. □

Proof of Property (3), continued:

Since x is an optimal solution to VC-Rel, we have

$$0 \leq \sum_i y_i - \sum_i x_i = \frac{1}{2}|S \cap I_0| - \frac{1}{2}|C_0 \setminus S|,$$

so $|C_0 \setminus S| \leq |S \cap I_0|$.

Now let $C = (S \setminus I_0) \cup C_0$.

The above inequality shows that $|C| \leq |S|$.

C is a VC:
it covers all edges incident with C_0, and
therefore all edges incident with I_0 (Observation 11), and
all edges with both end vertices in V_0 (since S is a VC). $\square$

A $2k$-vertex kernel for Vertex Cover

Properties:
(1) If D is a VC for $G[V_0]$, then $D \cup C_0$ is a VC for G.
(2) $G[V_0]$ has no VC of size less than $|V_0|/2$.
(3) There is a minimum VC C of G with $C_0 \subseteq C$.

(Q): Assuming we can solve VC-Rel in polynomial time, how does this give a $2k$-vertex kernel for VC?

(A): Consider $(G', k') = (G[V_0], k - |C_0|)$.
If $k' \geq |V(G')|/2$ then return (G', k'), otherwise return NO.

- If G' has a k'-VC, then G has a k-VC *(Property (1))*.
- If G has a k-VC S, then there is one that contains C_0 *(Property (3))*, so $S \setminus C_0$ is a k'-VC of G'.
- If $k' < |V(G')|/2$, then (G', k') is a NO-instance by *Property (2)*. Otherwise, $|V(G')| \leq 2k' \leq 2k$.

The only question that remains:

(Q2) How can VC-Rel be solved in polynomial time?

(A1) Using linear programming techniques (sketch):

• Relax VC-Rel further by allowing all values $0 \leq x_i \leq 1$.
• This yields a linear program without (half-)integer variables, which can be solved in polynomial time.
• Any such solution can efficiently be transformed to a VC-Rel solution of the same value. *(See Flum and Grohe 2006, p.218-219.)*

(A2) Using matchings in bipartite graphs.

• A graph $B = (V, E)$ is *bipartite* if a partition $\{L, R\}$ of V exists such that all edges of B have one end vertex in L and one end vertex in R.
L and R are the *sides* of the bipartition. (For connected graphs, these are basically unique.)

Let $G = (V, E)$ be the VC instance.

Construct a bipartite graph B as follows:

• For all $v \in V$, $V(B)$ contains a vertex v and a vertex v'.
(Notation: for any $S \subseteq V$, $S' = \{v' : v \in S\}$, so $V(B) = V \cup V'$.)
• For all $uv \in E$, $E(B)$ contains an edge uv' and an edge $u'v$.

Lemma

VC-Rel on G has a solution x with $\sum_i x_i = z \Leftrightarrow B$ has a vertex cover S with $|S| = 2z$.

Lemma proof, first direction

Lemma

VC-Rel on G has a solution x with $\sum_i x_i = z \Rightarrow B$ has a vertex cover S with $|S| = 2z$.

Proof: Construct S as follows from x:

- for all i with $x_i = 1$: add v_i and v'_i to S.
- for all i with $x_i = \frac{1}{2}$: add v_i to S.

Consider $v_i v'_j \in E(B)$.

S covers this edge unless $x_i = 0$.
But then $x_j = 1$ (since $v_i v_j \in E(G)$), so $v'_j \in S$. $\square$

Lemma proof, second direction

Lemma

VC-Rel on G has a solution x with $\sum_i x_i = z \Leftarrow B$ has a vertex cover S with $|S| = 2z$.

Proof: Construct x as follows from S:

- for all i with $v_i \in S$ and $v'_i \in S$: $x_i = 1$
- for all i with either $v_i \in S$ or $v'_i \in S$: $x_i = \frac{1}{2}$.
- for all other i: $x_i = 0$.

Consider $v_i v_j \in E(G)$.

Since S covers both $v_i v'_j$ and $v_j v'_i$, one of these cases holds:

- $v_i \in S$ and $v'_i \in S$. Then $x_i = 1$.
- $v_j \in S$ and $v'_j \in S$. Then $x_j = 1$.
- $v_i \in S$ and $v_j \in S$. Then $x_i \geq \frac{1}{2}$ and $x_j \geq \frac{1}{2}$.
- $v'_i \in S$ and $v'_j \in S$. Then $x_i \geq \frac{1}{2}$ and $x_j \geq \frac{1}{2}$. □

Finding a Minimum Vertex Cover in a bipartite graph: König's Theorem

- A *matching* in a graph G is a set of edges $M \subseteq E(G)$ that share no end vertices (every $v \in V(G)$ is incident with at most one edge of M).
A vertex $v \in V(G)$ is *saturated* by M if it is incident with an edge of M.

Theorem (König)

For a bipartite graph B, the size of a minimum vertex cover equals the size of a maximum matching, and both can be found in polynomial time.

A proof sketch of Koenig's Theorem

Theorem (König)

For a bipartite graph B, the size of a minimum vertex cover equals the size of a maximum matching, and both can be found in polynomial time.

- Clearly, since every matching edge needs to be covered, $|M| \leq |C|$ holds for any matching M and any VC C, so the challenge lies in proving equality.

- Let B be a graph with matching M. A path P in B is *alternating* if its edges are alternatingly in M / not in M. An alternating path in B is *augmenting* if its end vertices are not saturated by M.

- If there is an augmenting path P, a larger matching M' can be found by 'flipping all edges of P' (that is, $M' = (M \setminus E(P)) \cup (E(P) \setminus M)$).

When given a bipartite graph B with sides V and V', the following algorithm finds a matching M and vertex cover C with $|C| = |M|$:

(1) Start with $C = V$, $M = \emptyset$.

(2) If $|C| = |M|$ then return C and M, halt.

(3) Choose an unsaturated vertex $v \in C$, and construct an *alternating search tree* subgraph T of B, rooted at v.

(4) If T contains an augmenting path P, then augment M using P, goto (2).

(5) Otherwise, find a vertex set S with $v \in S$ such that

- $N(S)$ is saturated by M, and
- $|N(S)| \leq |S| - 1$.

Then $C' = (C \setminus S) \cup N(S)$ is a VC with $|C'| < |C|$. Set $C := C'$, goto (2).

Hall's Theorem

The following theorem also follows:

Theorem (Hall)

A bipartite graph B with sides V and V' has a matching saturating V if and only if there is no $S \subseteq V$ with $|N(S)| < |S|$.

Summary

- In polynomial time, we can find a matching M and VC C with $|M| = |C|$, which therefore are maximum resp. minimum.

- Applying this procedure to the bipartite graph B constructed from G, we can solve VC-Rel on G in polynomial time.

- This concludes the $2k$-vertex kernelization for k-Vertex Cover.

Crowns

- A *crown* in a graph G is a pair (C_0, I_0) of sets C_0 and I_0 such that

(a) all neighbors of I_0 are part of C_0,
(b) the edges between C_0 and I_0 contain a matching M that saturates all vertices of C_0.

- Property (a) is satisfied by the C_0 and I_0 returned by VC-Rel (Observation 11).
- In the proof of Lemma 2, we showed that C_0 is a minimum VC for $G[C_0 \cup I_0]$, which by Koenig's Theorem implies property (b).

- Therefore, an optimal solution to VC-Rel yields a crown in G if $C_0 \neq \emptyset$.
- Conversely, if G contains a crown (C, I) with $|C| < |I|$, VC-Rel has a solution with objective at most $|C| + \frac{1}{2}|V \setminus C \setminus I| < |V|/2$, so we will find a crown in polynomial time.

• Our earlier arguments showed that if (C, I) is a crown of G, then (G, k) and $(G - C - I, k - |C|)$ are equivalent k-VC instances.

• If $G = (V, E)$ contains no crown (C, I) with $|C| < |I|$, then every VC S of G has $|S| \geq |V|/2$.

Conclusion: A different way to express this $2k$-vertex kernel for k-VC: find crowns (C, I) with $|C| < |I|$ in polynomial time if they exist, and reduce them. A crownless graph is a $2k$-kernel.

• Crown reductions have also been used to find kernelizations for different problems.

Corollary: a 2-approximation algorithm for Vertex Cover

- A polynomial time algorithm for a minimization (maximization) problem is an *α-approximation* algorithm if it returns a feasible solution S with value$(S) \leq \alpha$value(opt) (resp. value$(S) \geq \frac{1}{\alpha}$value$(opt)$).

The following is a 2-approximation for Minimum Vertex Cover on graph G with $n = |V(G)|$:

- Apply the $2k$-kernelization algorithm to (G, n), which yields $(G', n - |C_0|)$.

- G contains an optimal vertex cover C_G^{opt} with $C_0 \subseteq C_G^{opt}$.

- $V_0 \cup C_0$ is a vertex cover for G with

$$|V_0 \cup C_0| \leq |C_0| + 2|C_{G'}^{opt}| \leq |C_0| + 2|C_G^{opt} \setminus C_0| \leq 2|C_G^{opt}|.$$

- ck-vertex kernels for ‘vertex subset’ problems usually yield c-approximation algorithms for the corresponding optimization problem.

Example: For Maximum Leaves Spanning Tree, the $5k$-vertex kernelization gives a 5-approximation algorithm.

Kernelization: summary

• For parameterized decision problems obtained from optimization problems, kernelization algorithms are a method to obtain FPT algorithms.

• These are preprocessing algorithms that can add to any algorithmic method (e.g. approximation algorithms).

• Kernelization algorithms usually consist of *reduction rules*, which reduce simple local structures (degree 1 vertices / high degree vertices / long clauses, etc), and a bound $f(k)$ for *irreducible* instances (X, k) that allows us to
-return NO if $|X| > f(k)$ for minimization problems, or
-return YES if $|X| > f(k)$ for maximization problems.

Designing kernelization algorithms

- What are the trivial substructures, where an optimal solution of a certain form can be guaranteed?

- Is there a reduction rule reflecting this?

- Can a bound be proved for irreducible instances? If not, which structures are problematic? Etc...