# Bounded Search Trees

We have seen a $\mathcal{O}(2^k(n+m))$ algorithm. Can we do better?

ALG1 $(G, k)$: Returns Yes iff $G$ has a vertex cover of size $\leq k$.

```
// halting cases:
if k ≤ 0 and E(G) ≠ ∅ : answer NO
if k ≥ 0 and E(G) = ∅ : answer YES

// Reduction Rules:
.
.
.

// Branching:
return ALG1(G \ N[v], k-1) or ALG1(G \ N²[v], k-d(v))
                for some non-isolated vertex v
```

Correctness of the algorithm: for any vertex cover $S$ of $G$, for any non-isolated vertex $v \in V(G)$ we have either $v \in S$ or $N(v) \subseteq S$

Naive analysis: Branching tree with width $=2$ and depth $\leq k \Rightarrow 2^k$ nodes, each node takes polynomial time.

idea of improvement: we would like $k$ to decrease more.

Reduction Rule 1: if $\exists v \in V(G)$ with $d(v) = 1$, then remove $v$ and its neighbor ($\Rightarrow$ its neighbor is in some optimal solution)

Better: Reduction Rule 2: if $d(v) \leq 2$ for all $v \in V(G)$: conclude in polynomial time (why? exercise).

BTW: we cannot extend it to "$d(v) \leq 3$" because the problem is NP-h if $\Delta(G)=3$.

$\Rightarrow$ we obtain a search tree where $k$ is decreased by 1 in one branch, and by $\geq 3$ in the other.

Let $T(k)$ be the number of nodes in the branching tree

Let $L(k)$ be the number of leaves in the branching tree

Lemma: $T(k) \leq 2L(k) - 1$

Proof: exercise. hint: no vertex of degree 2 apart from root and leaves

$\Rightarrow$ bounding $L(k)$ is enough.

~~Lemma: $L(k) \leq 1.4656^k$~~

~~Proof: by induction. Base case for $k \leq 0$.~~

We have $L(k) \leq \begin{cases} L(k-1) + L(k-3) & \text{if } k > 3 \\ 1 & \text{otherwise} \end{cases}$ (1)

Lemma: $L(k) \leq 1.4656^k$

proof: by induction. true if $k=0$.

$$L(k) \leq 1.4656^{k-1} + 1.4656^{k-3}$$
$$= 1.4656^{k-3}\underbrace{(1 + 1.4656^2)}_{\leq 1.4656^3}$$
$$\leq 1.4656^k$$

□

We just proved:

theorem: Vertex Cover can be solved in time $O(1.4656^k(n+m))$

## Solving general recurrence reduction

the problem: find the smallest upper bound of $T(n)$, where:

$$T(n) = \begin{cases} T(n-d_1) + T(n-d_2) + \dots + T(n-d_p) & \text{if } n \geq d_i \ \forall i \\ 1 & \text{if } n < d_i \ \forall i \end{cases}$$

we say the algorithm has branching vector $(d_1, \dots, d_p)$

(ALG 1 has branching vector $(1,3)$)

Solutions are of the form $T(n) \leq \lambda^n$, where $\lambda$ satisfies

$$\lambda^n \geq \lambda^{n-d_1} + \dots + \lambda^{n-d_p}$$

$\lambda_0$ called branching factor.

$\Rightarrow$ find the unique positive root $\lambda_0$ of the polynomial

$$P(\lambda) = \lambda^d - \lambda^{d-d_1} - \dots - \lambda^{d-d_p} \qquad (d = \max d_i)$$

ex: for ALG 1, we have to solve $\lambda^3 - \lambda^2 - 1 = 0$

Remark: Sometimes $n$ will be the size of our instance (ex: $|V|$), sometimes just a parameter (like for ALG 1).

Some properties of the branching vectors: let $\tau(d_1, \dots, d_p)$ be the branching factor of $(d_1, \dots, d_p)$:

- $\tau(d_1, \dots, d_p) > 0$
- $\tau(d_1, \dots, d_p) = \tau(d_{\pi(1)}, \dots, d_{\pi(p)})$ for any permutation $\pi$.
- if $d_1 > d_1'$ then $\tau(d_1, \dots, d_p) < \tau(d_1', d_2, \dots, d_p)$
- $\tau(k,k) \leq \tau(i,j)$ whenever $i+j = 2k$
- $\tau(i,j) > \tau(i+\varepsilon, j-\varepsilon)$ if $i<j$, $\forall\ \varepsilon < \frac{j-i}{2}$

↳ "better be balanced" (but decrease the sum)

# Enumerating all maximal independent sets of a graph

~~Observation: Let $S \subseteq V$ be a maximal~~

⚠ difference between maximum and maximal

**Observation**: let $S$ be a maximal independent set in $G$. $\forall v \in V(G)$, $\exists y \in N[v]$ s.t. $y \in S$ and $N(y) \cap S = \emptyset$.

Consider the following algorithm:

EnumIS $(G, S)$ with $S \subseteq V(G)$
- if $V(G) = \emptyset$ then print $S$
- else: pick $v \in V(G)$ of minimum degree,
  - for every $y \in N[v]$: run EnumIS $(G \setminus N[y], S \cup \{y\})$

$\Rightarrow$ in each leaf, we print a maximal independent set.

**Remark**: we cannot bound the number of nodes by a function of the size of a maximum IS. (hence this is not FPT param. by solution size).

Correctness follows from the 1st observation (all maximal IS are printed).

$$L(n) \leq L(n - d(v) - 1) + \sum_{y \in N(v)}' L(n - d(y) - 1)$$

since $v$ is of min. degree:

$$L(n) \leq (d(v) + 1)\, L(n - d(v) - 1)$$

letting $s = d(v) + 1$, we have $L(n) \leq s L(n - s)$

$\Rightarrow$ branching vector $(\underbrace{s, \ldots s}_{s \text{ times}})$ What is the branching factor?

find the root of $\lambda^s - \underbrace{\lambda^0 - \ldots - \lambda^0}_{s \text{ times}} = 0 \Leftrightarrow \lambda^s - s = 0$

solution is $\lambda = s^{1/s}$ for integral values of $s$, $s \mapsto s^{1/s}$ has its maximal value for $s = 3$

$\Rightarrow L(n) \leq 3^{n/3}$

**Theorem**: Any graph has at most $3^{n/3}$ maximal independent sets

**Exercise**: Find a graph with $3^{n/3}$ maximal independent sets.

# Solving k-SAT better than $2^n$

Vocabulary: a CNF formula $\varphi$ is a conjunction of clauses $c_1 \wedge \dots \wedge c_m$

a clause is a disjunction of literals $c_i = l_1 \vee \dots \vee l_k$;

a literal $l_j$ is either $x$ or $\bar{x}$ for some variable $x \in X$. $|X| = n$

a truth assignment assigns a Boolean value to each variable.

**k-SAT problem:**
- input: a CNF formula $\varphi$ with clauses of size $\leq k$
- question: is there a truth assignment which evaluates to True?

Naive algorithm (brute force): $O^*(2^n)$

Open problem: Can SAT (with no constraint on the size of the clauses) be solved in $O^*((2-\varepsilon)^n)$ for some $\varepsilon > 0$?

Here: we will see a $O^*(\alpha_k^n)$ algorithm solving k-SAT, with $\alpha_k < 2$ for every fixed $k \geq 3$.

By Monien and Speckenmeyer. $k$ is a fixed integer

Let $\varphi$ be a CNF k-SAT formula. Pick a clause $c = (l_1 \vee l_2 \vee \dots \vee l_q)$

For $i = 1 \dots q$, define $\varphi_i$ as the clause where:

- for all $j = 1, \dots, i-1$, variable corresponding to $l_j$ is set so that $l_j$ = False
- variable corresponding to $l_i$ is set so that $l_i$ = True

$\Rightarrow$ in $\varphi_1$, $l_1$ = True,
$\Rightarrow$ in $\varphi_2$, $l_1$ = False, $l_2$ = True
$\Rightarrow$ in $\varphi_3$, $l_1$ = false, $l_2$ = false, $l_3$ = True.

Propagation routine: if, after setting some variables, we have for some clause:

- $\rightarrow$ one literal evaluates to True $\Rightarrow$ delete the clause.
- $\rightarrow$ one literal evaluates to false $\Rightarrow$ delete the literal.

**Observation** (proof as exercise): $\varphi$ is satisfiable iff $\exists i$ : $\varphi_i$ is satisfiable.

**Algorithm**: k-SAT$'$($\varphi$)

- if $\varphi$ contains an empty clause, return False
- if $\varphi$ is empty, return True.
- Pick any clause $c = (l_1 \vee \dots \vee l_q)$
- Return $\bigvee_{i=1}^{q}$ k-SAT$'$($\varphi_i$) for every $i$: construct $\varphi_i$, run the propagation on it

## Analysis of Running time

$\forall i = 1 \dots q$, the number of (non fixed) variables of $\varphi_i$ is $n - i$

$\Rightarrow$ branching vector $(1, 2, \dots, q)$

$$L(n) \leq L(n-1) + L(n-2) + \dots + L(n-q)$$

$\Rightarrow$ unique positive root $\beta_q$ of $\lambda^q = \lambda^{q-1} - \lambda^{q-2} - \dots - \lambda - 1$.

**Theorem**: k-SAT can be solved in $\mathcal{O}(\beta_q^n (n+m))$ time

$\beta_2 < 1.6181$, $\beta_3 \leq 1.8393$, $\beta_4 \leq 1.9276$

**Improvement**: idea 1: branch on a clause with minimum size.

**Definition**: a partial assignment $t$ is called an autark if for every clause $c$, either:

- no literal of $c$ is set by $t$
- there exist a literal of $c$ which evaluates to true by $t$.

**Lemma 1**: let $t$ be an autark assignment of $\varphi$, and denote by $\varphi'$ the formula obtained after propagating $t$.
Then $\varphi$ is satisfiable iff $\varphi'$ is.

**Proof**: exercise

**Lemma 2**: if $t$ is an assignment of $\varphi$ which is not an autark, then after the propagation there is a clause with strictly less literals. **Proof** follows for definition.


**Algo** k-SAT 2 ($\varphi$):

// halting cases (same as previously).
pick a clause $c$ of minimum size.
Construct $\varphi_i$ for all $i = 1 \dots q$, propagate.
if $\varphi_i$ is an autark for some $i$, then
  return k-SAT2($\varphi_i$)
else
  return $\bigvee_{i=1}^{q}$ k-SAT2($\varphi_i$)

~~branching vector $(1, 2, \dots, q)$~~

$\Rightarrow$ apart from the first branch, we solve an instance of $(k-1)$-SAT

**Theorem**: k-SAT can be solved in time $\mathcal{O}(\beta_{k-1}^n (n+m))$.