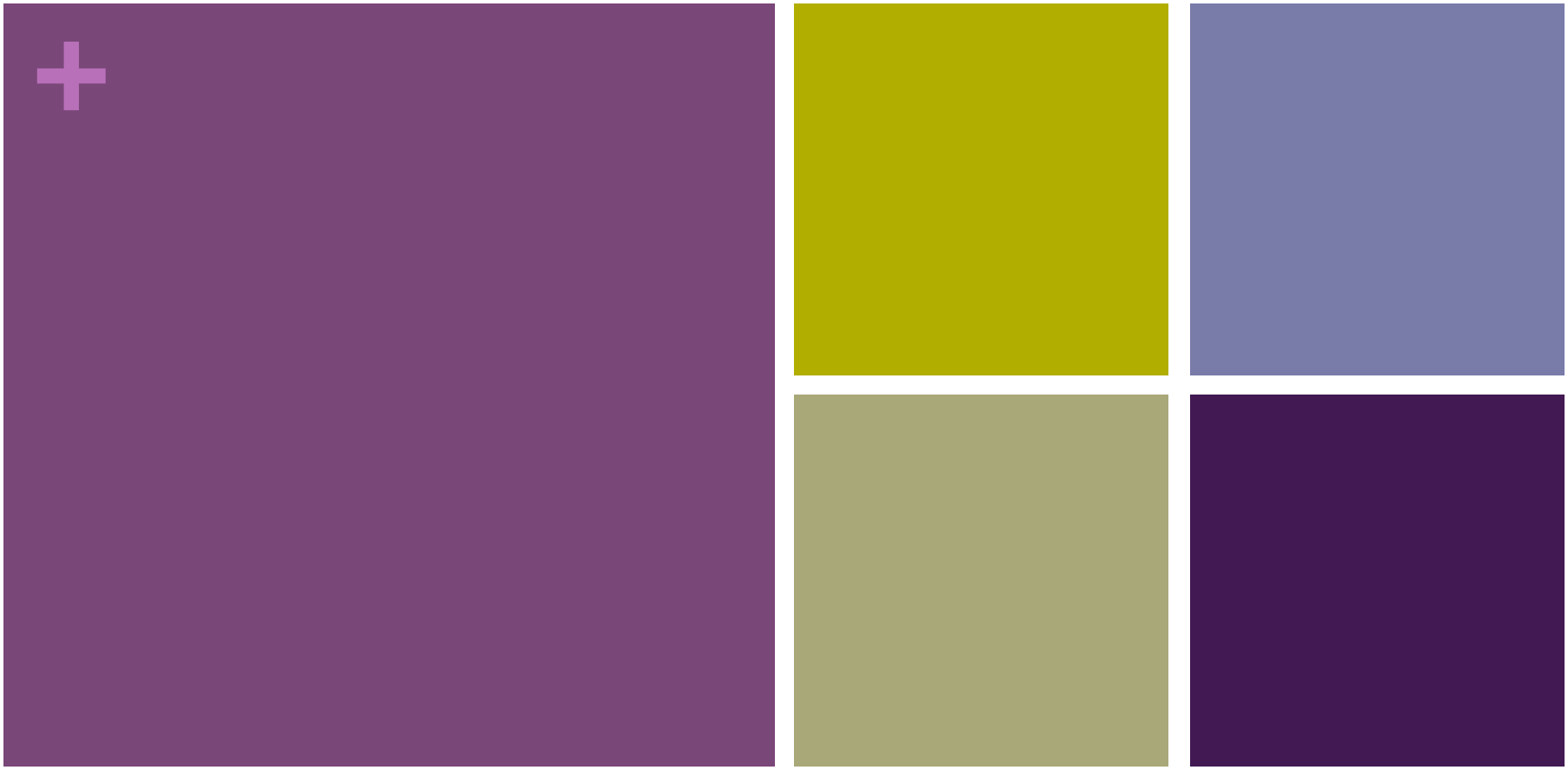


Exact exponential and fixed parameter algorithms

FPT Slides based on original material by:
Rolf Niedermeier and Mike Fellows

Map of the next 45 minutes

- By the end of the first hour you will be able to:
  - Define the O*() notaion
  - Explain how to solve TSP in $O^*(2^n)$
  - Explain how to solve MIS in $O^*(3^{n/3})$

Why bother?

10^6 Instructions per second Input size $n = 10^6$

Time complexity	Running time
n	1 sec.
$n \log n$	20 sec.
n^2	12 days
2^n	40 quadrillion (10^{15}) years

Motivation

- NP-complete problems
- Better understanding of NP-hard problems
- $O(2^n)$ to $O(1.4^n)$→ multiplicatively larger instance sizes possible

e.g. n^4 vs.$n2^{n/10}$ is worse for $n<=100$

Alan Perlis:

> "for every polynomial-time algorithm you have, there is an exponential algorithm that I would rather run."

Exact exponential algorithms

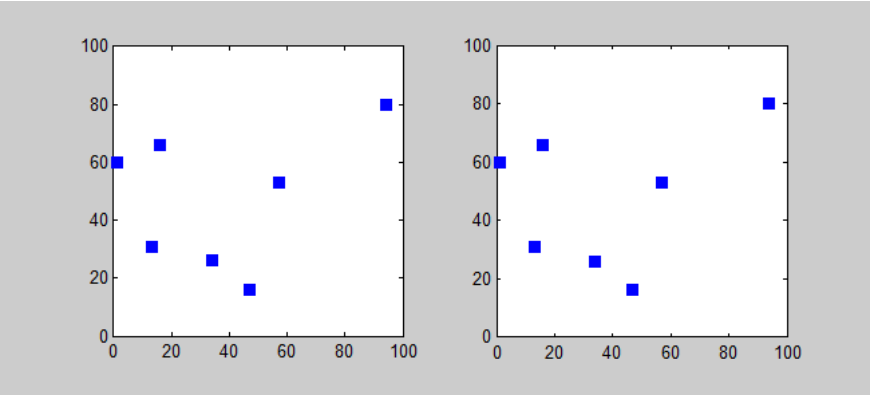
- O* notation
  - $f = O(g)$ → Exist n_0 and c s.t from $n > n_0$, $f(n) < c \cdot g(n)$
  - $f(n) = O*(g(n))$ → $f(n) = O(g(n)poly(n))$
- Types of problems:
  - Subset – Maximum independent set
  - Permutation - TSP
  - (Partition – Graph Coloring)

Permutation: TSP

- Given: Cities $\{c_1, c_2 \ldots c_n\}$ with distances $d(c_i, c_j)$
- Find the shortest path going through all the cities exactly once and returning to the starting point
- Another point of view: Find the permutation π of $\{1 \ldots n\}$ that minimizes:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)})$$

Naïve solution : n!



Improving O*(n!)

- S - subset of $\{c_2 \dots c_n\}$, $c_i \in S$
- Computing for every pair (S, c_i) the optimal tour of S, starting at c_1 and ends at c_i → $OPT[S, c_i]$
- $|S| = 1$ is just $d(c_1, c_i)$
- $|S| > 1$
  - $OPT[S,c_i] = \min\{OPT[S\backslash\{c_i\},c_j]+d(c_j,c_i): c_j \in S\backslash\{c_i\}\}$.

Adding the distance to c_i

A smaller group

For all elements in the smaller group

Solution explained

Algorithm `tsp`$(\{c_1, c_2, \ldots c_n\}, d)$**.**
Input: Set of cities $\{c_1, c_2, \ldots, c_n\}$ and for each pair of cities c_i, c_j the distance $d(c_i, c_j)$.
Output: The minimum length of a tour.

for $i = 2$ *to* n **do**
  $OPT[c_i, c_i] = d(c_1, c_i)$
for $j = 2$ *to* $n-1$ **do**
  forall $S \subseteq \{2, 3, \ldots, n\}$ *with* $|S| = j$ **do**
    $OPT[S, c_i] = \min\{OPT[S \setminus \{c_i\}, c_k] + d(c_k, c_i) : c_k \in S \setminus \{c_i\}\}$
return $\min\{OPT[\{c_2, c_3, \ldots, c_n\}, c_i] + d(c_i, c_1) : i \in \{2, 3, \ldots, n\}\}$

- The final solution:

$OPT[\{c_2,\ldots,c_n\},c_i]+d(c_i,c_1) \quad i \in \{1 \ldots n\}$

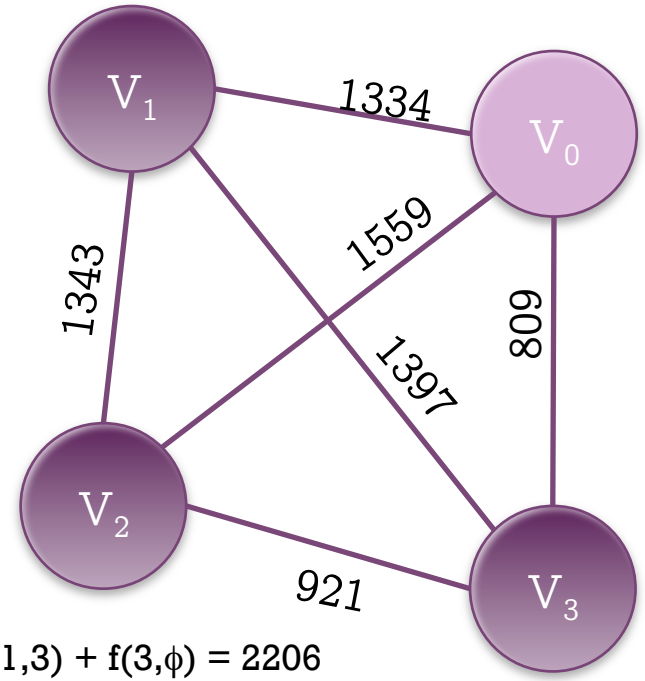
Class ex.

For i=1 to 3 do
    $f[v_i,s] = d(v_0,v_i)$
    For j= 1 to 3 do
        for all $S \subseteq \{1,2,3\}$ with $|S| = j$ do
            $f[v_i, s] = \min\{f[S\setminus\{v_i\},v_k]+d(v_k,v_i): v_k \text{ from } S \setminus\{v_i\}\}$
    Return $\min\{d(v_i,v_0) + f[v_i,\{v1,v2,v3\}] : i \text{ from } \{1,2,3\}\}$

$f(1, \phi) = d(1,0) = 1334$

$f(2, \phi) = d(2,0) = 1559$

$f(3, \phi) = d(3,0) = 809$



f(2,{1})= d(2,1) + f(1,ϕ) = 2677 f(3,{1})= d(3,1) + f(1,ϕ) = 2731
f(1,{2})= d(1,2) + f(2,ϕ) = 2902 f(3,{2})= d(3,2) + f(2,ϕ) = 2480 f(1,{3})= d(1,3) + f(3,ϕ) = 2206

$|S| = 2$

For i=1 to 3 do
    $f[v_i,s] = d(c_0,c_i)$
For j= 1 to 3 do
    for all $S \subseteq \{1,2,3\}$ with $|S| = j$ do
        $f[v_i, s] = \min\{F[S\backslash\{v_i\},v_k]+d(v_k,v_i): v_k \text{ from } S \backslash\{v_i\}\}$
Return $\min\{d(v_i,v_0) + f[v_i,\{v1,v2,v3\}] : i \text{ from } \{1,2,3\}\}$

- Possible values for s: {1,2}, {1,3}, {2,3}

$f(3,\{1,2\}) = ?$; $f(2,\{1,3\}) = ?$; $f(1,\{2,3\}) = ?$

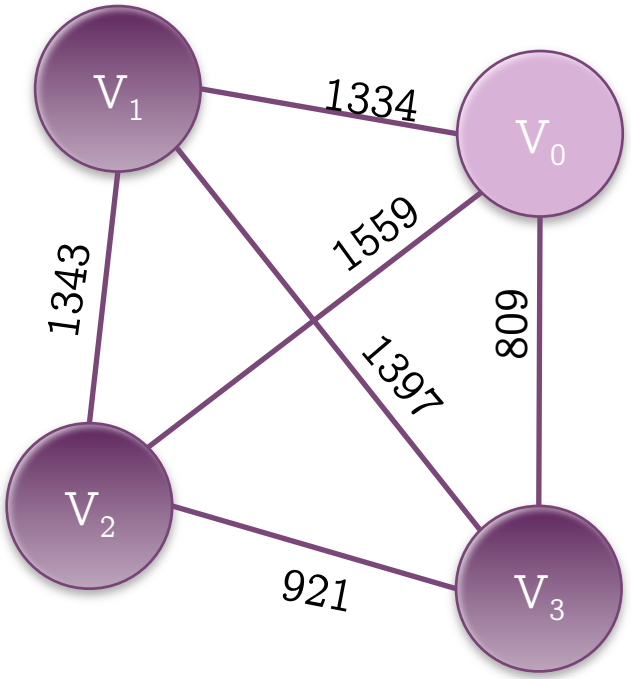
$f(3,\{1,2\}) = \min \{d(3,j) + f(j,s\backslash\{j\}): j \text{ in } \{1,2\} \}$

$= \min \{d(3,1) + f(1,\{2\}) , d(3,2) + f(2,\{1\})\}$

$= \min \{1397+2902,921+2677\}$

$= \min \{4299,3598\}$

$= 3598 , \quad N(3,\{1,2\})=\{2\}$



$|S| = 3$

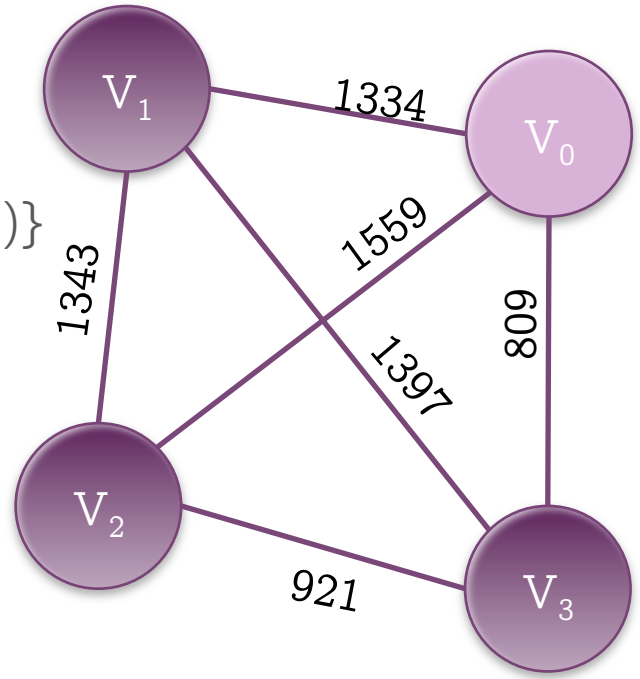
For i=1 to 3 do
    $f[v_i,s] = d(c_0,c_i)$
For j= 1 to 3 do
    for all $S \subseteq \{1,2,3\}$ with $|S| = j$ do
        $f[v_i, s] = \min\{F[S\backslash\{v_i\},v_k]+d(v_k,v_i): v_k \text{ from } S \backslash\{v_i\}\}$
Return $\min\{d(v_i,v_0) + f[v_i,\{v1,v2,v3\}] : i \text{ from } \{1,2,3\}\}$

$f(0,\{1,2,3\})=\min \{d(0,j) + f(j,\{1,2,3\}\backslash\{j\}: j \text{ in } \{1,2,3\} =$

$\min\{d(0,1)+f(1,\{2,3\}), d(0,2)+ f(2,\{1,3\}), d(0,3)+f(3,\{1,2\})\}$

$= \min\{1334+3073, 1559+3549, 809 + 3598\}$

$= \min\{4407,5108,4407\} = 4407, N(0,\{1,2,3\})=\{1,3\}$



Complexity

- $OPT[S,c_i] = \min\{OPT[S\backslash\{c_i\},c_j]+d(c_j,c_i): c_j \in S\backslash\{c_i\}\}$.

$|S| = k$, each step is $O(k^2)$

Why?

-Choosing c_i (k possible), and for each finding the preceeding best c_j.

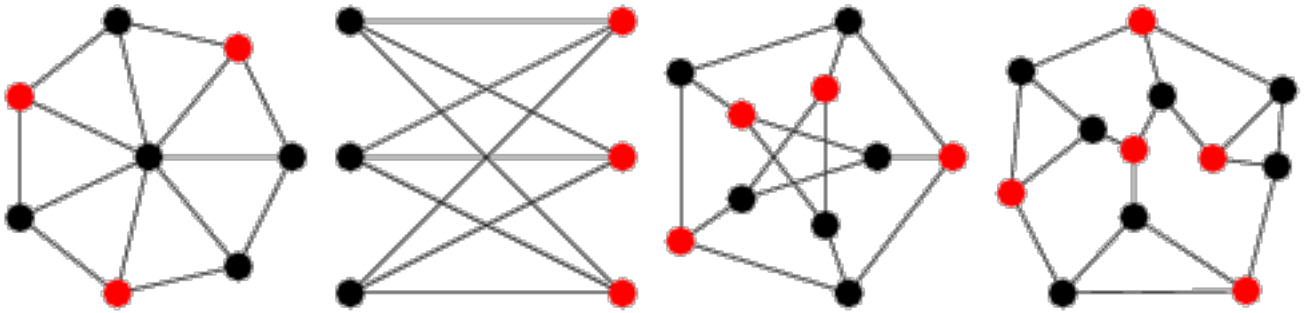
How many subsets are there?

For given k - $\binom{n}{k}$, Over all k's $\sum_{k=1}^{n-1}\binom{n}{k}$

Together: $\sum_{k=1}^{n-1}\binom{n}{k}k^2 = O*(2^n)$

Subset: Maximum Independent Set

- Given: undirected graph G=(V,E)
- Find the largest subset of I $\in$ V s.t any pair of vertices in I is non-adjacent
- Naïve solution: try all 2^n possible subsets

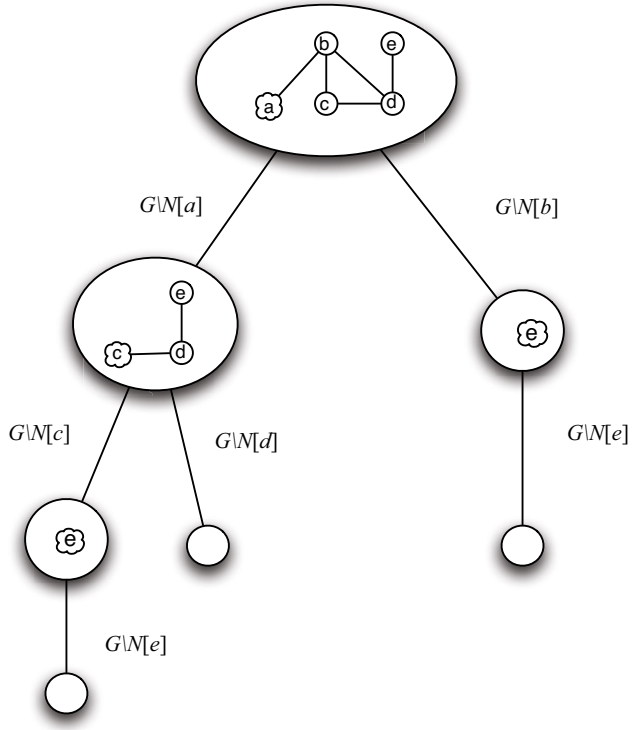


Improving O*(2^n)

- If v is in I, none of its neighbors are in I.
- If v is not in I, at least one of its neighbors is, but then none of its neighbors are in I.

Algorithm `mis1`(G).
Input: Graph $G = (V, E)$.
Output: The maximum cardinality of an independent set of G.

if $|V| = 0$ **then**
  return 0
choose a vertex v of minimum degree in G
return $1 + \max\{\texttt{mis1}(G \setminus N[y]) : y \in N[v]\}$



Recursion tree

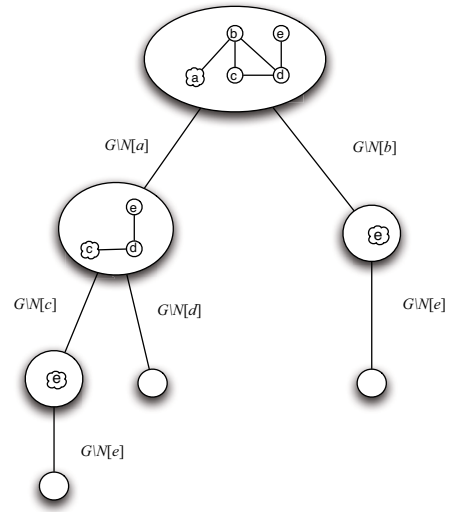
For a node v with d(v) children

If v is in, none of his neighbors are

$$T(n) \leq 1 + T(n - d(v) - 1) + \sum_{i=1}^{d(v)} T(n - d(v_i) - 1).$$

The solution with each of the rest.
$d(v_i)$ the degree of the selected node.
$1 \leq i \leq d(v)$

We chose the a node with minimal degree,
so $d(v) \leq d(v_i)$



Complexity analysis - 1

$$T(n) \leq 1 + T(n - d(v) - 1) + \sum_{i=1}^{d(v)} T(n - d(v_i) - 1).$$

$d(v) \leq d(v_i) \rightarrow \quad n\text{-}d(v_i)\text{-}1 \leq \; n\text{-}\; d(v)\text{-}1$

T is a monotonically increasing $\rightarrow$ $\quad T(n\text{-}d(v_i)\text{-}1) \leq T(\; n\text{-}\; d(v)\text{-}1)$

$$T(n) \leq 1 + T(n - d(v) - 1) + \sum_{i=1}^{d(v)} T(n - d(v) - 1)$$

In other words,

$$T(n) \; \leq 1 + (d(v) + 1) \cdot T(n - d(v) - 1).$$

Finally…

$$T(n) \leq 1 + (d(v)+1) \cdot T(n - d(v) - 1).$$

We assign s=d(v)+1 and get:

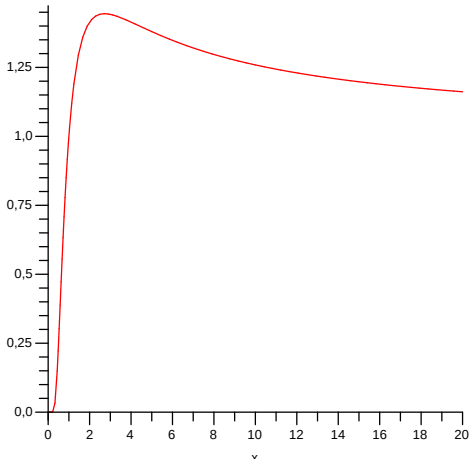
$$T(n) \leq 1 + s \cdot T(n-s) \leq 1 + s + s^2 + \cdots + s^{n/s}$$

Geometric series bounded $O^*(S^{n/s})$

Written differently: $s^{n/s} = (s^{1/s})^n$

The function $s^{1/s}$ peaks at s=3, thus:

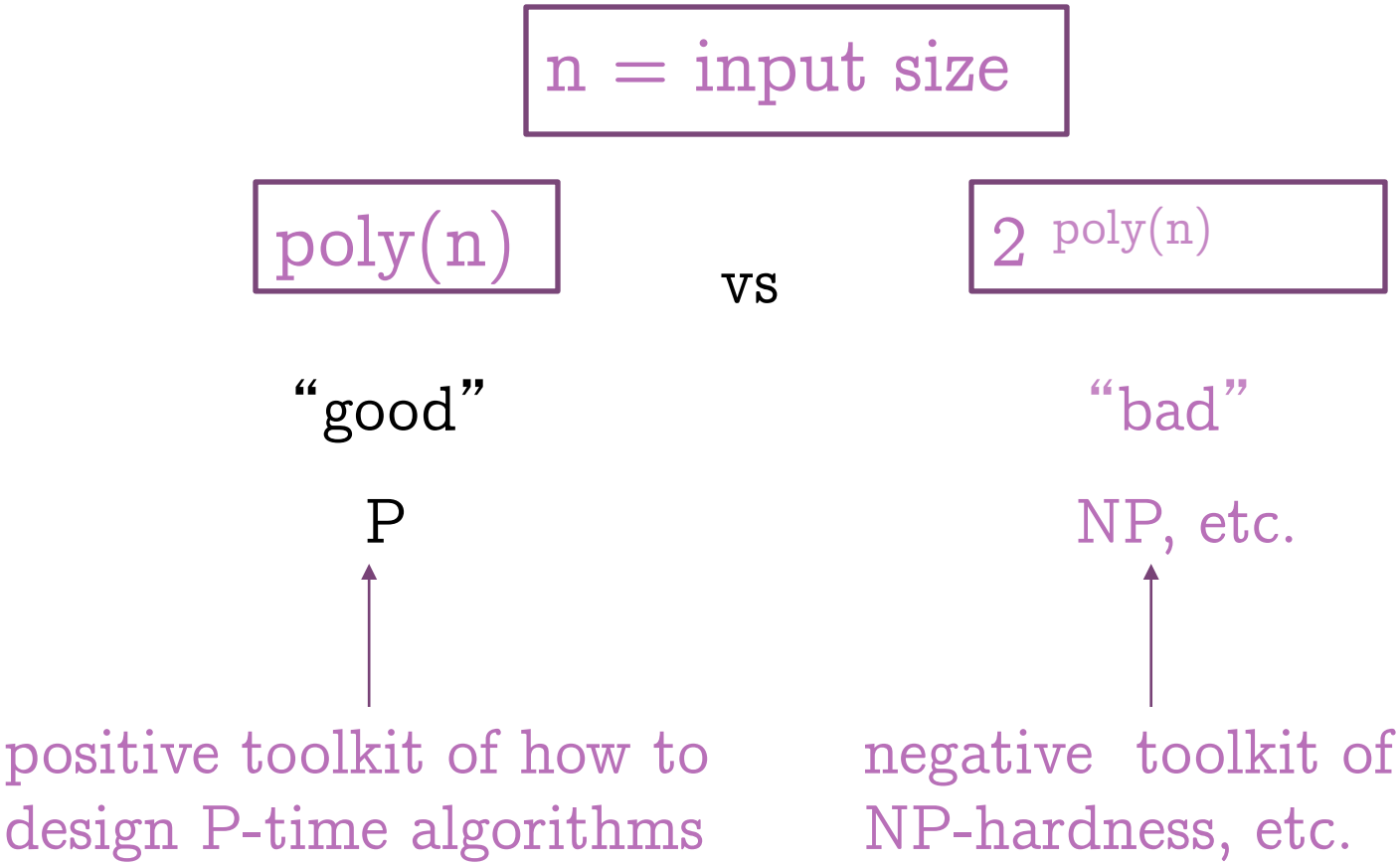
$T(n) = O^*(3^{n/3})$



Part 2- FPT

- By the end of this section you will be able to:
  - Define what is an FPT algorithm
  - Explain how to get a k^2 kernel for vertex cover
  - Explain how to get a k^2 kernel for max-satisfiability
  - Understand what is kernelization and what is its connection with FPT

The "classical" P vs. NP framework is one-dimensional



Unfortunately, almost everything turns out to be NP-hard.

The parameterized framework is two-dimensional

n = input size

k = a relevant secondary measure

$f(k)n^c$ vs $n^{g(k)}$

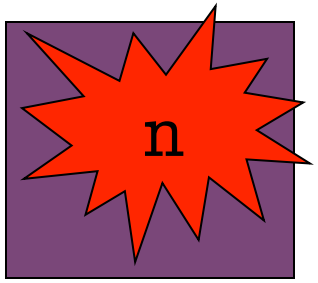
"good" "bad"

FPT Not discussed in this lecture

Framework in pictures

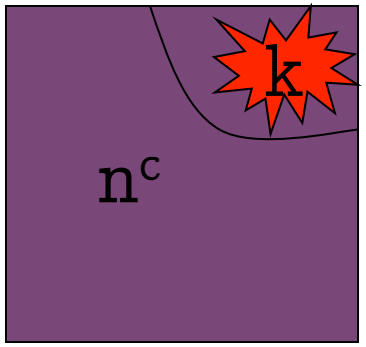
The *classical* P vs NP framework

Intrinsic Combinatorial explosion: Most problems are NP-hard or worse.



The *parameterized* framework

FPT



Try to confine the explosion to the parameter.

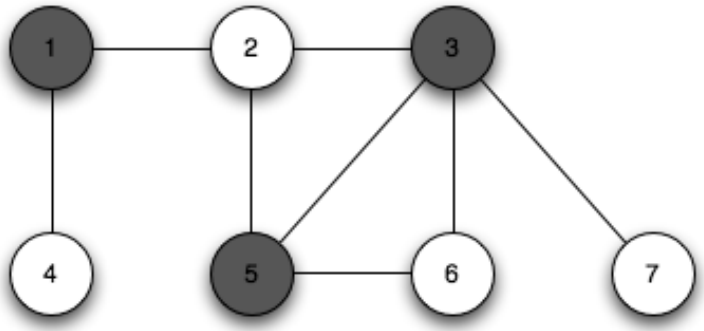
Vertex cover

- An NP-complete problem
- Easily parameterized (by the size of the set)

Vertex-cover(G,k)

Input: An undirected graph G = (V, E) and a nonnegative integer k.

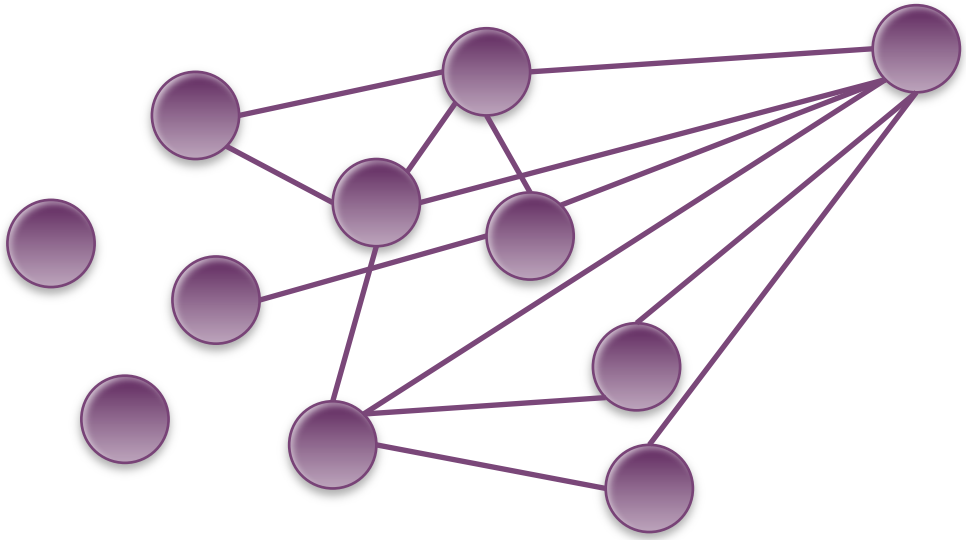
Task: Find a subset of vertices C ⊆ V with k or fewer vertices such that each edge in E has at least one of its endpoints in C.



Any other parameters?

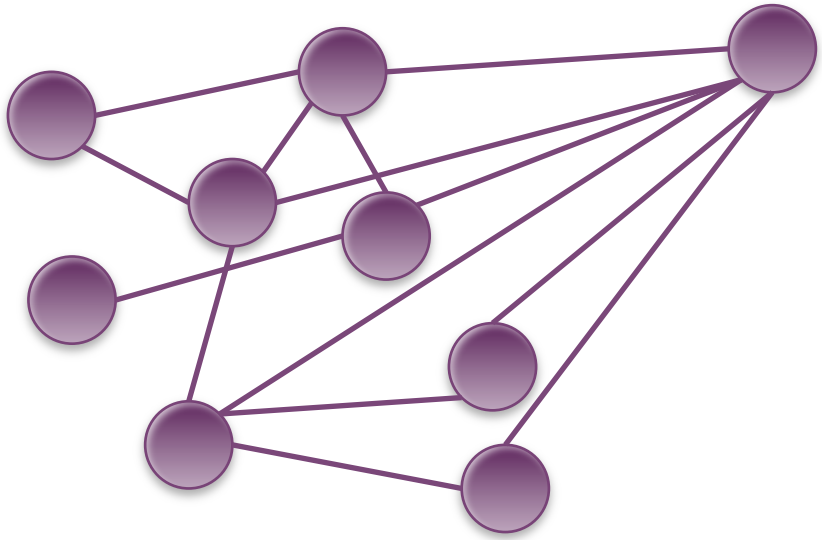
Vertex cover: Observations

- First, all vertices without edges connected are discarded.



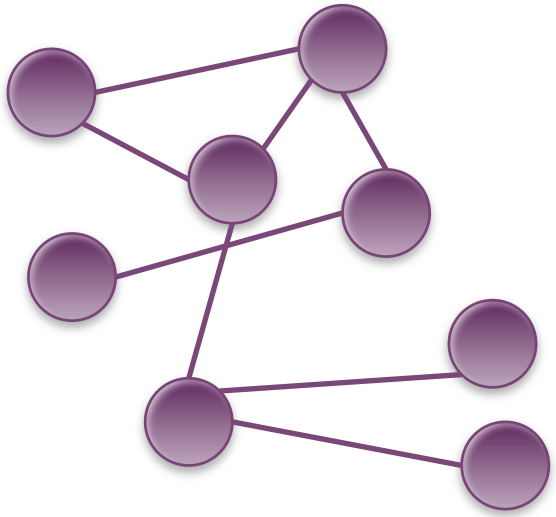
Vertex cover: Observation 1

- If v has degree at least $k+1$, then v belongs to each vertex cover in G of size at most k.
  - If v is not in the vertex cover, then all its neighbors are in the vertex cover.



Vertex cover: Observation 2

- (After applying observation 1) We are left with vertices with degree at most k
- Every edge needs to be covered, but every vertex has at most k neighbors.
  - Thus, if we have more than k^2 edges, no Vertex cover is possible



We are left with

- At most k^2 edges and at most $2k^2$ vertices are left (otherwise we can safely reject the instance)

Iterating on the vertices in the kernel we can achieve a running time of :

$$O^*(2^{2k^2})$$

- Exponential explosion ? Yes, but only w.r.t k and not n.

Definitions

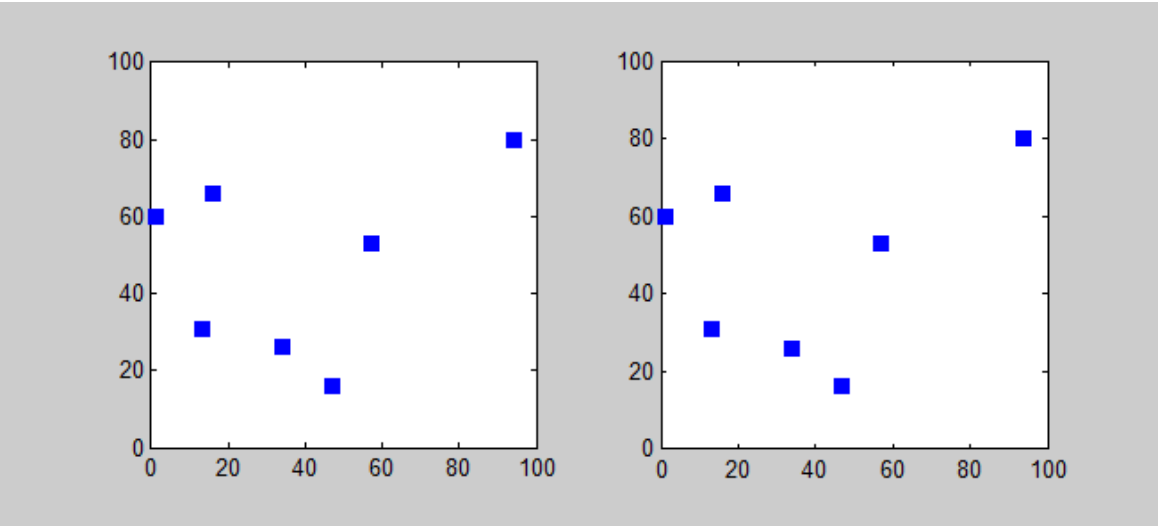
- A *parameterized* problem is a language

$L \subseteq \Sigma^* \times \Sigma^*$, where Σ is a finite alphabet. The second component is called the *parameter* of the problem.

- A parameterized problem is *fixed-parameter tractable* if it can be determined in $f(k) \cdot |x|^{O(1)}$ time whether $(x, k) \in L$, where f is a computable function only depending on k. The corresponding complexity class is called FPT.

TSP is also FPT ...

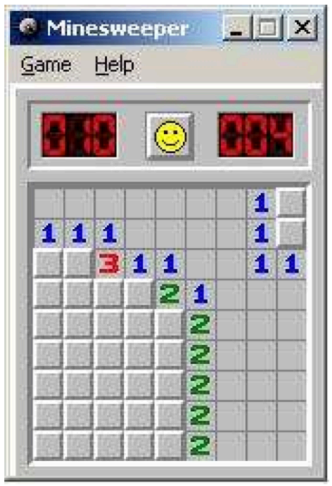
- for k = number of cities by "try all permutations" implies a - k! n algorithm
- A parameterized problem can be "trivially FPT"
- As before, "better FPT" $2^k n$ *is desirable*



Kernelization

- Reduce starting instance to $f(k)$ size instance, in polynomial time
- Use any algorithm to solve problem on kernel
- Total time:

$$p(n) + g(f(k))$$



Formal definition of kernelization

- Let L be a parameterized problem, that is, L consists of (I, k), where I is the problem instance and k is the parameter.
- *Reduction to a problem kernel:* replace instance (I, k) by a "reduced" instance (I', k') (called *problem kernel*) such that:
  1. $k' \leq k$, $|I'| \leq g(k)$ for **some** function g only depending on k
  2. $(I,k) \in L$ iff $(I',k') \in L$
  3. The reduction from (I, k) to (I', k') has to be **computable in polynomial time**.

Kernelization for Vertex Cover

H = G; (S = ∅ ;)

While there is a vertex v in H of degree at least k+1 do:

Remove v and its incident edges from H

k = k − 1; (S = S + v ;)

If k < 0 then return false

If H has at least k^2+1 edges, then return false

Remove vertices of degree 0

Solve vertex cover on (H,k) with some algorithm

CNF, a reminder

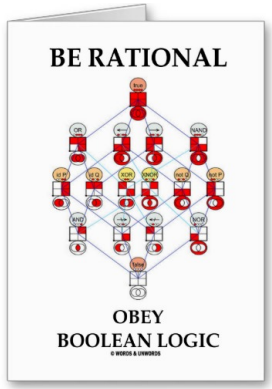
- A formula is in conjunctive normal form (CNF) if it is a conjunction of **clauses**, where a clause is a disjunction of **literals**

- Example :

$$(x1 \vee x2 \vee \neg x3) \wedge (x1 \vee x2 \vee x3) \wedge (\neg x1 \vee x2 \vee x3) \wedge (x2)$$

The size of the instance is the number of literals in total

In this example : 10



MAXIMUM SATISFIABILITY

- Input: A boolean formula F in conjunctive normal form consisting of **m clauses** and a nonnegative integer k.
- Task: Find a truth assignment satisfying at least k clauses.

$(x1 \vee \neg x2 \vee x3) \wedge (x1 \vee x2 \vee x3) \wedge (x1 \vee x2 \vee x3) \wedge (x2)$

Kernel of size k^2

Observation 1:

- If $k \leq \lceil m/2 \rceil$, then the desired truth assignment trivially exists:
  - Take a random truth assignment.
  - If it satisfies at least k clauses then we are done.
  - Otherwise, flip the assignment

$(x1 \vee \neg x2 \vee x3) \wedge (\neg x1 \vee \neg x2 \vee \neg x3) \wedge (x1 \vee x2 \vee x3) \wedge (x1) \wedge (x1 \vee \neg x2)$

Class ex.:

1. Find an assignment that satisfies at most 2 clauses
2. Show that by flipping the assignment, 3 or more clauses are satisfied

We can assume $k > \lceil m/2 \rceil \rightarrow m < 2k$

Flipping : Class ex.

$(x1 \vee \neg x2 \vee x3) \wedge (\neg x1 \vee \neg x2 \vee \neg x3) \wedge (x1 \vee x2 \vee x3) \wedge (x1) \wedge (x1 \vee \neg x2)$

x1=F x2=T x3=F

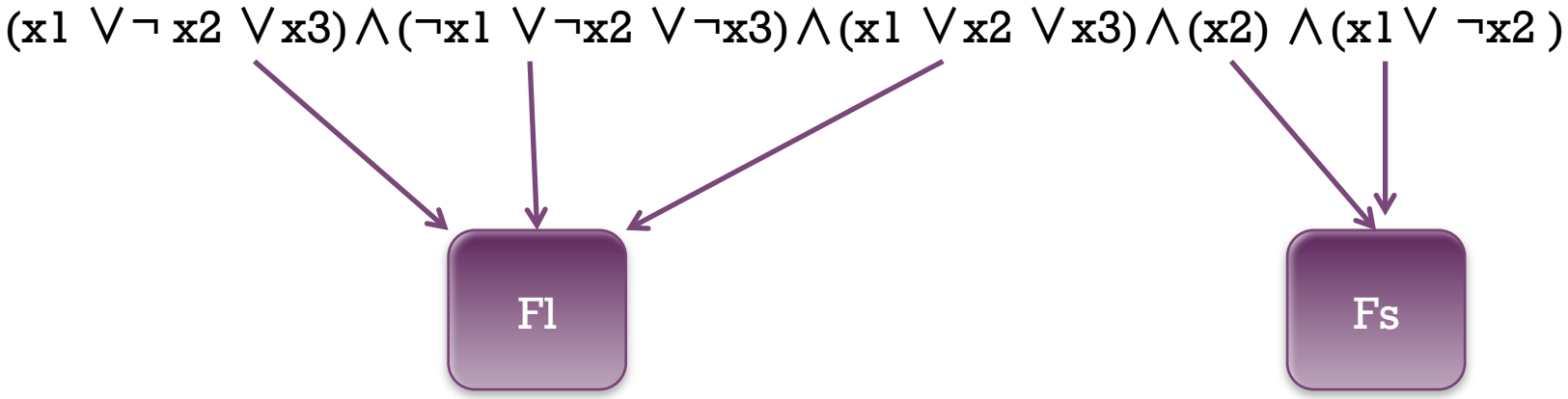
1,4,5 are false 2 and 3 are true. Let's fliip it!

X1=T x2=F x3=T

Without even looking at 2 or 3, we know that 1,4,5 are now satisfied

Observation 2:

- Partition the clauses of F into Fl and Fs:
  Fl: long clauses containing at least k literals;
  Fs: short clauses containing less than k literals.



Observation 2 (cont.)

- Let L := number of long clauses (|Fl|) .
- If L ≥ k, then at least k clauses can be satisfied.
  - for each (w.c) pick one literal and set it to satisfy the formula

(x1 ∨¬ x2 ∨x3 ∨ … ∨ x_k ∨ … ∨ x_{ci})

Fl

Fs

Now we are left with the case where L < k

Example

- (x1 ∨¬ x2 ∨x3)∧(¬x1 ∨¬x2 ∨¬x3)∧(x1 ∨x2 ∨x3)∧(x2) ∧(x1∨ ¬x2)
- The 3 members of Fl are the first 3.
- We can set x1 as the first clause demands (T), x2 for the second (F) and x3 for the third (T)
- We are now secure that at least k (3) clauses are satisfied

Observation 3:

- By now we already know that the instance has $L < k$ large clauses and the total number of clauses m is strictly smaller than $2k$
- (F, k) is a yes-instance **if and only if** $(F_s, k - L)$ is a yes-instance.

Observation 3:

- (F, k) is a yes-instance **if and only if** (Fs, k − L) is a yes-instance.

  ->if (F,K) has an assignment, there are between 0 and L large clauses satisfied but certainly no less than k-L small clauses.

  Thus, if there is an assignment for (F, k) there must also be one for (Fs, k − L)

Observation 3:

- (F, k) is a yes-instance **if and only if** (Fs, k − L) is a yes-instance.

  <- if (Fs , k − L) is a yes-instance of MaxSat:

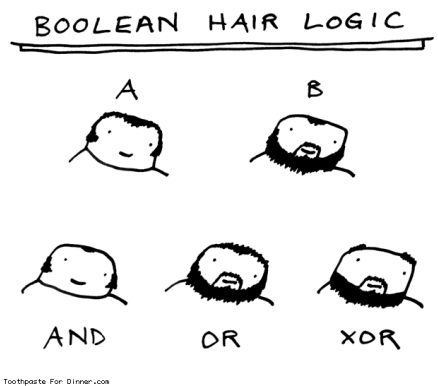
  1. To satisfy k − L clauses at most k − L variables (at most one variable per satisfied clause) are needed to be assigned, leaving L variables unassigned.

  2. we can use the trick before and satisfy all the large clauses and get (F, k) of MaxSat

Conclusion

- $|Fs| = O(k^2)$ because :
  - There are $m - L \leq m$ small clauses, each containing at most k literals
  - $m < 2k$
  - The total number of literal occurrences in Fs is bounded from above by $2k \cdot k = 2k^2$.

- MAXIMUM SATISFIABILITY has a problem kernel of size $O(k^2)$



Recall: Formal definition of Kernel

- Let L be a parameterized problem, that is, L consists of (I, k), where I is the problem instance and k is the parameter.
- *Reduction to a problem kernel:* replace instance (I, k) by a "reduced" instance (I', k') (called *problem kernel*) such that:
  1. $k' \leq k$, $|I'| \leq g(k)$ for **some** function g only depending on k
  2. $(I,k) \in L$ iff $(I',k') \in L$
  3. The reduction from (I, k) to (I', k') has to be **computable in polynomial time**.

The connection: Kernels and FPT

- **Theorem**: Consider a decidable parameterized problem. Then the problem belongs to FPT, **if and only if** it has a kernel

<= Build the kernel and then solve the problem on the kernel

=> A little trickier ...

FPT -> Kernel

⇒ assume that the given fixed-parameter algorithm has running time $f(k) \cdot n^c$.

- Run this algorithm on the problem for at most n^{c+1} steps
- two cases are possible:
  1. The algorithm has finished its task within that time, now we have a kernelization algorithm running in polynomial time n^{c+1}, which simply outputs either a trivial "no"-instance or a trivial "yes"-instance.
  2. The algorithm is not finished, but we can argue that $n < f(k)$. Thus, our problem kernel is the original input instance itself.

Tips and tricks for selecting the parameter

- Parameter really small
  - Example: Vertex Cover on planar graphs
- Guaranteed parameter value
  - Example: Independent set on planar graphs
- More than one obvious parameterization
  - Example :Weighted vertex cover
- Close to trivial problem instances
  - Example :Traveling Salesperson problem in the two-dimensional Euclidean plane