Tecniche Algoritmiche per Grafi e Reti

Algoritmi parametrizzati

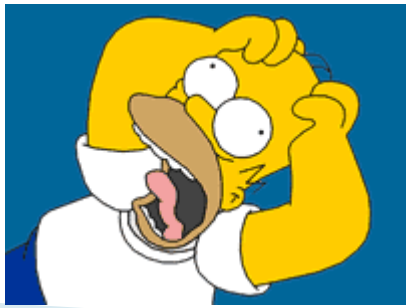
Patrizio Angelini

Dipartimento di Informatica e Automazione
Università degli Studi Roma Tre

A world full of NP-Hard problems

- Problem: I want to travel among a set of cities driving the fewest possible number of KMs.
  - IT'S DIFFICULT!!

- Problem: I want to put a set of objects into some bags, knowing that each bag can not afford more than 10 KGs and trying to use as few bags as possible.
  - IT'S DIFFICULT!!



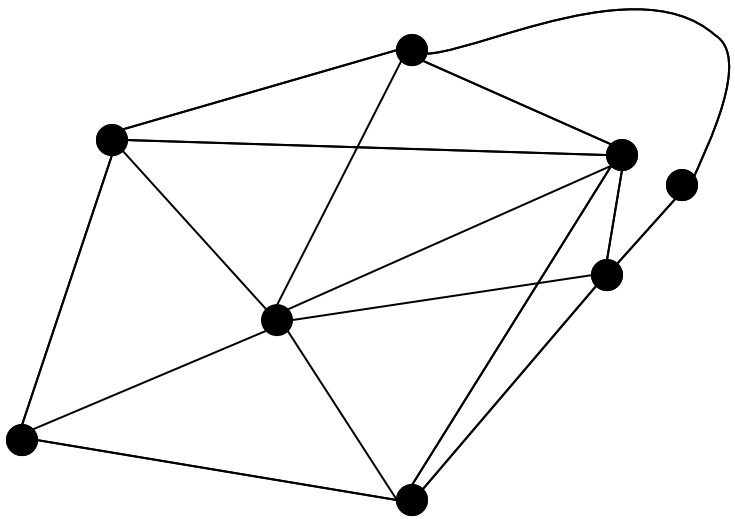
How to deal with NP-Hard problems?

- **Exact Algorithms**
  - Esatti ma lenti. Solo istanze piccole
- **Approximation Algorithms**
  - Veloci ma non esatti, errore limitato
- **Fixed-Parameter Algorithms**
  - Se un certo parametro è costante, problema polinomiale
- **Randomized Algorithms**
  - Non-determinismo
  - Probabilmente esatti; veloci e probabilmente non molto sbagliati; oppure esatti e probabilmente non troppo lenti

Fixed-Parameter Algorithms

Fixed-Parameter Algorithms

- Problem (optimization): Given a non-planar graph *G*, what is the minimum number of vertices to delete in order to make *G* planar?



NP-hard ☹

[Lewis-Yannakakis]

Fixed-Parameter Algorithms

- Problem (decision): Given a non-planar graph G and an integer k, is it possible to make G planar by deleting k vertices?

Polynomial ☺

- For every set of k vertices, remove such vertices from G and test the planarity of the resulting graph.
  - There are $O(n^k)$ such sets.
  - Testing planarity requires $O(n)$ time.
  - $T(n,k)=O(n^k)\ O(n) = \mathbf{O(n^{k+1})}$

Where is the trick?

Polynomial ☺

Where is the trick?

In the decision version, *k* is a *constant* parameter that is part of the input.

$\boldsymbol{O(n^{k+1})}$ is polynomial… ☺

But the algorithm is still very slow ☹

Fixed-Parameter Tractability

A problem is *Fixed-Parameter Tractable*

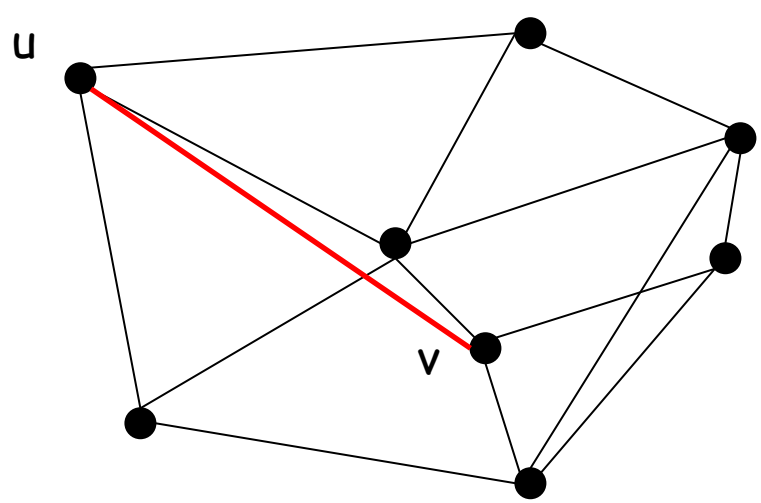
if it can be solved in

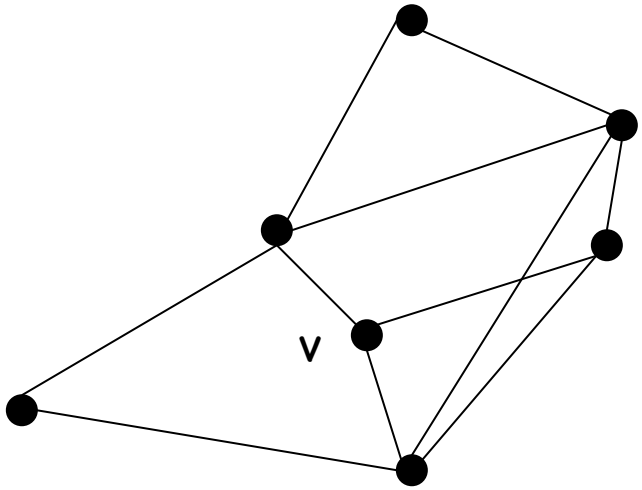
$O(f(k)\, n^{O(1)})$ time, where

- k is a parameter of the problem,
- $f()$ is an arbitrary function, and
- $n^{O(1)}$ is a polynomial not depending on k.

Vertex Cover

- Theorem [Melhorn]: There exists a $O(2^k\ n)$-time algorithm for Vertex Cover
- For each edge (u,v), at least one out of u and v belongs to any solution S.

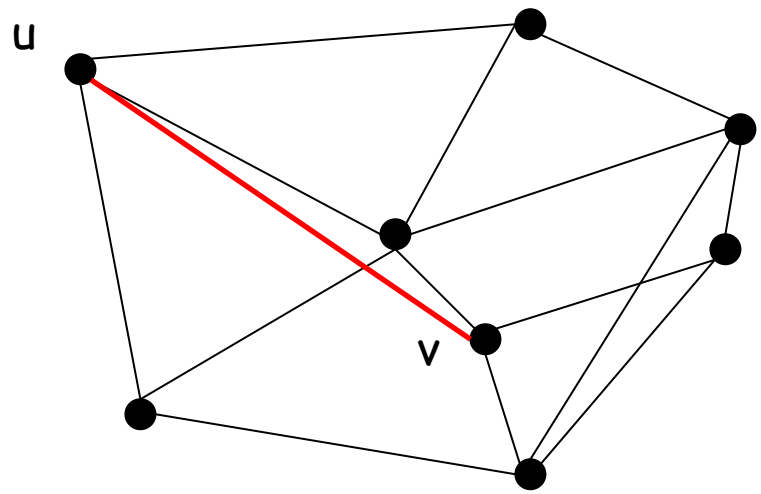


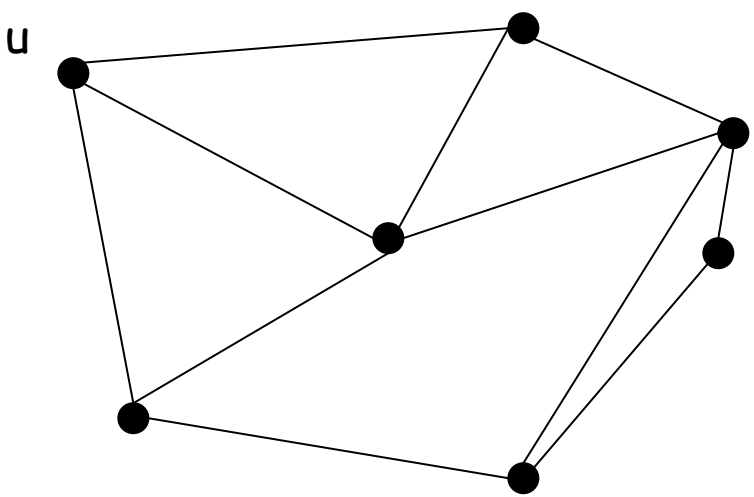


Add u to S and solve instance $(G-\{u\}, k-1)$

Vertex Cover

- Theorem [Melhorn]: There exists a $O(2^k\ n)$-time algorithm for Vertex Cover
- For each edge (u,v), at least one out of u and v belongs to any solution S.





Add v to S and solve instance $(G-\{v\}, k-1)$

Vertex Cover – Time Complexity

- $T(n,0) = O(n)$
- $T(n,k) = 2\,T(n-1, k-1) + O(n) \leq \quad 2\,T(n, k-1) + O(n) \leq$

$$2\,(2\,T(n, k-2) + O(n)) + O(n) \leq$$

$$2\,(2\,(2\,T(n, k-3) + O(n)) + O(n)) + O(n) \leq$$

$$2\,(2\,(2\,(\ldots(2\,T(n, 0) + O(n)) + \ldots + O(n)) + O(n)) + O(n) \leq$$

$$2^k\,O(n) + (2^{k-1} + 2^{k-2} + 2^{k-3} + \ldots + 1)\,O(n) =$$

$$2^k\,O(n) + (2^k - 1)\,O(n) = O(2^k\,n)$$

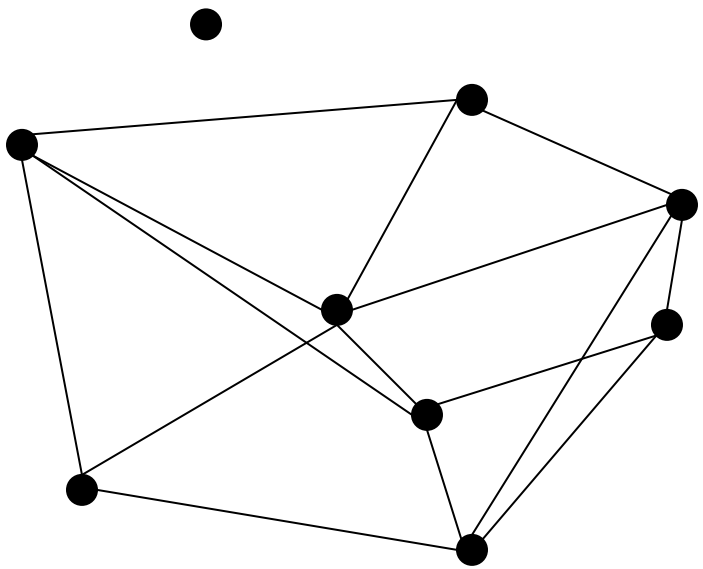
Reduction Rules

- A *rule* (a polynomial-time algorithm) that transforms an instance (I,k) into an equivalent and simpler instance (I',k')

- Equivalent:
  - (I,k) is a YES instance if and only if (I',k') is a YES instance
- Simpler:
  - $|I'| < |I|$
  - I' has fewer occurrences of a particular substructure
  - $k' < k$

Reduction Rules – Vertex Cover

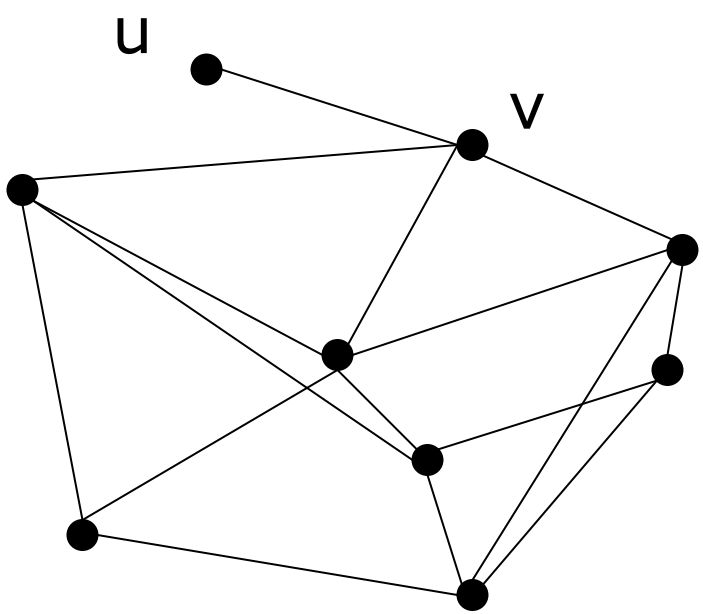
- If *G* has a vertex of degree *0*, then remove it



Reduction Rules – Vertex Cover

- If *G* has a vertex *u* of degree *1*,

  then add its adjacent vertex *v* to the solution and remove *u* from *G*



Reduction Rules – Vertex Cover

- If neither of the two rules can be applied, then every vertex has degree at least 2

- Same algorithm as before *(Melhorn)* but*:*
  - add u to S and solve $(G-\{u\}, k-1)$
  - add the neighbors $N(u)$ of u to S and solve $(G-N(u), k-|N(u)|)$

RR – VC – Time Complexity

- Since $|N(u)| \geq 2$,
  - $T(n,k) \leq T(n,k-1) + T(n,k-2) + O(n^2)$.
    - Fibonacci series: every number is the sum of the previous two: $x_k = x_{k-1} + x_{k-2} + c$.
    - The *k-th* number of the Fibonacci series tends to the *golden ratio* to the power of *k*.
    - $T(n,k) = \Phi^k O(n^2) = ((1+\sqrt{5})/2)^k O(n^2) = O(1.6181^k n^2)$.

Kernelization

- Sometimes, the reduction rules work till the size of the problem is reduced really a lot.

- After the reduction rules have been applied, the problem has size *g(k)*, where *g()* is a function only depending on *k*.

- Then, the problem can be solved by using an exact algorithm, even brute-force, on the reduced instance

FPT – Complexity classes

- W[1]: no FPT-algorithm possible
  - Graph Coloring
- FPT-reduction: the reduction has to be performed with an FPT-algorithm
- no easy correspondence between these parameterized complexity classes and NP- or PSPACE-classes