Exact Algorithms for NP-hard problems

Nicolas Bourgeois (ESSEC)

24 mai 2012

Why the **P**-border ?

1. Practical reasons (**Jack Edmonds, 1965**)

   For practical purposes the difference between algebraic and exponential order is more crucial than the difference between [computable and not computable]. . .

2. Algebric reasons (**Alan Cobham, 1964**)

   If we formalize the definition relative to various general classes of computing machines we seem always to end up with the same well-defined class of functions. Thus we can give a mathematical characterization of P having some confidence it characterizes correctly our informally defined class. This class then turns out to have several natural closure properties

Why the **P**-border ?

1 Practical reasons (**Jack Edmonds, 1965**)

For practical purposes the difference between algebraic and exponential order is more crucial than the difference between [computable and not computable]. . .

2 Algebric reasons (**Alan Cobham, 1964**)

If we formalize the definition relative to various general classes of computing machines we seem always to end up with the same well-defined class of functions. Thus we can give a mathematical characterization of P having some confidence it characterizes correctly our informally defined class. This class then turns out to have several natural closure properties

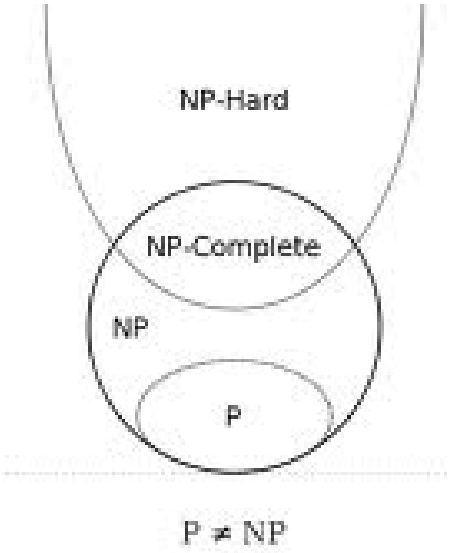
P versus NP

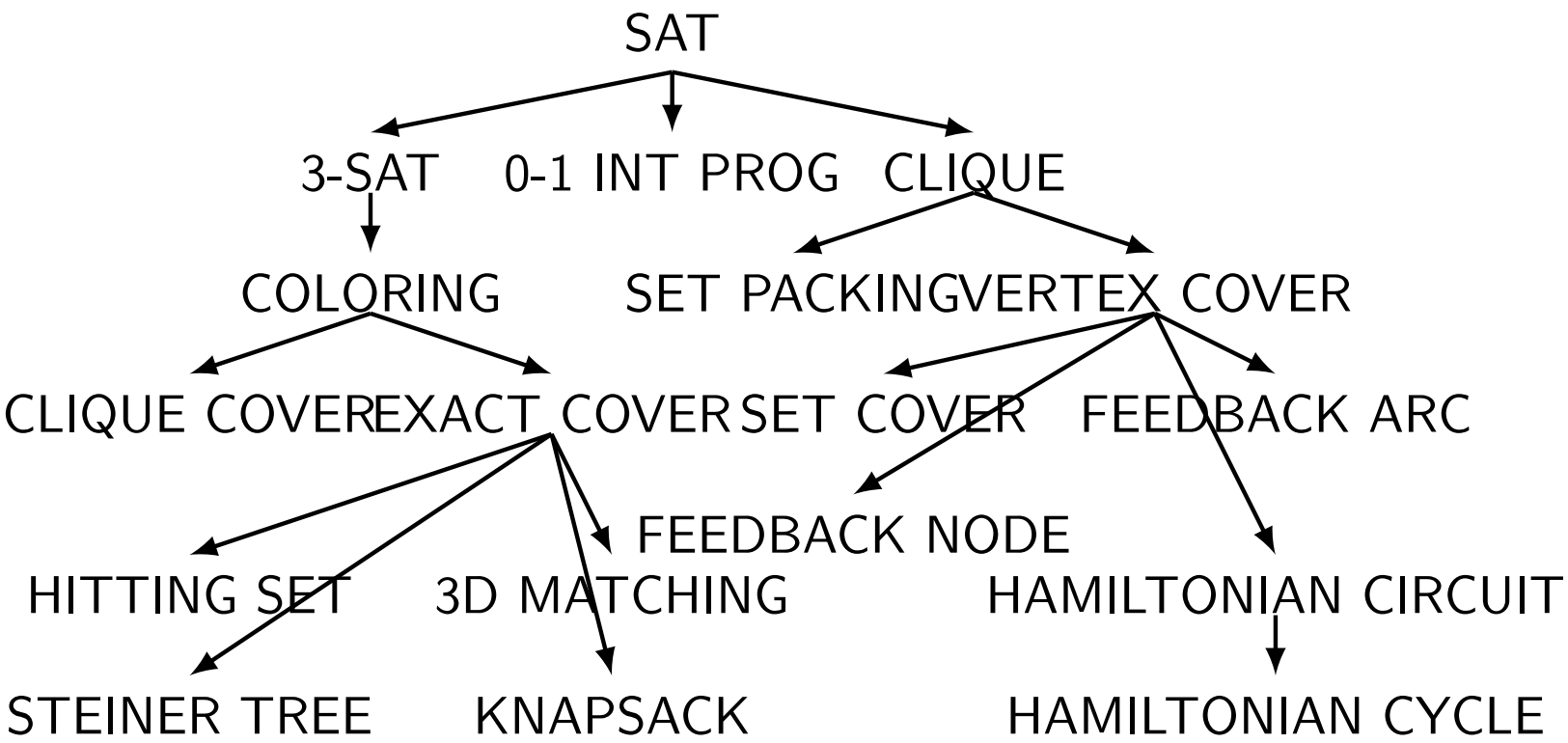
P — *There exists an algorithm who finds a solution to the problem and whose execution time is bounded with $O(n^d)$.*

NP — *There exists an algorithm who verifies a solution to the problem and whose execution time is bounded with $O(n^d)$.*

NP-hard — *If this problem is polynomial, then P=NP.*

P versus NP




SAT
3-SAT
0-1 INT PROG
CLIQUE
COLORING
SET PACKING
VERTEX COVER
CLIQUE COVER
EXACT COVER
SET COVER
FEEDBACK ARC
FEEDBACK NODE
HITTING SET
3D MATCHING
HAMILTONIAN CIRCUIT
STEINER TREE
KNAPSACK
HAMILTONIAN CYCLE

Example : matching problems

Matching : set of non-adjacent edges

Proposition

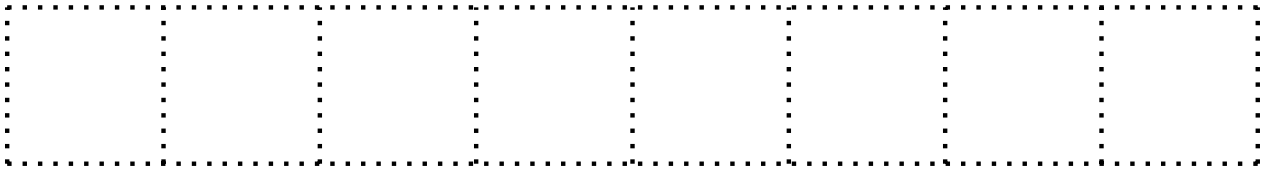
*Maximum-size matching is in **P***

Proposition

*Minimum-size maximal matching is **NP**-hard*

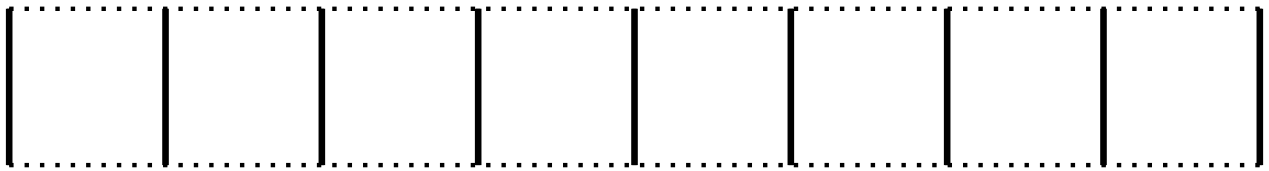
Exercise I

a) Find a maximum-size matching on a $(1 \times n)$-grid.
b) Find a minimum-size maximal matching on a $(1 \times n)$-grid.



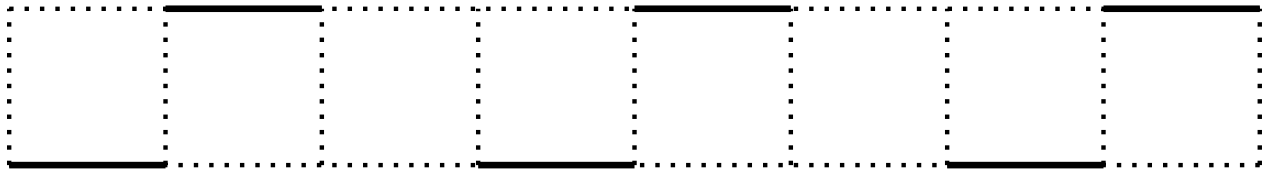
Solutions

a) Find a maximum-size matching on a $(1 \times n)$-grid.



Solutions

b) Find a minimum-size maximal matching on a $(1 \times n)$-grid.



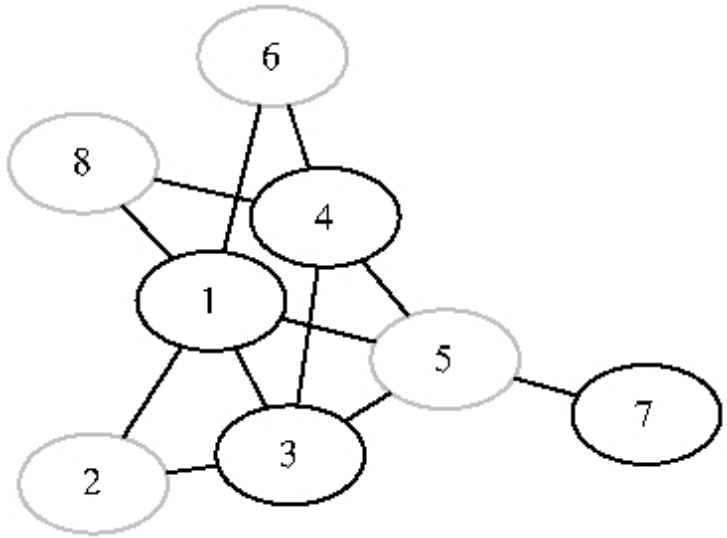
General Principle

- ★ Aim : maximize $f(S)$ among all the sets S that verifies some property.
- ★ Problem : this is NP-hard.
- ★ Solution : design an algorithm that computes a solution S' which is 'not too far' from the optimal.

A R-approximation algorithm guarantees that $f(OPT)/f(S') \leq R$

Exercise II

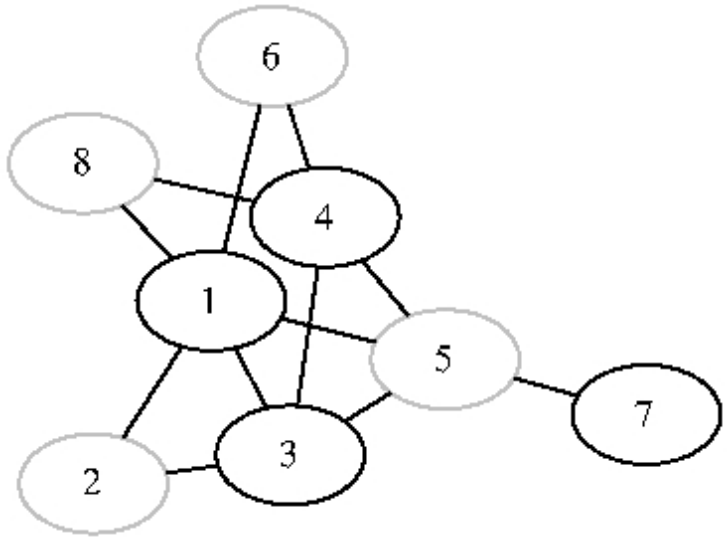
A Vertex Cover is a subgraph $H \subset G$ such that any edge in the initial graph G is incident to a vertex in the cover H.



Exercise II

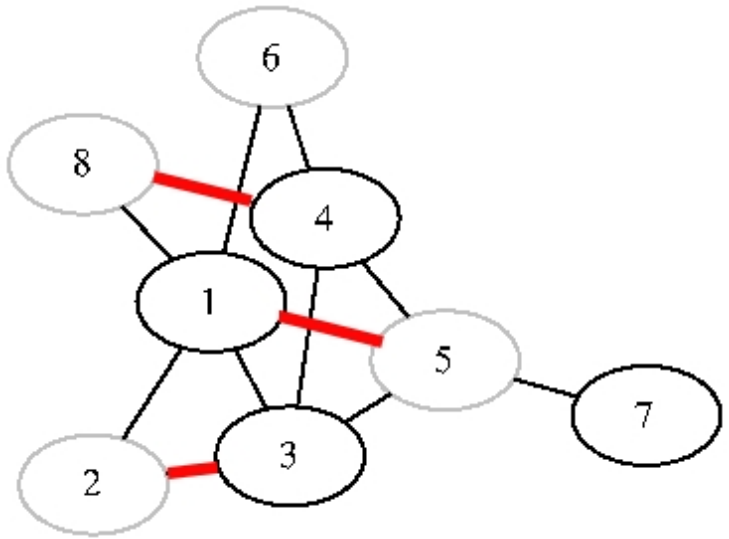
Question : How to make a 2-approximation algorithm for the vertex cover ?

Clue : Maximum Matching is in **P**.



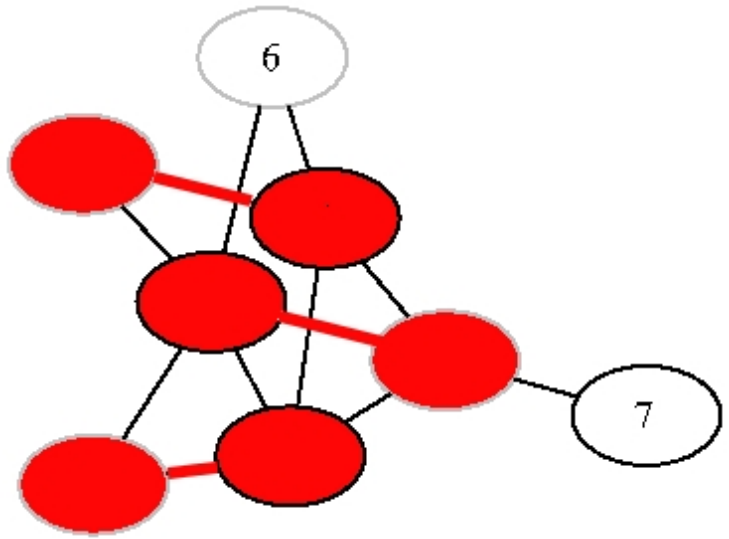
Exercise II

- ✓ any vertex cover has at least the same size of a maximal matching : $f(OPT) \geq M$
- ✓ all the vertices incident to a maximal matching form a vertex cover : $f(S') = 2M$



Exercise II

- ✓ any vertex cover has at least the same size of a maximal matching : $f(OPT) \geq M$
- ✓ all the vertices incident to a maximal matching form a vertex cover : $f(S') = 2M$



Inapproximation

Ratio 2 isn't very sexy, still it's among the best situations.

Many important problems don't admit constant approximation ratio except if **P=NP**.

Example : Min Coloring, Max Clique, Min Dominating Set, Traveling Salesman Problem...

Because we can

Jack Edmonds, 1965 (again) :
It would be unfortunate for any rigid criterion to inhibit the practical development of algorithms which are either not known or known not to conform nicely to the criterion. . .

Modern computers can compute $\sim 2^{50}$ operations in reasonable time.

TSP (Traveling Salesman Problem) : Given a complete graph with weighted edges, find a path of minimum cost that visits every vertex exactly once.

Proposition

There exists an algorithm that solves TSP within 2^n steps.

Assume we have an instance of 50 vertices :

Why would we use heuristics or approximation when we can do it exactly ?

TSP (Traveling Salesman Problem) : Given a complete graph with weighted edges, find a path of minimum cost that visits every vertex exactly once.

Proposition

There exists an algorithm that solves TSP within 2^n steps.

Assume we have an instance of 50 vertices :

Why would we use heuristics or approximation when we can do it exactly ?

Exponential algorithms can even be faster

Size of the instance	n^3	1.1^n
$n = 30$	27000	18
$n = 50$	125000	118
$n = 100$	10^6	13781
$n = 200$	8×10^6	2×10^8

Pause

Any question before we start the main course ?

Exercise : brute force algorithms

Design an exact (exponential) algorithm for the following problems :

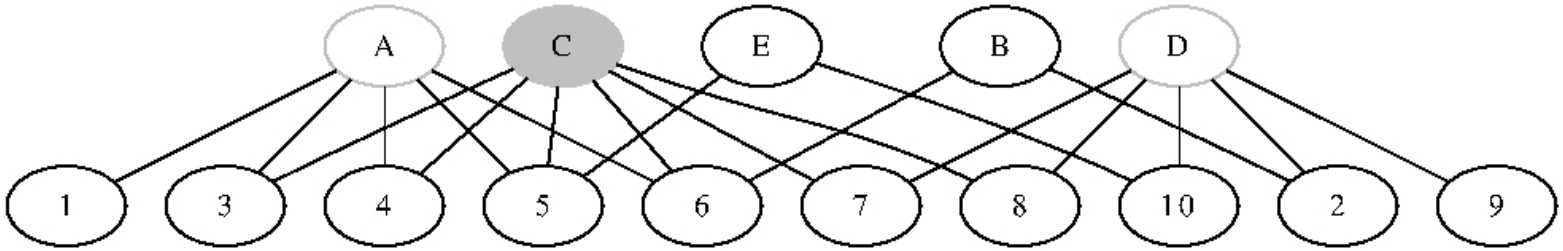
1. Minimum Vertex Cover (running time 2^n)
2. Traveling Salesman Problem (running time $n!$)

Solution

1. Compute every subgraph and keep the largest one which is a vertex cover
2. A path is just a way to order vertices : n possibilities for the first vertex, then $n-1$ for the second one, etc...

Example : Minimum Set Cover

Given a set S and a family of subsets $\mathcal{F} \subset 2^S$, find the smallest subfamily $X \subset \mathcal{F}$ that covers S.



Trivial algorithm in 2^m, with $m = |\mathcal{F}|$

Example : Minimum Set Cover

Recursive algorithm :

1. if all subsests have size two or less, the problem is polynomial.
2. otherwise, if some element x belong to only one set T, take T.
3. otherwise, if some set T contains only one element x, discard T.
4. otherwise, branch on some subset T of maximum carinality : either take it or discard it. Solve the reduce problem in both cases.

Example : Minimum Set Cover

Branching recurrence :

$$SOL(S, F) = \min(SOL(S \setminus T, \mathcal{F} \setminus \{T\}) + 1, SOL(S, \mathcal{F} \setminus \{T\}))$$

Bounding value :

$$|T| \geq 3$$

Branching inequation :

$$T(n+m) \leq T(n+m-1) + T(n+m-4)$$

Bound on complexity :

$$T(n+m) \leq 1.38^{n+m}$$

General Principle

Find a recurrence (branch) :

$$SOL(I) = f(SOL(I_1), SOL(I_2), \ldots)$$

with $|I_j| < |I|$

Find a stop condition (cut) :

$\phi(I)$ implies the problem is polynomial, and

$|I| < k \Longrightarrow \phi(I)$

General Principle

Translate the recurrence numerically :

$$T(n) \leq T(n-k_1) + T(n-k_2) + \dots$$

with $n = |I|, k_j = |I| - |I_j|$

We know that the solution is $T(n) = x^n$, where

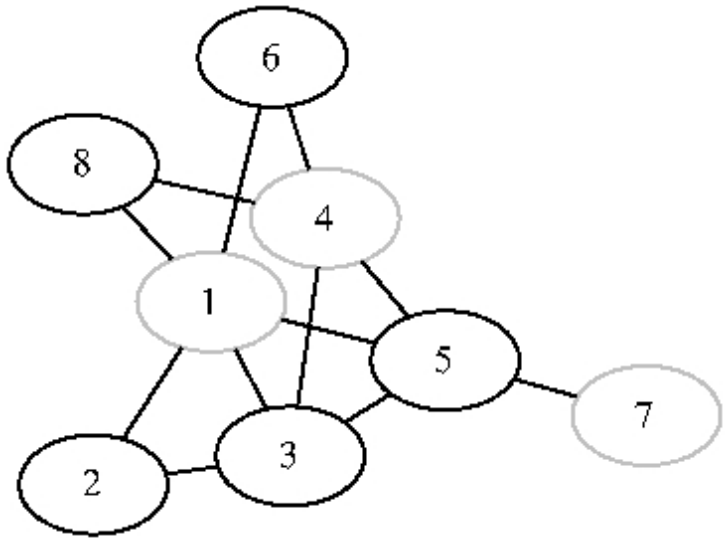
x is the solution of :

$$1 = x^{-k_1} + x^{-k_2} + \dots$$

Exercise : Minimum Independent Dominating Set

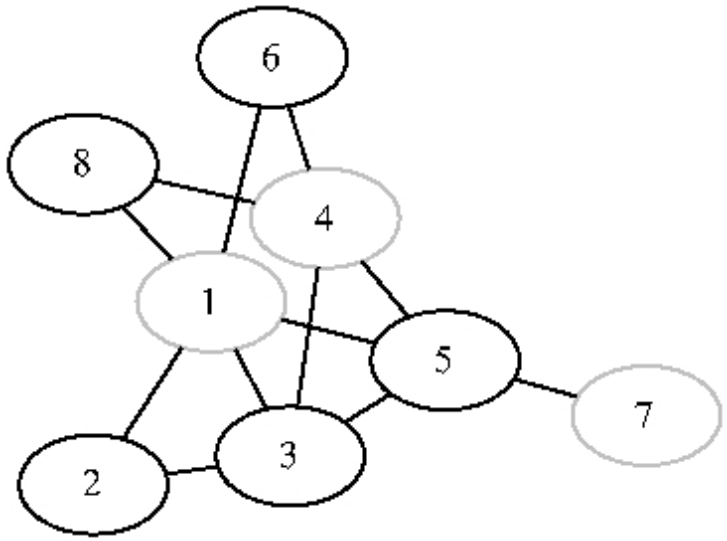
Let G a graph. An independent dominating set is a subgraph $H \subset G$ such that :

- ✓ H contains no edge.
- ✓ every edge in G is adjacent to a vertex in H.



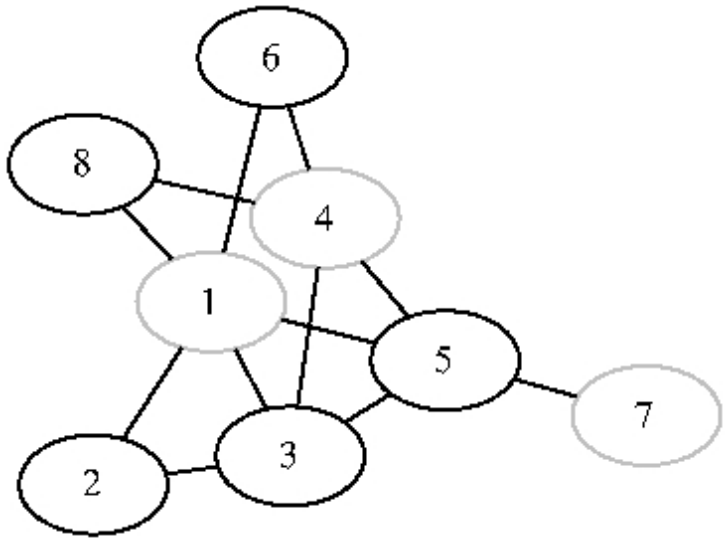
Exercise : Minimum Independent Dominating Set

1) find a recurrence algorithm whose recurrence equation is $T(n) \leq (k+1)T(n-k-1)$
2) which k is the worse case ?
3) what is the overall running time of this algorithm ?

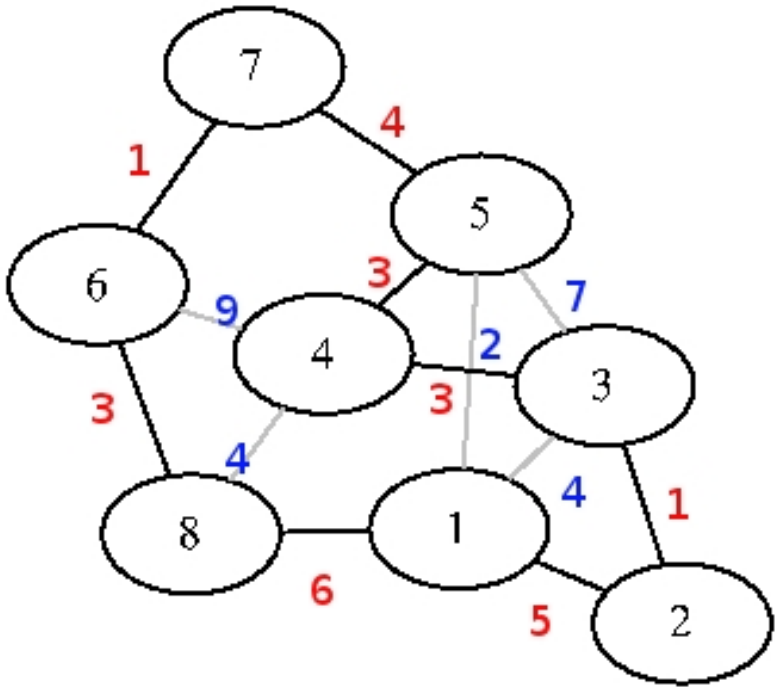


Exercise : Minimum Independent Dominating Set

1) Pick a vertex of minimum degree k : either add it or one of its neighbors. Discard the neighbors of the selected vertex.
2) Worse case is $k = 2$.
3) Overall complexity is $3^{n/3}$.



Example : TSP



Example : TSP

For any subgraph $W \subset V$ and any vertex $i \in W$, $P(W, i)$ is the shortest hamiltonian path in W between 1 and i.

Let $j \in W$ the vertex just before i in $P(W, i)$, we just need to compute $P(W \setminus \{i\}, j)$ for every $j \in W$.

$$P(W, i) = \min_{j \in W} \left(P(W \setminus \{i\}, j) + w(e_{ij}) \right)$$

Example : TSP

If we store results in a big table by increasing size of W, the recurrence is polynomial.

Thus, running time is equal to the size of the the table, which is 2^n.

General Principle

Find a recurrence (branch) :

$$SOL(I) = f(SOL(I_1), SOL(I_2), \ldots)$$

with $|I_j| < |I|$

Compute by increasing size and store all results in a table M.

Typically, $M = 2^I$.

General Principle

Advantages :

- ✓ The complexity is independent from the branching rule (provided it is polynomial).
- ✓ No subproblem is computed twice.
- ✓ Good for poor branchings (TSP, coloring . . .)

Weaknesses :

- —◦ Lots of useless subproblems are computed.
- —◦ Huge memory requirement (equal to the running time).
- —◦ Not efficient for subgraph mining, for example.

Inclusion-Exclusion

Reference : Björklund, Husfeldt and Koivisto, 2006.

$$c_k(V) = \sum_{X \subset V} (-1)^{|X|} a(X)^k$$

where $a(X)$ is the number of subset from some well-chosen family $\mathcal{F}$ disconnected with X.

$c_k(V)$ stands for the number of covers of V with exactly k items from $\mathcal{F}$.

Many cover or partition problems reduce to the computation of these coeffs (like, min coloring)

Inclusion-Exclusion

All the $c_k(V)$ can be computed within :

- ✓ 4^n with brute force.
- ✓ 3^n with dynamic programming.
- ✓ 2^n with Zeta transform.

Inclusion-Exclusion

All the $c_k(V)$ can be computed within :

- ✓ 4^n with brute force.
- ✓ 3^n with dynamic programming.
- ✓ 2^n with Zeta transform.

Inclusion-Exclusion

All the $c_k(V)$ can be computed within :

- ✓ 4^n with brute force.
- ✓ 3^n with dynamic programming.
- ✓ 2^n with Zeta transform.

Inclusion-Exclusion

All the $c_k(V)$ can be computed within :

- ✓ 4^n with brute force.
- ✓ 3^n with dynamic programming.
- ✓ 2^n with Zeta transform.

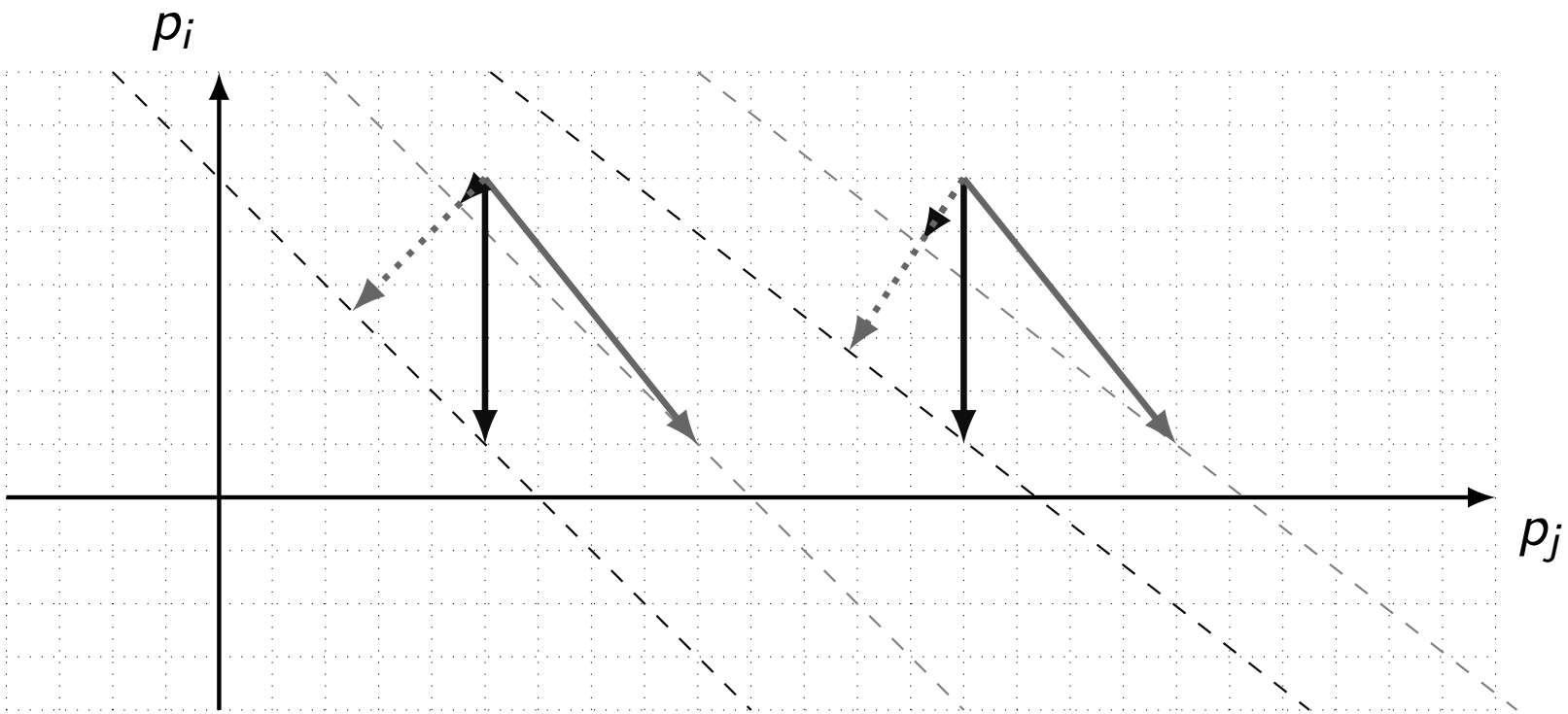
Measure and Conquer

Reference : Fomin, Grandoni and Kratsch, 2006.

General idea :

- Use a branch-and-bound algorithm.
- Give different weights to vertices.
- Balance this weights to make worst case better and best case worse.

Measure and Conquer



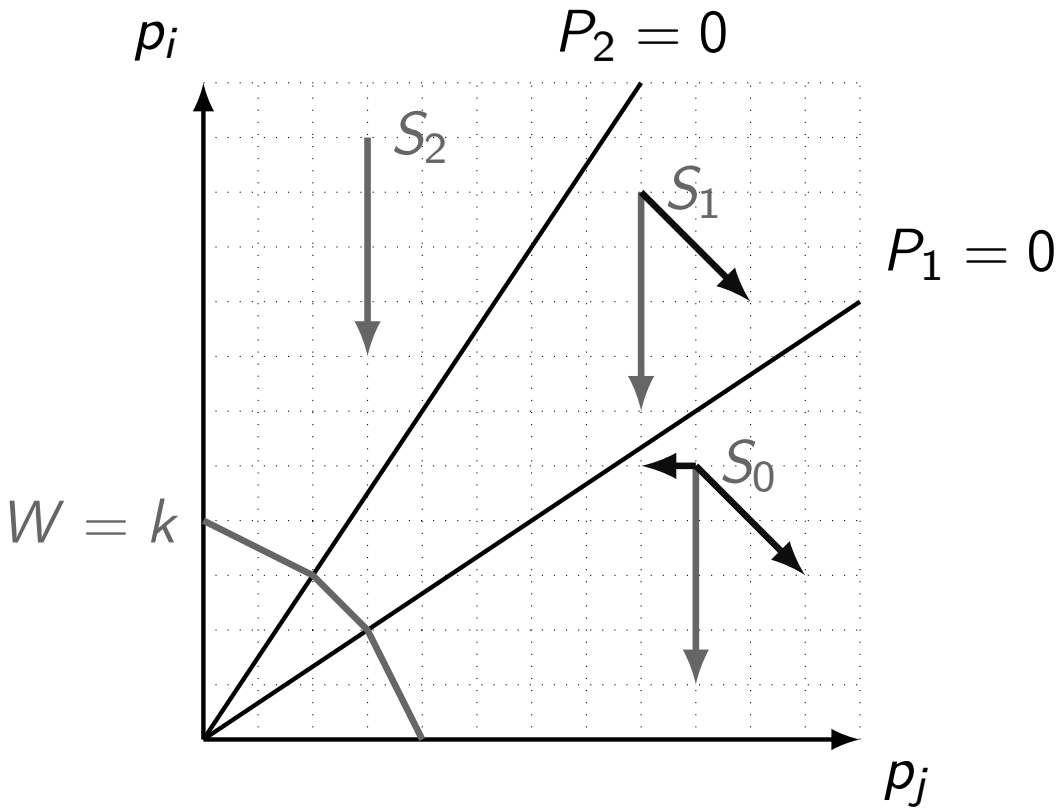
Partition and Measure

Reference : Bourgeois, Escoffier, Paschos and van Rooij, 2010.

General idea :

- Use a branch-and-bound algorithm.
- Split the space of possible instances among polyhedrons with specific properties.
- Make measure and conquer analyzis on each polyhedron.
- Balance all the weights to satisfy boundary conditions.

Partition



Opening

There exist different ways than exact algorithms to deal with inapproximable problems :

- ★ moderatly exponential approximation
- ★ parameterized algorithms
  $f(k) \times n^d$