Lecture

Solving combinatorial optimization problems using mathematical programming

Section 0 : Introduction

Pierre Fouilhoux

Université Sorbonne Paris Nord - LIPN CNRS

Hanoi, Vietnam, 2024

A **Combinatorial Optimization Problem**

is to

find a greatest (smallest) element
within a valuated finite set.

Combinatorial Optimization Problem

To find a greatest (smallest) element within a valuated finite set.

Combinatorial Optimization Problem

To find a greatest (smallest) element within a valuated finite set.

Given :

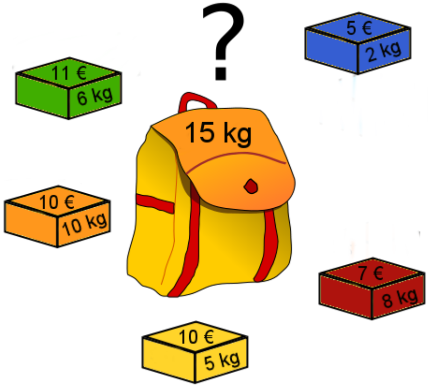
- a finite subset of elements $E = \{e_1, \ldots, e_n\}$
- a **solution set** $\mathcal{F}$ of subsets of E
- a weight $c = (c(e_1), \ldots, c(e_n))$

a **Combinatorial Optimization Problem** is to find a solution $F \in \mathcal{F}$ whose weight $c(F) = \sum_{e \in F} c(e)$ is maximum (or min.),

$$\textit{i.e.} \quad \max \{c(F) \mid F \in \mathcal{F}\}.$$

The knapsack problem

Which boxes to chose
to maximize the profit
without exceeding 15kg ?



Définition

Let us consider n **objects**.
Each object $i \in \{1, \ldots, n\}$
- with a **profit** g_i
- with a **weight** p_i
to be put inside a knapsack of **maximum total weight** P.

Définition

Let us consider n **objects**.
Each object $i \in \{1, \ldots, n\}$
- with a **profit** g_i
- with a **weight** p_i
to be put inside a knapsack of **maximum total weight** P.

Then we need to find a subset $S \subset \{1, \ldots, n\}$ such that

Définition

Let us consider n **objects**.
Each object $i \in \{1, \ldots, n\}$
- with a **profit** g_i
- with a **weight** p_i
to be put inside a knapsack of **maximum total weight** P.

Then we need to find a subset $S \subset \{1, \ldots, n\}$ such that

its weight $\sum_{i \in S} p_i$ is non-greater than P *(Knapsack Constraint)*

Définition

Let us consider n **objects**.
Each object $i \in \{1, \ldots, n\}$
- with a **profit** g_i
- with a **weight** p_i

to be put inside a knapsack of **maximum total weight** P.

Then we need to find a subset $S \subset \{1, \ldots, n\}$ such that

its weight $\sum_{i \in S} p_i$ is non-greater than P *(Knapsack Constraint)*

its profit $\sum_{i \in S} g_i$ is maximum. *(Objective function)*

How to incode a knapsack solution ?

An instance is given by :

i	1	2	3	4	5	
g_i	5	7	10	11	10	
p_i	2	8	10	6	5	≤ 15

A solution $S \subset \{1, \ldots, n\}$ is corresponding to **an incidence vector** χ^S such that

$$\chi^S[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise.} \end{cases}$$

$S_1 = \{1, 2, 5\}$

$\chi^{S_1} =$

1	1	0	0	1

How to incode a knapsack solution ?

An instance is given by :

i	1	2	3	4	5	
g_i	5	7	10	11	10	
p_i	2	8	10	6	5	≤ 15

A solution $S \subset \{1, \ldots, n\}$ is corresponding to **an incidence vector** χ^S such that

$$\chi^S[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise.} \end{cases}$$

$S_1 = \{1, 2, 5\}$

$\chi^{S_1} =$

1	1	0	0	1

profit :	5	7			10	$= 22$
weight :	2	8			5	$= 15$

How to test whether a vector is a solution ?

How to test whether a vector is a solution ?

i	1	2	3	4	5	
g_i	5	7	10	11	10	
p_i	2	8	10	6	5	≤ 15

Recognition Algorithm :

$S_2 = \{1, 4, 5\}$

$\chi^{S_2} =$

1	0	0	1	1

$pds \leftarrow 0$

For i from 1 to n

$pds \leftarrow pds + p_i * \chi^S[i]$

EndFor

If $pds \leq P$ **Then** True

Else False

How to test whether a vector is a solution ?

i	1	2	3	4	5	
g_i	5	7	10	11	10	
p_i	2	8	10	6	5	≤ 15

Recognition Algorithm :

$pds \leftarrow 0$

For i from 1 to n

$pds \leftarrow pds + p_i * \chi^S[i]$

EndFor

If $pds \leq P$ **Then** True

Else False

$S_2 = \{1, 4, 5\}$

$\chi^{S_2} =$	1	0	0	1	1	
weight :	2			6	5	$= 13$

Solution of profit 26

How to test whether a vector is a solution ?

i	1	2	3	4	5	
g_i	5	7	10	11	10	
p_i	2	8	10	6	5	≤ 15

Recognition Algorithm :

$pds \leftarrow 0$

For i from 1 to n

$pds \leftarrow pds + p_i * \chi^S[i]$

EndFor

If $pds \leq P$ **Then** True

Else False

$$S_3 = \{1, 2, 4\}$$

$\chi^{S_3} =$

1	1	0	1	0

How to test whether a vector is a solution ?

i	1	2	3	4	5	
g_i	5	7	10	11	10	
p_i	2	8	10	6	5	≤ 15

Recognition Algorithm :

$pds \leftarrow 0$

For i from 1 to n

$pds \leftarrow pds + p_i * \chi^S[i]$

EndFor

If $pds \leq P$ **Then** True

Else False

$S_3 = \{1, 2, 4\}$

$\chi^{S_3} =$

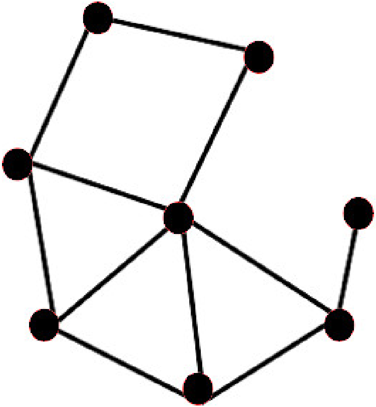
1	1	0	1	0

weight : 2 8 6 $= 16$

Not a solution.

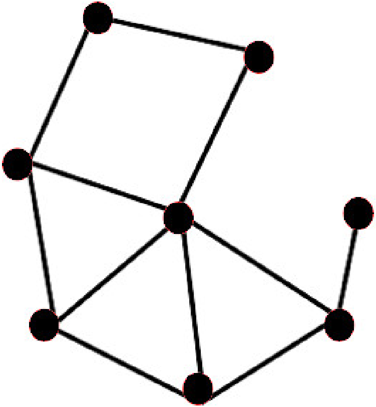
Huge knapsacks...

Graph



A **graph** $G = (V, E)$ is a pair where
- V is the node set
- $E \subseteq V \times V$ is the edge set.

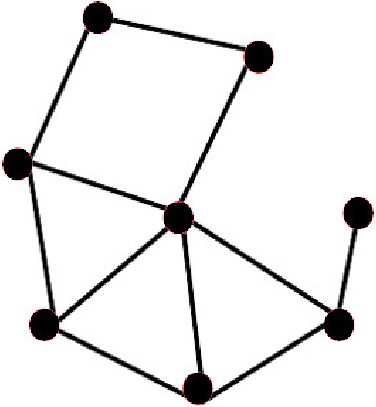
Graph



A **graph** $G = (V, E)$ is a pair where
- V is the node set
- $E \subseteq V \times V$ is the edge set.

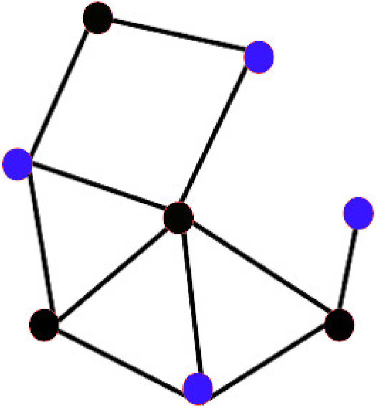
Two nodes are **adjacent** if they are linked by an edge.

Stable Set problem



A **stable set**
(or independent set)
is a pairwise non-adjacent node subset.

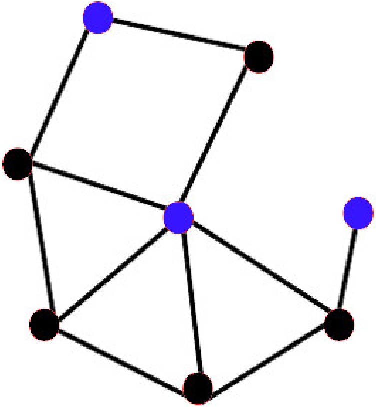
Stable Set problem



A **stable set**
(or independent set)
is a pairwise non-adjacent node subset.

The **stable set problem** is to find a stable set with a maximum number of nodes.

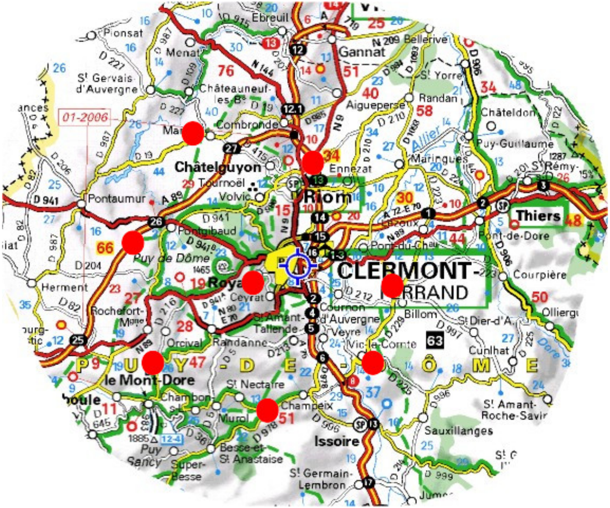
Stable Set problem



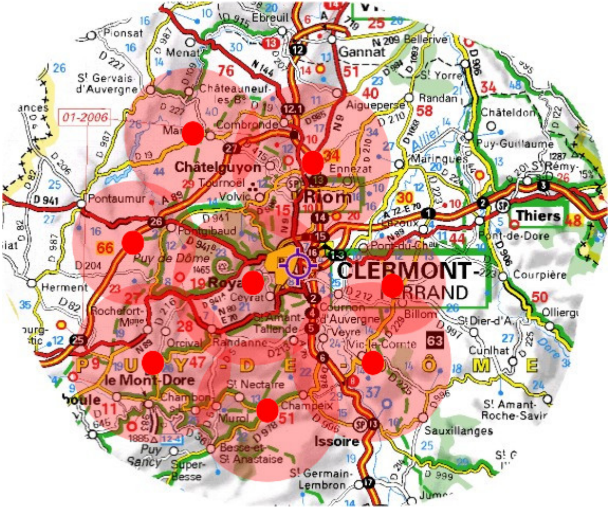
A **stable set**
(or independent set)
is a pairwise non-adjacent node subset.

The **stable set problem** is to find a stable set with a maximum number of nodes.

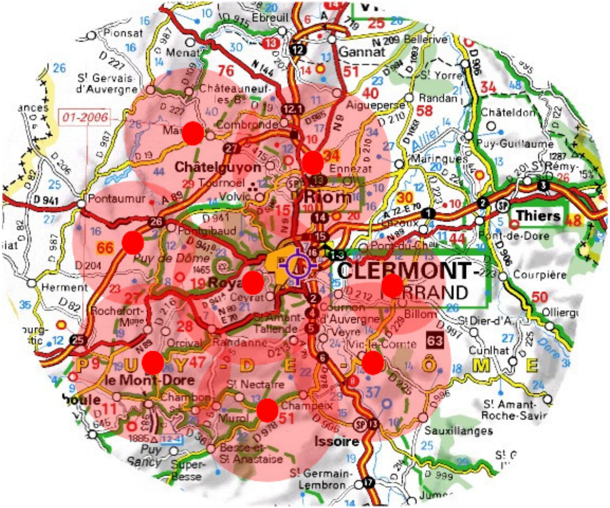
A set of antennas



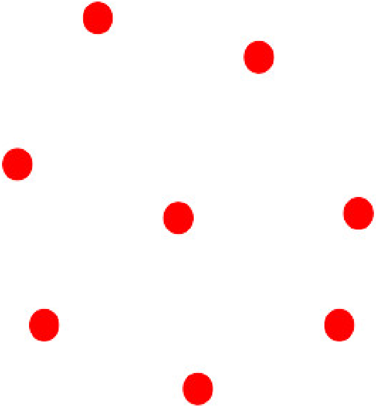
A set of antennas with their interference disks



How to find a subset of antennas with no interference ?



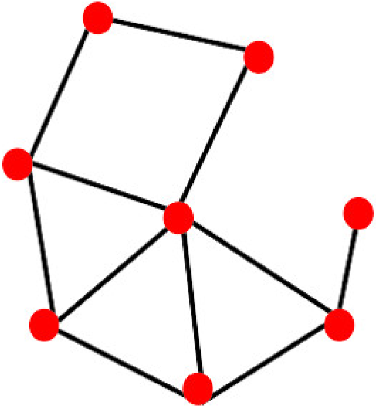
A model graph



Un graphe $G = (S, V)$ avec
- V : a node by antenna
- E : an edge between two antennas if their interference disks are in conflict.

Finding a subset of antennas with no interference reduces to the stable set problem.

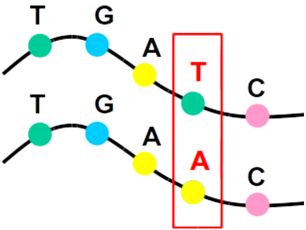
A model graph



Un graphe $G = (S, V)$ avec
- V : a node by antenna
- E : an edge between two antennas if their interference disks are in conflict.

Finding a subset of antennas with no interference reduces to the stable set problem.

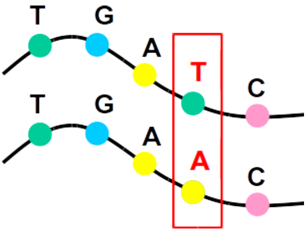
Combinatorics within genomic



Genome sequencing involves sorting through small fragments of DNA (called SNPs) that have been sequenced using bio-mechanical processes. In the process, false SNPs are created.

We want to sort the fragments, omitting as few SNPs as possible.

Combinatorics within genomic

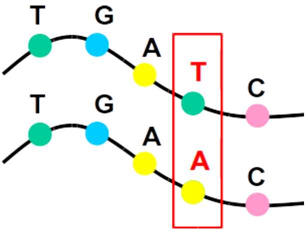


Genome sequencing involves sorting through small fragments of DNA (called SNPs) that have been sequenced using bio-mechanical processes. In the process, false SNPs are created.

We want to sort the fragments, omitting as few SNPs as possible.

Its a **Combinatorial optimisation problem**

Combinatorics within genomic



Genome sequencing involves sorting through small fragments of DNA (called SNPs) that have been sequenced using bio-mechanical processes. In the process, false SNPs are created.

We want to sort the fragments, omitting as few SNPs as possible.

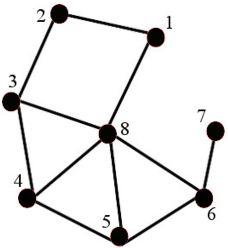
Its a **Combinatorial optimisation problem**

which also reduces to the **stable set problem !**.

Lippert, R., Schwartz, R., Lancia, G., Istrail, S. : Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. Brief. Bioinform 3, 23–31 (2002)

How to incode a stable set solution ? ?

Instance given by a graph with n nodes.



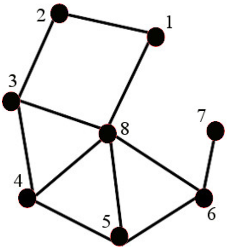
A solution $S \subset \{1, \ldots, n\}$ is corresponding to an **vecteur d'incidence** χ^S such that

$$\chi^S[i] = \begin{cases} 1 & \text{si } i \in S \\ 0 & \text{sinon.} \end{cases}$$

$S_1 = \{1, 7, 8\}$

i	1	2	3	4	5	6	7	8
$\chi^{S_1} =$	1	0	0	0	0	0	1	1

How to test whether a vector is a solution ?



Recognition Algorithm :

For i from 1 to n

For j from 1 to n

If $\{i, j\}$ is an edge

and if $\chi^S[i] = 1$ and $\chi^S[j] = 1$

Then STOP : False

EndFor

EndFor

STOP : True

$$S_1 = \{2, 7, 8\}$$

i	1	2	3	4	5	6	7	8
$\chi^{S_1} =$	0	1	0	0	0	0	1	1

How to test whether a vector is a solution ?

Recognition Algorithm :

For i from 1 to n

For j from 1 to n

If $\{i, j\}$ is an edge

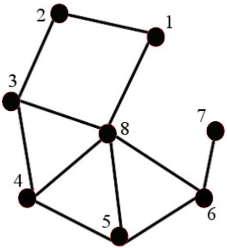
and if $\chi^S[i] = 1$ and $\chi^S[j] = 1$

Then STOP : False

EndFor

EndFor

STOP : True



$S_1 = \{2, 7, 8\}$

i	1	2	3	4	5	6	7	8
$\chi^{S_1} =$	0	1	0	0	0	0	1	1

Solution (with 3 nodes

Recognition algorithms

Knapsack

$pds \leftarrow 0$
For i from 1 to n
$pds \leftarrow pds + p_i * \chi^S[i]$
EndFor
If $pds \leq P$ **Then** True
Else False

Execution time proportionnal to n.

Recognition algorithms

Knapsack

$pds \leftarrow 0$
For i from 1 to n
  $pds \leftarrow pds + p_i * \chi^S[i]$
EndFor
If $pds \leq P$ **Then** True
  Else False

Execution time proportionnal to n.

Stable set

For i from 1 to n
  For j from 1 to n
    If $\{i, j\}$ is an edge
    and if $\chi^S[i] = 1$
    and $\chi^S[j] = 1$
    Then
     STOP : False
  EndFor
EndFor
STOP : True

Execution time proportionnal to n^2.

Recognition algorithms

Knapsack

$pds \leftarrow 0$
For i from 1 to n
  $pds \leftarrow pds + p_i * \chi^S[i]$
EndFor
If $pds \leq P$ **Then** True
  Else False

Execution time proportionnal to n.

Stable set

For i from 1 to n
  For j from 1 to n
    If $\{i, j\}$ is an edge
    and if $\chi^S[i] = 1$
    and $\chi^S[j] = 1$
    Then
     STOP : False
  EndFor
EndFor
STOP : True

Execution time proportionnal to n^2.

Polynomial Algorithms
which can be fast ($\leq n^4$ for instance).

Recognition algorithms

Knapsack

pds $\leftarrow 0$
For i from 1 to n
    $pds \leftarrow pds + p_i * \chi^S[i]$
EndFor
If $pds \leq P$ **Then** True
Else False

Execution time proportionnal to n.

Stable set

For i from 1 to n
    For j from 1 to n
        If $\{i, j\}$ is an edge
        and if $\chi^S[i] = 1$
        and $\chi^S[j] = 1$
        Then
         STOP : False
    EndFor
EndFor
STOP : True

Execution time proportionnal to n^2.

Polynomial Algorithms
which can be fast ($\leq n^4$ for instance).

Solving algorithms

Recognition algorithms

Knapsack

$pds \leftarrow 0$
For i from 1 to n
$pds \leftarrow pds + p_i * \chi^S[i]$
EndFor
If $pds \leq P$ **Then** True
Else False

Execution time proportionnal to n.

Stable set

For i from 1 to n
For j from 1 to n
If $\{i, j\}$ is an edge
and if $\chi^S[i] = 1$
and $\chi^S[j] = 1$
Then
STOP : False
EndFor
EndFor
STOP : True

Execution time proportionnal to n^2.

Polynomial Algorithms
which can be fast ($\leq n^4$ for instance).

Solving algorithms

Which is the complexity of the best known "Generic Algorithm" so find the optimal solution of combinatorial optimization problems ?

Recognition algorithms

Knapsack

$pds \leftarrow 0$
For i from 1 to n
$pds \leftarrow pds + p_i * \chi^S[i]$
EndFor
If $pds \leq P$ **Then** True
Else False

Execution time proportionnal to n.

Stable set

For i from 1 to n
For j from 1 to n
If $\{i, j\}$ is an edge
and if $\chi^S[i] = 1$
and $\chi^S[j] = 1$
Then
STOP : False
EndFor
EndFor
STOP : True

Execution time proportionnal to n^2.

Polynomial Algorithms
which can be fast ($\leq n^4$ for instance).

Solving algorithms

Which is the complexity of the best known "Generic Algorithm" so find the optimal solution of combinatorial optimization problems ?

Only **very very slow** algorithms with an execution time proportional to 2^n, 3^n, $!n$

Exponential algorithms.

Enumerate ?

Consider an optimization problem with n elements.
Assuming that we know a **polynomial** (and. fast) algorithm for recognizing a solution,
can we finde the best solution through enumeration ?

Enumerate ?

Consider an optimization problem with n elements.
Assuming that we know a **polynomial** (and. fast) algorithm for recognizing a solution,
can we finde the best solution through enumeration ?

For each subset S of elements
$\chi^S \leftarrow$ the incidence vector of S.
Recogntion algorithm for χ^S.
If χ^S is a solution
Stock S as the best known solution encountered yet.
EndFor
STOP : the stocked solution.

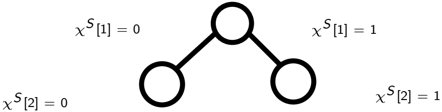
Enumerate ?

Consider an optimization problem with n elements.
Assuming that we know a **polynomial** (and. fast) algorithm for recognizing a solution,
can we finde the best solution through enumeration ?

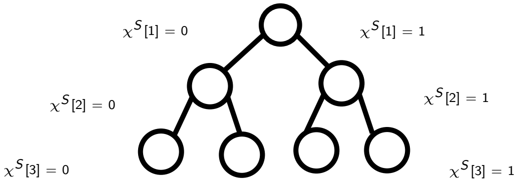
For each subset S of elements
  $\chi^S \leftarrow$ the incidence vector of S.
  Recogntion algorithm for χ^S.
  If χ^S is a solution
    Stock S as the best known solution encountered yet.
EndFor
STOP : the stocked solution.

There are 2^n **subsets** : **exponential** execution time !

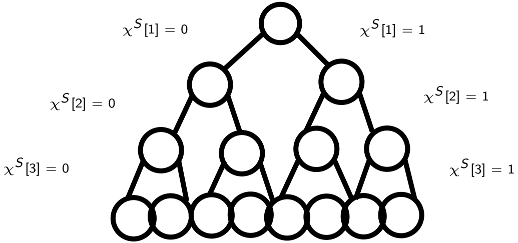
Combinatorial explosion



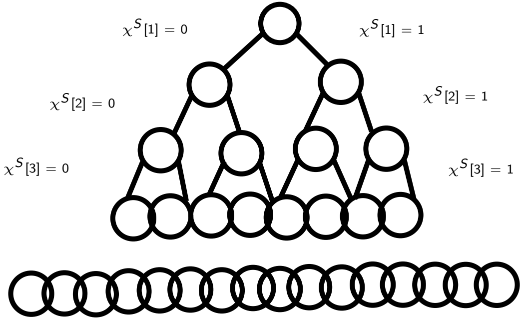
Combinatorial explosion



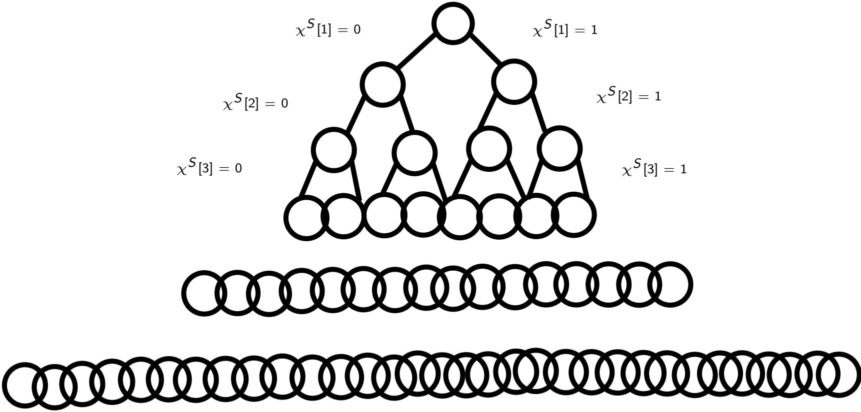
Combinatorial explosion



Combinatorial explosion



Combinatorial explosion



Combinatorial explosion

Let's take one of the most powerful computers in 2015

Tianhe-2 (China)

33.86 petaflops where 1 petaflop represents the processing of 10^{15} operations per second (one million billion).

Assume that the recognition algorithm takes $100n$ elementary operations. For $n = 10$, we can process 33 860 billion subsets in 1 second !

	Tianhe-2			Futuristic Computer
n	n^3	n^5	$100n2^n$	$100n2^n$
10	0,00...001 sec	0,00...001 sec		
20	0,00...001 sec	0,00...001 sec		
50	0,00...001 sec	0,00...001 sec		
60	0,00...001 sec	0,00000002 sec		
80	0,00...001 sec	0,0000009 sec		
100	0,00...001 sec	0,0000003 sec		
1000	0,00000003 sec	0,03 sec		

Combinatorial explosion

Let's take one of the most powerful computers in 2015

Tianhe-2 (China)

33.86 petaflops where 1 petaflop represents the processing of 10^{15} operations per second (one million billion).

Assume that the recognition algorithm takes $100n$ elementary operations. For $n = 10$, we can process 33 860 billion subsets in 1 second !

	Tianhe-2			Futuristic Computer
n	n^3	n^5	$100n2^n$	$100n2^n$
10	0,00...001 sec	0,00...001 sec	0,00...001 sec	
20	0,00...001 sec	0,00...001 sec	0,00000006 sec	
50	0,00...001 sec	0,00...001 sec	168,9 sec	
60	0,00...001 sec	0,00000002 sec	57,6 h	
80	0,00...001 sec	0,0000009 sec	9200 years	
100	0,00...001 sec	0,0000003 sec		
1000	0,00000003 sec	0,03 sec		

Combinatorial explosion

Let's take one of the most powerful computers in 2015

Tianhe-2 (China)

33.86 petaflops where 1 petaflop represents the processing of 10^{15} operations per second (one million billion).

Assume that the recognition algorithm takes $100n$ elementary operations. For $n = 10$, we can process 33 860 billion subsets in 1 second !

	Tianhe-2			Futuristic Computer
n	n^3	n^5	$100n2^n$	$100n2^n$
10	0,00...001 sec	0,00...001 sec	0,00...001 sec	
20	0,00...001 sec	0,00...001 sec	0,00000006 sec	
50	0,00...001 sec	0,00...001 sec	168,9 sec	
60	0,00...001 sec	0,00000002 sec	57,6 h	
80	0,00...001 sec	0,0000009 sec	9200 years	
100	0,00...001 sec	0,0000003 sec	$1,21.10^{10}$ years	
1000	0,00000003 sec	0,03 sec	1.10^{282} years	

Age of the universe $13,7.10^9$ years

Combinatorial explosion

Let's take one of the most powerful computers in 2015

Tianhe-2 (China)

33.86 petaflops where 1 petaflop represents the processing of 10^{15} operations per second (one million billion).

Assume that the recognition algorithm takes $100n$ elementary operations. For $n = 10$, we can process 33 860 billion subsets in 1 second !

	Tianhe-2			Futuristic Computer
n	n^3	n^5	$100n2^n$	$100n2^n$
10	0,00...001 sec	0,00...001 sec	0,00...001 sec	0,00...001 sec
20	0,00...001 sec	0,00...001 sec	0,00000006 sec	0,00...001 sec
50	0,00...001 sec	0,00...001 sec	168,9 sec	0,000001 sec
60	0,00...001 sec	0,00000002 sec	57,6 h	0,0001 sec
80	0,00...001 sec	0,0000009 sec	9200 years	100 years
100	0,00...001 sec	0,0000003 sec	$1,21.10^{10}$ years	$1,21.10^{9}$ years
1000	0,00000003 sec	0,03 sec	1.10^{282} years	...

Age of the universe $13,7.10^9$ years

Combinatorial Explosion

Enumerating 2^n solutions is not a technical problem
that very powerful computers could sweep away.

It's necessary to **circumvent the combinatorial explosion** with mathematical and algorithmic tools.

Problem complexity

Just because we know an exponential algorithm for solving a problem doesn't mean it's difficult !

Problem complexity

Just because we know an exponential algorithm for solving a problem doesn't mean it's difficult !

To crack a nut, you can :

Problem complexity

Just because we know an exponential algorithm for solving a problem doesn't mean it's difficult !

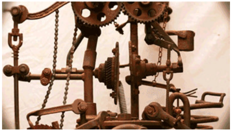
To crack a nut, you can :

Problem complexity

Just because we know an exponential algorithm for solving a problem doesn't mean it's difficult !

To crack a nut, you can :

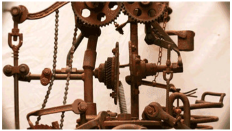




Problem complexity

Just because we know an exponential algorithm for solving a problem doesn't mean it's difficult !

To crack a nut, you can :

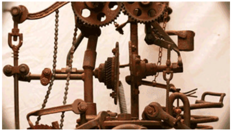


Problem complexity

Just because we know an exponential algorithm for solving a problem doesn't mean it's difficult !

To crack a nut, you can :






We're looking for the **fastest** algorithm to solve a problem !

Problem Complexity

A problem is **of exponential complexity**
if an exponential algorithm exists to solve it.
$\Rightarrow$ the problem class $\mathcal{EXP}$.

Problem Complexity

A problem is **of exponential complexity**
if an exponential algorithm exists to solve it.
$\Rightarrow$ the problem class $\mathcal{EXP}$.

A problem is **of polynomial complexity**
if a polynomial algorithm exists to solve it.
$\Rightarrow$ the problem class $\mathcal{P}$.

Problem Complexity

A problem is **of exponential complexity**
if an exponential algorithm exists to solve it.
$\Rightarrow$ the problem class $\mathcal{EXP}$.

A problem is **of polynomial complexity**
if a polynomial algorithm exists to solve it.
$\Rightarrow$ the problem class $\mathcal{P}$.

So $\mathcal{P}$ is **included** in $\mathcal{EXP}$, we note $\mathcal{P} \subset \mathcal{EXP}$

Problem Complexity

A problem is **of exponential complexity**
if an exponential algorithm exists to solve it.
$\Rightarrow$ the problem class $\mathcal{EXP}$.

A problem is **of polynomial complexity**
if a polynomial algorithm exists to solve it.
$\Rightarrow$ the problem class $\mathcal{P}$.

So $\mathcal{P}$ is **included** in $\mathcal{EXP}$, we note $\mathcal{P} \subset \mathcal{EXP}$

Question : In what complexity classes are Combinatoriel Optimization problems ?

$\mathcal{NP}$ **problems**

A particular classe have been created...
Combinatorial optimization problems for which we know **how to recognize a solution** with a polynomial algorithm

$\Rightarrow$ the problem class $\mathcal{NP}$ "Nondeterministic polynomial time"

$\mathcal{NP}$ problems

A particular classe have been created...
Combinatorial optimization problems for which we know **how to recognize a solution** with a polynomial algorithm

$\Rightarrow$ the problem class $\mathcal{NP}$ "Nondeterministic polynomial time"

Under this assumption, we've seen that it's possible to use an exponential enumeration algorithm, then

$\Rightarrow \mathcal{NP} \subset \mathcal{EXP}$

$\mathcal{NP}$ problems

A particular classe have been created...
Combinatorial optimization problems for which we know **how to recognize a solution** with a polynomial algorithm

$\Rightarrow$ the problem class $\mathcal{NP}$ "Nondeterministic polynomial time"

Under this assumption, we've seen that it's possible to use an exponential enumeration algorithm, then

$\Rightarrow \mathcal{NP} \subset \mathcal{EXP}$

In addition, a polynomial problem is in $\mathcal{NP}$

$\Rightarrow \mathcal{P} \subset \mathcal{NP} \subset \mathcal{EXP}$

And then ?

Is $\mathcal{P}$ equal to $\mathcal{NP}$?

i.e.

"Is there a polynomial algorithm for solving combinatorial optimization problems ?"

Answer :

And then ?

Is $\mathcal{P}$ equal to $\mathcal{NP}$?

i.e.

"Is there a polynomial algorithm for solving combinatorial optimization problems ?"

Answer : We don't know !

On a human scale, we only know this enumeration algorithm !

It's one of the 7 problems in the Clay Mathematics Institute of Cambridge's million-dollar Millennium Prize Problems !

Is every Combinatorial Optimisation problem exponential ?

Is every Combinatorial Optimisation problem exponential ?

A very simple knapsack subcase

Let's look at a very simple knapsack instance :
a knapsack with all objects of the same weight.

Is every Combinatorial Optimisation problem exponential ?

A very simple knapsack subcase

Let's look at a very simple knapsack instance :
a knapsack with all objects of the same weight.

A **greedy** algorithm :

- sort the *n* objects from the most expensive to the least expensive
- take the objects one by one in that order as long as they fit in the bag !

Is every Combinatorial Optimisation problem exponential ?

A very simple knapsack subcase

Let's look at a very simple knapsack instance :
a knapsack with all objects of the same weight.

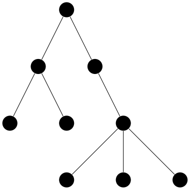
A **greedy** algorithm :

- sort the n objects from the most expensive to the least expensive
- take the objects one by one in that order as long as they fit in the bag !

This algorithm is polynomial (of the order of n^2).

A very simple stable set subcase

Let's look at a very simple stable set instance :
finding a maximum cardinality stable set over a **tree**.



Tree property :
Given a leaf v, there exists a maximum cardinality stable set containing v.

(Proof : Let v a leaf and u its unique neighbour. Let S be a maximum stable set. Either $u \notin S$, then $S \cup \{v\}$; or $u \notin S$, then $S \cup \{u\} \setminus \{v\}$ is another maximum stable set).

A very simple stable set subcase

A **greedy** algorithm :

$S \leftarrow \emptyset$
While G has at least one edge
    Let v be a leaf and u its neighbour
    $S \leftarrow S \cup \{v\}$
    Delete from G nodes u and v and all their incident edges
EndWhile
Add to S all the remaining nodes.
STOP : S

A very simple stable set subcase

A **greedy** algorithm :

$S \leftarrow \emptyset$
While G has at least one edge
    Let v be a leaf and u its neighbour
    $S \leftarrow S \cup \{v\}$
    Delete from G nodes u and v and all their incident edges
EndWhile
Add to S all the remaining nodes.
STOP : S

This algorithm is polynomial (of the order of n).

$\mathcal{NP}$-hard

In the current state of scientific knowledge, it is not known whether or not there is a polynomial algorithm for solving the knapsack or the stable set problem in general !

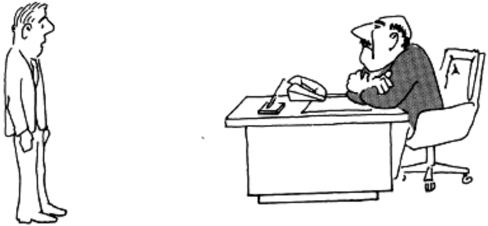
$\mathcal{NP}$-hard

In the current state of scientific knowledge, it is not known whether or not there is a polynomial algorithm for solving the knapsack or the stable set problem in general !

In fact, we were able to prove that knapsack or the stable set problems are just as difficult as all the problems in the $\mathcal{NP}$ class !

A problem is said to be $\mathcal{NP}$-**hard**
if it is as difficult as any problem in the $\mathcal{NP}$ class.
$\Rightarrow$ the problem class $\mathcal{NP}$-hard.

$\mathcal{NP}$-hardness



Boss, I can't find a polynomial algorithm for the stable set problem

Figure from "The Garey et Johnson"

$\mathcal{NP}$-hardness

But if you think I'm just an ungifted searcher

Figure from "The Garey et Johnson"

$\mathcal{NP}$-hardness

...neither are all the others !

Figure from "The Garey et Johnson"

Strongly or weakly $\mathcal{NP}$-hard

A computational problem may have numerical parameters : like the weight of a knapsack.

A $\mathcal{NP}$-hard problem is said to be **weakly $\mathcal{NP}$-hard**
if there is an algorithm for the problem whose running time is polynomial in the dimension of the problem and magnitudes of its data.

And otherwise, it is called **strongly $\mathcal{NP}$-hard**.

Strongly or weakly $\mathcal{NP}$-hard
A computational problem may have numerical parameters : like the weight of a knapsack.

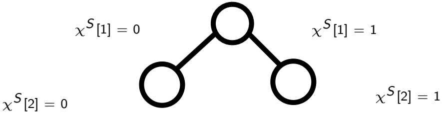
A $\mathcal{NP}$-hard problem is said to be **weakly $\mathcal{NP}$-hard**
if there is an algorithm for the problem whose running time is polynomial in the dimension of the problem and magnitudes of its data.

And otherwise, it is called **strongly $\mathcal{NP}$-hard**.

For weakly $\mathcal{NP}$-hard, it's often the case, that there exists a **dynaming programming scheme** whose complexity depends on the magnitudes of the data.

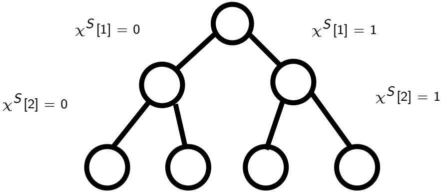
Dynamic Programming scheme

Some subcases are the same within the enumeration : this puts the brakes on the combinatorial explosion.



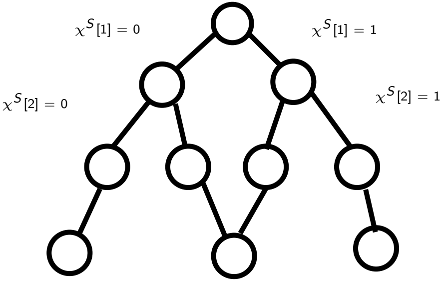
Dynamic Programming scheme

Some subcases are the same within the enumeration : this puts the brakes on the combinatorial explosion.



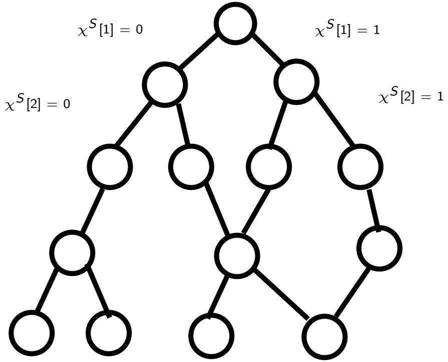
Dynamic Programming scheme

Some subcases are the same within the enumeration : this puts the brakes on the combinatorial explosion.



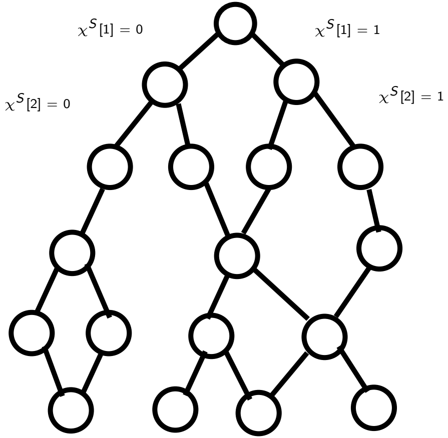
Dynamic Programming scheme

Some subcases are the same within the enumeration :
this puts the brakes on the combinatorial explosion.



Dynamic Programming scheme

Some subcases are the same within the enumeration : this puts the brakes on the combinatorial explosion.



1. Definition and complexity

2. List of OC problems

Knapsack problem

Input : n objects
profit g_i $\forall i \in \{1, \ldots, n\}$
weight p_i $\forall i \in \{1, \ldots, n\}$
maximum total weight P.

Output : Subset $S \subseteq \{1, \ldots, n\}$
s.t. $\sum_{i \in S} p_i \leq P$

Objective : Max $\sum_{i \in S} w_i$

Knapsack MIP

$$\mathrm{Max} \sum_{i=1}^{n} w_i x_i$$

$$\sum_{i=1}^{n} p_i x_i \leq P,$$

$$0 \leq x_i \leq 1, \text{ pour tout objet } i = 1, ..., n,$$

$$x_i \in \boldsymbol{N}, \text{ pour tout objet } i = 1, ..., n.$$

Maximum weight stable set problem

Input : Undirected graph $G = (V, E)$
cost $w_u\ \forall u \in V$

Output : Subset $S \subseteq V$ of non-adjacent nodes

Objective : Max $\sum_{i \in S} w_i$

Complexity : Strongly NP-hard
Polynomial cases : perfect graphs (tree, planar, interval graphs...)
Difficulty : With MIP solver $n = 1000$ in often more than 1 hour
Some dedicated methodes (russian doll algo)

Shortest path problem

Input : Undirected (or directed) graph $G = (V, E)$
Two nodes $u_0, u_1 \in V$
Lenghts $l_e \ \forall e \in E$

Output : Path μ of G from u_O to u_1

Objective : Min $\sum_{e \in \mu} l_e$

Complexity : Polynomial
Difficulty : With Dijkstra algorithm up to thousand of nodes in a few sec
With A^* algorithme, up to billions !

Shortest path problem

To quickly go from a point to another

The Traveling Salesman Problem (TSP)

Input : Undirected (or directed) graph $G = (V, E)$
lenght $l_e \ \forall e \in E$

Output : An hamiltonian cycle C of G (i.e. C goes once through each node)

Objective : Min $\sum_{e \in C} l_e$

Complexity : Strongly NP-hard
Polynomial cases : ?
Difficulty : Before 2003 : 200 nodes within several hours

The Traveling Salesman Problem (TSP)

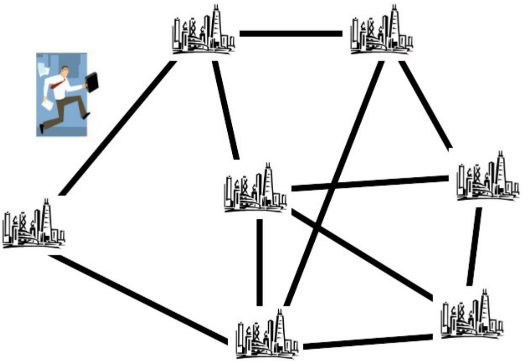
Input : Undirected (or directed) graph $G = (V, E)$
lenght $l_e \ \forall e \in E$

Output : An hamiltonian cycle C of G (i.e. C goes once through each node)

Objective : Min $\sum_{e \in C} l_e$

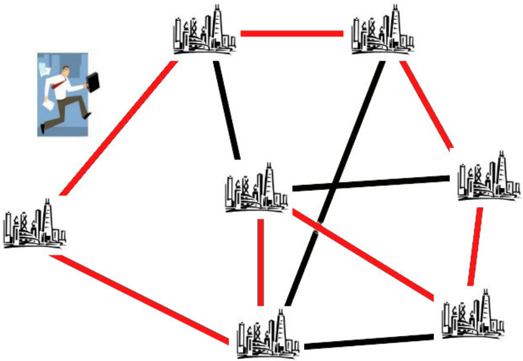
Complexity : Strongly NP-hard
Polynomial cases : ?
Difficulty : Before 2003 : 200 nodes within several hours
Concorde : a Branch-and-Cut algorithm and polyhedral results solves up to 200000 cities within one day !

The Traveling Salesman Problem (TSP)



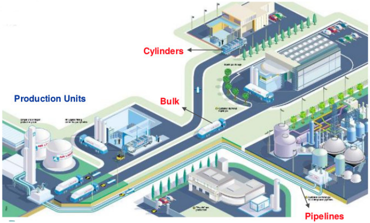
Starting from a city and going back to it
going once througt all the other cities
with a sorthest cycle.

The Traveling Salesman Problem (TSP)



Starting from a city and going back to it
going once througt all the other cities
with a sorthest cycle.

Real-world Operations Research problem



The **inventory routing problem** introduced by **Air Liquide** company with real daily data and questions to answer :

Determine routes for liquid oxygen trucks to deliver hospitals so that

- hospital tanks are never empty (remote monitoring)
- rounds are feasible within the driver's working day
- costs are minimized !

Source : ROADEF Challenge 2016

How to solve Combinatorial Optimization Problem ?

• If the instances are very large and the problem very hard to solve
or if you do not have much time to spend on solving method

→ There exist methods to obtain "good" solutions (heuristics, meta-heuristics, machine learning...)

• If the problem is really important with a higly cost associated to solution
and if you have some time (several hours...)
The optimal solution is to be computed !

→ We will see in this lecture **how to circumvent the combinatorial explosion using mathematical programming !**

The Traveling salesman problem has been solved till 200 000 cicites !