

# **Validation théorique et évaluation expérimentale d'algorithmes**

Christine Solnon

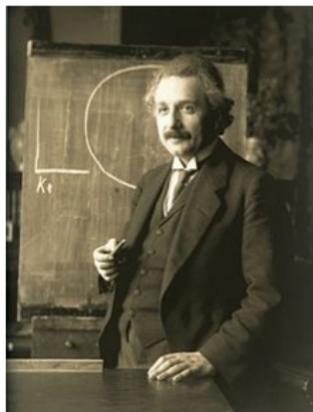
INSA de Lyon - 5IF

2015

# Pourquoi confronter théorie et expérimentation ?

*In theory, theory and practice are the same. In practice, they are not.*

(A. Einstein)



*Experience without theory is blind, but theory without experience is like mere intellectual play.*

(I. Kant)



# De la théorie à l'expérimentation (1/2)

Etapes permettant de passer d'un problème à une solution informatique

## Etape 1 : Etude théorique du problème

- Spécification des entrées, sorties, préconditions et postrelations
- Etude de la complexité théorique du problème
- Choix d'une approche de résolution adaptée au problème

## Etape 2 : Conception théorique de l'algorithme

- Choix d'un principe de résolution + struct. de données (*Algorithm tuning*)
- Etude théorique de la complexité et de la correction de l'algorithme

Retour à l'étape 1 si performances asymptotiques insuffisantes :

↪ Choisir une autre approche de résolution ?

↪ Exploiter des préconditions sur les entrées pour réduire la complexité ?

## Etape 3 : Ecriture du code source

- Choix d'un langage et des bibliothèques

# De la théorie à l'expérimentation (2/2)

Etapes permettant de passer d'un problème à une solution informatique

## Etape 4 : Préparation de l'évaluation expérimentale

- Conception de l'expérimentation : Critères de performance, Protocole expérimental, Jeux de test, ...
- Constitution de l'environnement de test : Machines, scripts, ...

## Etape 5 : Réalisation de l'expérimentation

- Lancer les exécutions et collecter les résultats
- Analyser les résultats

↪ Retour aux étapes 1, 2 ou 3 si nécessaire

## Etape 6 : Amélioration du code

- *Refactoring*, *Profiling* et *Code tuning*
- *Parameter tuning* et *Configuration tuning*

↪ Retour à l'étape 5

## Quelques livres pour en savoir plus

- Cormen, Leiserson, Rivest : Introduction à l'algorithmique  
MIT Press and McGraw-Hill, 2009 (3ème édition)
- Perifel : Complexité algorithmique  
[http://www.liafa.univ-paris-diderot.fr/~sperifel/livre\\_complexite.html](http://www.liafa.univ-paris-diderot.fr/~sperifel/livre_complexite.html)
- Papadimitriou : Computational complexity  
Editions Addison-Wesley, 1994
- McGeoch : A guide to experimental algorithmics  
Cambridge University Press, 2012
- Baillargeon : Petit cours d'autodéfense intellectuelle  
Lux Editeur, 2005

... et une BD à (se faire) offrir

- Doxiatis, Papadimitriou : Logicomix, an epic search for truth  
Bloomsbury Publishing, 2009.

# Plan du cours

## 1 Validation théorique d'algorithmes

- Définitions préliminaires (rappels)
- Classes de complexité
- Illustration : Complexité des problèmes d'appariement de graphes

## 2 Evaluation expérimentale d'algorithmes

## 3 Ingénierie algorithmique

# Problèmes, instances et algorithmes

## Spécification d'un problème :

- Paramètres en entrée et en sortie
- Eventuellement : Préconditions sur les paramètres en entrée
- Postrelation entre les valeurs des paramètres en entrée et en sortie

## Instance d'un problème :

Valuation des paramètres en entrée satisfaisant les préconditions

## Algorithme pour un problème $P$ :

Séquence d'instructions élémentaires permettant de calculer les valeurs des paramètres en sortie à partir des valeurs des paramètres en entrée, pour toute instance de  $P$

## Exemple 1 : Recherche d'un élément dans un tableau trié

### Spécification du problème :

- Entrées :
  - un tableau  $tab$  comportant  $n$  entiers indicés de 0 à  $n - 1$
  - une valeur entière  $e$
- Sortie : un entier  $i$
- Précondition : les éléments de  $tab$  sont triés par ordre croissant
- Postrelation :
  - si  $\forall j \in [0, n - 1], tab[j] \neq e$  alors  $i = n$
  - sinon  $i \in [0, n - 1]$  et  $tab[i] = e$

### Exemples d'instances :

- Entrées :  $e = 8$  et  $tab =$ 

4	4	7	8	10	11	12
---	---	---	---	----	----	----

  
 $\rightsquigarrow$  Sortie :  $i = 3$
- Entrées :  $e = 9$  et  $tab =$ 

4	4	7	8	10	11	12
---	---	---	---	----	----	----

  
 $\rightsquigarrow$  Sortie :  $i = 7$

## Exemple 2 : Tri des éléments d'un tableau

### Spécification du problème :

- Entrées : un tableau  $tab^-$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Sortie : un tableau  $tab^+$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Postrelation :
  - $tab^+[0..n - 1]$  est une permutation de  $tab^-[0..n - 1]$
  - $\forall i \in [0, n - 2], tab^+[i] \leq tab^+[i + 1]$

### Exemples d'instances :

- Entrées :  $tab^- =$ 

4	2	9	4	0	7	1
---	---	---	---	---	---	---

  
 $\rightsquigarrow$  Sorties :  $tab^+ =$ 

0	1	2	4	4	7	9
---	---	---	---	---	---	---
- Entrées :  $tab^- =$ 

4	7	7	4
---	---	---	---

  
 $\rightsquigarrow$  Sorties :  $tab^+ =$ 

4	4	7	7
---	---	---	---

## Exemple 3 : Satisfiabilité d'une formule booléenne (SAT)

### Spécification du problème :

- Entrées : une formule booléenne  $F$  définie sur un ens.  $X$  de  $n$  variables
- Sortie : une valuation  $V : X \rightarrow \{\text{vrai}, \text{faux}\}$  des variables de  $F$
- Précondition :  $F$  est sous forme normale conjonctive (CNF)
- Postrelation : Si  $F$  est satisfiable alors  $V$  satisfait  $F$

### Exemple d'instance

- Entrées :  $X = \{a, b, c, d, e\}$ ,  
 $F = (a \vee \neg b) \wedge (\neg a \vee c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg e) \wedge (a \vee b \vee d \vee e)$   
 $\rightsquigarrow$  Sortie :  $V = \{a = \text{vrai}, b = \text{faux}, c = \text{vrai}, d = \text{vrai}, e = \text{faux}\}$

# Notion de complétude

## Algorithme complet pour un problème :

Algorithme capable de calculer des valeurs en sortie pour toutes les instances du problème

## Exemple d'algorithme incomplet pour SAT :

---

**Entrées** : Une formule booléenne  $F$  définie sur un ensemble  $X$  de  $n$  variables

**Sorties** : Une valuation  $V : X \rightarrow \{\text{vrai}, \text{faux}\}$

**début**

Initialiser toutes les variables de  $X$  à *vrai* dans  $V$

**tant que**  $V$  ne satisfait pas toutes les clauses de  $F$  **faire**

    Choisir une variable  $x_i$  de  $X$

    Changer la valuation de  $x_i$  dans  $V$

**retourner**  $V$

---

→ Les algos basés sur la recherche locale sont généralement incomplets !

# Notion de correction

## Algorithme correct pour un problème :

La postrelation entre entrées et sorties est vérifiée pour toutes les sorties

## Exemple d'algorithme incorrect pour maxSAT :

**Entrées** : Un ens.  $X$  de  $n$  var., une formule booléenne  $F$ , et un nb max. d'itérations  $maxIter$

**Sorties** : nb max de clauses de  $F$  pouvant être satisfaites par une valuation de  $X$

**début**

Initialiser toutes les variables de  $X$  à *vrai* dans  $V$

$nbSat \leftarrow$  nombre de clauses de  $F$  satisfaites par  $V$

$nbIter \leftarrow 0$

**tant que**  $nbIter < maxIter$  et  $nbSat < n$  **faire**

    Changer la valuation d'une variable dans  $V$

**si**  $nbSat <$  nombre de clauses de  $F$  satisfaites par  $V$  **alors**

$nbSat \leftarrow$  nombre de clauses de  $F$  satisfaites par  $V$

$nbIter \leftarrow nbIter + 1$

**retourner**  $nbSat$

## Changer la postcondition !

→ La sortie est une borne inf du nb de clauses pouvant être satisfaites

# Complexité d'un algorithme

## Estimation des ressources nécessaires à l'exécution d'un algorithme :

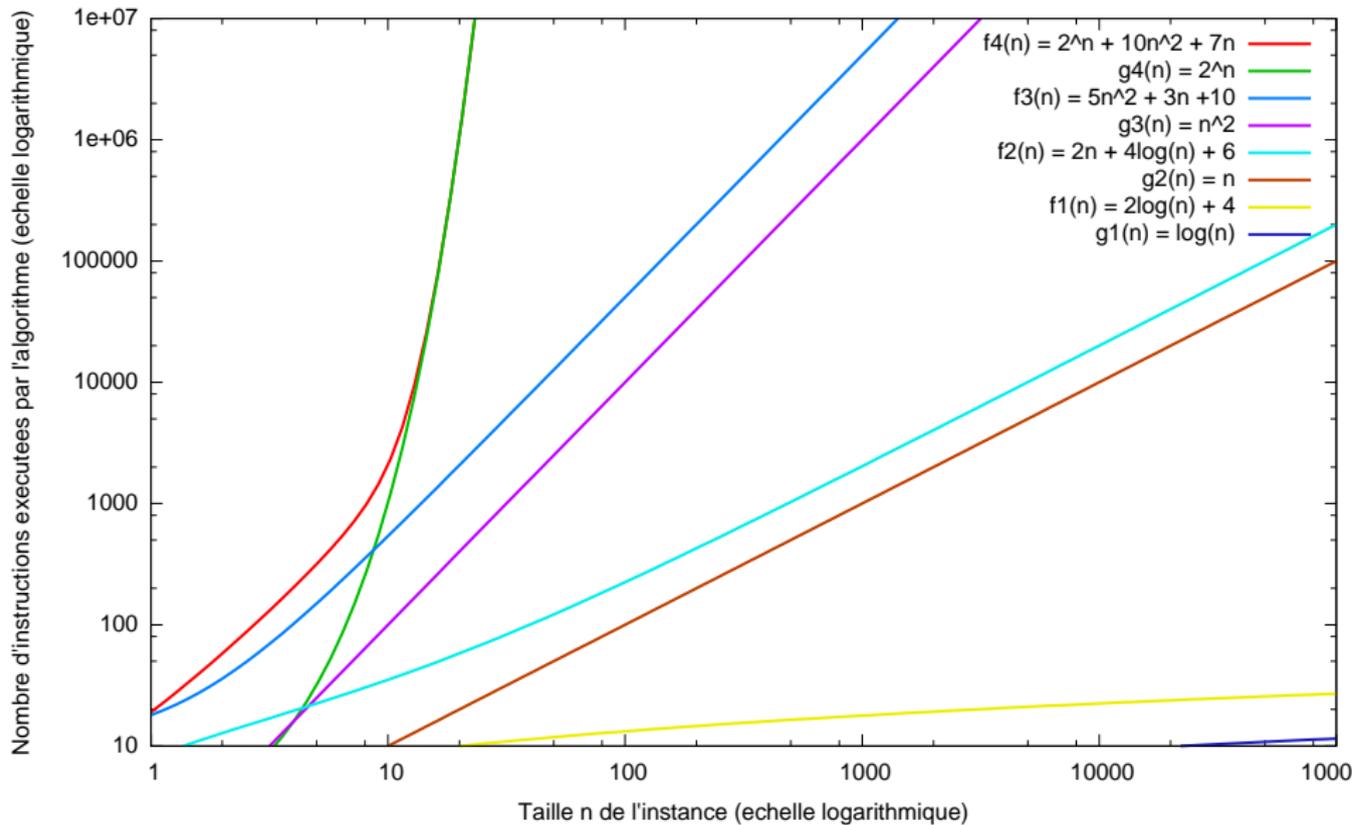
- Temps = estimation du nombre d'instructions élémentaires
- Espace = estimation de l'espace mémoire utilisé

↪ Estimation dépendante de la taille  $n$  des paramètres en entrée

## Ordre de grandeur d'une fonction $f(n)$ :

$\mathcal{O}(g(n)) \rightsquigarrow \exists c, n_0$  tel que  $\forall n > n_0, f(n) < c.g(n)$

- $\mathcal{O}(\log_k(n))$  : logarithmique
- $\mathcal{O}(n)$  : linéaire
- $\mathcal{O}(n^k)$  : polynomial
- $\mathcal{O}(k^n)$  : exponentiel



# Quiz : Complexité de quelques algorithmes de tri

## Spécification du problème (rappel) :

- Entrées : un tableau  $tab^-$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Sortie : un tableau  $tab^+$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Postrelation :  $tab^+[0..n - 1]$  est une permutation de  $tab^-[0..n - 1]$  et  $\forall i \in [0, n - 2], tab^+[i] \leq tab^+[i + 1]$

## Complexité du tri par sélection ?

début

```
pour  $i$  variant de 0 à  $n - 2$  faire
```

```
  /* Invariant :  $tab[0..i - 1]$  est trié et contient des          */
```

```
  /* éléments inférieurs ou égaux à ceux de  $tab[i..n - 1]$       */
```

```
  rechercher l'indice  $ipp$  du plus petit élément de  $tab[i..n - 1]$ 
```

```
  échanger  $tab[ipp]$  et  $tab[i]$ 
```

# Quiz : Complexité de quelques algorithmes de tri

## Spécification du problème (rappel) :

- Entrées : un tableau  $tab^-$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Sortie : un tableau  $tab^+$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Postrelation :  $tab^+[0..n - 1]$  est une permutation de  $tab^-[0..n - 1]$  et  $\forall i \in [0, n - 2], tab^+[i] \leq tab^+[i + 1]$

## Complexité du tri par insertion ?

début

```
pour  $i$  variant de 1 à  $n - 1$  faire
```

```
  /* Invariant :  $tab[0..i - 1]$  est trié
```

```
  insérer  $tab[i]$  dans  $tab[0..i - 1]$ 
```

```
*/
```

# Quiz : Complexité de quelques algorithmes de tri

## Spécification du problème (rappel) :

- Entrées : un tableau  $tab^-$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Sortie : un tableau  $tab^+$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Postrelation :  $tab^+[0..n - 1]$  est une permutation de  $tab^-[0..n - 1]$  et  $\forall i \in [0, n - 2], tab^+[i] \leq tab^+[i + 1]$

## Complexité du tri rapide ?

### début

**si**  $n > 1$  **alors**

Choisir un élément *pivot* dans  $tab[0..n - 1]$

Ré-arranger les éléments de  $tab[0..n - 1]$  de sorte que :

→  $tab[0..i - 1]$  ne contienne que des éléments inférieurs ou égaux à *pivot*

→  $tab[i] = pivot$

→  $tab[i + 1..n - 1]$  ne contienne que des éléments supérieurs à *pivot*

Trier  $tab[0..i - 1]$

Trier  $tab[i + 1..n - 1]$

# Quiz : Complexité de quelques algorithmes de tri

## Spécification du problème (rappel) :

- Entrées : un tableau  $tab^-$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Sortie : un tableau  $tab^+$  comportant  $n$  entiers indicés de 0 à  $n - 1$
- Postrelation :  $tab^+[0..n - 1]$  est une permutation de  $tab^-[0..n - 1]$  et  $\forall i \in [0, n - 2], tab^+[i] \leq tab^+[i + 1]$

## Complexité du tri par dénombrement ?

### début

Soient  $min$  et  $max$  les valeurs minimales et maximales de  $tab[0..n - 1]$

Initialiser à 0 un tableau  $nbOcc$  indicé de  $min$  à  $max$

**pour**  $i$  variant de 0 à  $n - 1$  **faire**

$nbOcc[tab[i]] \leftarrow nbOcc[tab[i]] + 1$

$i \leftarrow 0$

**pour**  $v$  variant de  $min$  à  $max$  **faire**

**pour**  $k$  variant de 1 à  $nbOcc[v]$  **faire**

$tab[i] \leftarrow v$

$i \leftarrow i + 1$

# Quelques pièges du calcul de la complexité d'algorithmes

## Quand la complexité dépend de l'instance considérée

Exemples : Tri par insertion, Simplex

- Complexité dans le pire et dans le meilleur des cas
  - Complexité en moyenne
- ↪ Dépend d'une distribution de probabilité donnée sur les instances

## Quand l'algorithme fait appel à l'aléatoire

Exemple : Tri rapide

- Complexité en moyenne

## Quand la complexité dépend des valeurs de paramètres en entrée

Exemples : Tri par dénombrement, Programmation dynamique / sac-à-dos

- Complexité pseudo-polynomiale

## Quand la sortie est composée d'un nombre exponentiel d'éléments

Ex : recherche de motifs fréquents dans une base de cartes combinatoires

- Complexité polynomiale incrémentale (Incremental polynomial time)
- Si le temps entre 2 sorties d'éléments est borné par un polynôme

## Quelques bonnes pratiques pour présenter un algorithme

- Spécifier le problème en termes de
  - Paramètres en entrée et en sortie
  - Préconditions sur les valeurs en entrée
    - ↪ Importantes à identifier car elles peuvent changer la complexité !
  - Postrelations entre les entrées et les sorties
- Trouver le bon niveau de détail pour décrire l'algorithme  
... mais attention aux raccourcis qui "cachent" des opérations complexes
- Etudier la complexité de l'algorithme
- Prouver la correction et la complétude
  - S'appuyer sur les invariants de boucles
  - Envisager tous les cas possibles pour les valeurs en entrées

*"Beware of bugs in the above code ; I have only proved it correct, not tried it."*

D. Knuth

# Plan du cours

## 1 Validation théorique d'algorithmes

- Définitions préliminaires (rappels)
- Classes de complexité
- Illustration : Complexité des problèmes d'appariement de graphes

## 2 Evaluation expérimentale d'algorithmes

## 3 Ingénierie algorithmique

# Complexité des problèmes de décision

## Problèmes de décision :

La sortie et la postrelation sont remplacées par une question binaire sur les paramètres en entrée ( $\rightsquigarrow$  Réponse  $\in \{\text{vrai}, \text{faux}\}$ )

## Exemple : Description du problème de décision *Recherche*

- Entrées = un tableau *tab* contenant *n* entiers et un entier *e*
- Question = Existe-t'il un élément de *tab* qui soit égal à *e* ?

## Complexité d'un problème *X* :

- Complexité du meilleur algo (pas nécessairement connu) résolvant *X* :
  - Chaque algorithme résolvant *X* fournit une borne supérieure
  - On peut trouver des bornes inférieures en analysant le problème
- Si plus grande borne inférieure = plus petite borne supérieure  
Alors la complexité de *X* est connue ; Sinon la complexité est ouverte. . .

# La classe $\mathcal{P}$

## Appartenance d'un problème de décision $X$ à la classe $\mathcal{P}$ :

- $X \in \mathcal{P}$  s'il existe un algorithme Polynomial pour résoudre  $X$   
↪ Complexité en  $\mathcal{O}(n^k)$  avec
  - $n$  = taille des données en entrée de l'instance
  - $k$  = constante indépendante de l'instance
- $\mathcal{P}$  est la classe des problèmes traitables en un temps raisonnable  
↪ *Tractable problems*

## Exemples de problèmes de décision appartenant à $\mathcal{P}$ :

- Déterminer si un entier appartient à un tableau (trié ou pas)
- Déterminer s'il existe un chemin entre 2 sommets d'un graphe
- Déterminer s'il existe un arbre couvrant de coût borné dans un graphe
- ...
- Déterminer si un nombre est premier  
↪ Prime is in  $\mathcal{P}$  [Agrawal - Kayal - Saxena 2002]!

# La classe $\mathcal{NP}$

## Appartenance d'un problème de décision $X$ à la classe $\mathcal{NP}$ :

- $X \in \mathcal{NP}$  s'il existe un algorithme Polynomial pour une machine de Turing Non déterministe
- Autrement dit :  $X \in \mathcal{NP}$  si pour toute instance  $I$  de  $X$  telle que  $\text{réponse}(I) = \text{vrai}$ , il existe un certificat  $c(I)$  permettant de vérifier en temps polynomial que  $\text{réponse}(I) = \text{vrai}$   
 $\leadsto$  Il existe un algo polynomial pour vérifier qu'un candidat est solution

## Exemple : SAT $\in \mathcal{NP}$

- Description du problème SAT (rappel) :
  - Entrées = une formule booléenne  $F$  portant sur un ensemble  $X$  de  $n$  variables booléennes
  - Question = Existe-t'il une valuation des var. de  $X$  qui satisfait  $F$  ?
- Certificat : une valuation  $V : X \rightarrow \{\text{vrai}, \text{faux}\}$  qui satisfait  $F$

## Relation entre $\mathcal{P}$ et $\mathcal{NP}$ :

- $\mathcal{P} \subseteq \mathcal{NP}$
- Conjecture :  $\mathcal{P} \neq \mathcal{NP}$   
1 million de dollars à gagner (cf [www.claymath.org/millennium-problems](http://www.claymath.org/millennium-problems))

## Problèmes $\mathcal{NP}$ -complets :

- Les problèmes les plus difficiles de la classe  $\mathcal{NP}$  :  
 $\leadsto X$  est  $\mathcal{NP}$ -complet si  $(X \in \mathcal{NP})$  et  $(X \in \mathcal{P} \Rightarrow \mathcal{P} = \mathcal{NP})$
- Théorème de [Cook 1971] : SAT est  $\mathcal{NP}$ -complet
- Depuis 1971, des centaines de problèmes montrés  $\mathcal{NP}$ -complets  
 $\leadsto$  Voir par exemple [Garey et Johnson 1979]

## Démonstration de $\mathcal{NP}$ -complétude :

- Montrer que le problème  $X$  appartient à  $\mathcal{NP}$
- Trouver une réduction polynomiale pour transformer un problème  $\mathcal{NP}$ -complet connu en  $X$

# Exercice 1

## Description du problème Clique :

- Entrées : un graphe  $G = (V, E)$  et un entier positif  $k$
- Question : Existe-t'il  $S \subseteq V$  tel que  $|S| = k$  et  $\forall i, j \in S, i \neq j \Rightarrow (i, j) \in E$

## Montrer que Clique $\in \mathcal{NP}$ :

$\rightsquigarrow$  Certificat ?

## Montrer que Clique est $\mathcal{NP}$ -complet :

$\rightsquigarrow$  Réduction de SAT :

- Donner un algorithme polynomial permettant de transformer n'importe quelle instance  $I_1$  de SAT en une instance  $I_2$  de Clique
- Montrer que  $\text{réponse}(I_1) = \text{réponse}(I_2)$

## Exercice 2

### Description du problème Somme Sous-ensemble :

- Entrées : un ensemble  $S$  de  $n$  entiers
- Question : Existe-t'il un sous-ensemble non vide de  $S$  dont la somme des éléments soit égale à 0

### Montrer que Somme Sous-ensemble $\in \mathcal{NP}$ :

↪ Certificat ?

### Montrer que Somme-Sous-ensemble est $\mathcal{NP}$ -complet :

↪ Réduction de 3-SAT :

- Précondition de 3-SAT : toute clause de  $F$  contient 3 littéraux
- Donner un algo polynomial permettant de transformer n'importe quelle instance  $I_1$  de 3-SAT en une instance  $I_2$  de Somme Sous-ensemble
- Montrer que réponse( $I_1$ ) = réponse( $I_2$ )

$$\text{Ex. : } F = (\neg a \vee c \vee \neg d) \wedge (\neg b \vee \neg c \vee \neg d) \wedge (a \vee \neg b \vee d)$$

	a	b	c	d	$c_1$	$c_2$	$c_3$
$i_a$	1	0	0	0	0	0	1
$i_{\neg a}$	1	0	0	0	1	0	0
$i_b$		1	0	0	0	0	0
$i_{\neg b}$		1	0	0	0	1	1
$i_c$			1	0	1	0	0
$i_{\neg c}$			1	0	0	1	0
$i_d$				1	0	0	1
$i_{\neg d}$				1	1	1	0
$i_{c_1}^1$					1	0	0
$i_{c_1}^2$					2	0	0
$i_{c_2}^1$						1	0
$i_{c_2}^2$						2	0
$i_{c_3}^1$							1
$i_{c_3}^2$							2
$t$	-1	1	1	1	4	4	4

$$\{\neg a, \neg b, c, \neg d\}$$

$$\Leftrightarrow$$

$$\begin{array}{r}
 1000100 \\
 + 100011 \\
 + 10100 \\
 + 1110 \\
 + 100 \\
 + 20 \\
 + 1 \\
 + 2 \\
 - 1111444 \\
 \hline
 = 0
 \end{array}$$

## Exercice 2 (suite)

### Algorithme pour résoudre Somme Sous-ensemble :

**Entrées :**  $n$  entiers  $x_1, x_2, \dots, x_n$

**Sorties :** vrai s'il existe  $S \subseteq \{x_1, \dots, x_n\}$  tel que  $S \neq \emptyset$  et  $(\sum_{k \in S} k) = 0$

/\* Soit  $min$  la somme des valeurs négatives de  $\{x_1, \dots, x_n\}$  \*/

/\* Soit  $max$  la somme des valeurs positives de  $\{x_1, \dots, x_n\}$  \*/

/\* Soit  $tab[1..n][min..max]$  un tableau de booléens tel que \*/

/\*  $tab[i][v]=vrai$  ssi  $\exists S \subseteq \{x_1, \dots, x_i\}$  tel que  $S \neq \emptyset$  et  $(\sum_{k \in S} k) = v$  \*/

**début**

Initialiser tous les éléments de  $tab$  à *faux*

$tab[1][x_1] \leftarrow vrai$

**pour**  $i$  variant de 2 à  $n$  **faire**

**pour**  $v$  variant de  $min$  à  $max$  **faire**

$tab[i][v] \leftarrow tab[i-1][v] \parallel (x_i == v) \parallel tab[i-1][v-x_i]$

**retourner**  $tab[n][0]$

### Calculer la complexité de l'algorithme :

A-t-on montré que  $\mathcal{P} = \mathcal{NP}$  ?

## Exercice 2 (suite)

### Algorithme pour résoudre Somme Sous-ensemble :

**Entrées :**  $n$  entiers  $x_1, x_2, \dots, x_n$

**Sorties :** vrai s'il existe  $S \subseteq \{x_1, \dots, x_n\}$  tel que  $S \neq \emptyset$  et  $(\sum_{k \in S} k) = 0$

/\* Soit  $min$  la somme des valeurs négatives de  $\{x_1, \dots, x_n\}$  \*/

/\* Soit  $max$  la somme des valeurs positives de  $\{x_1, \dots, x_n\}$  \*/

/\* Soit  $tab[1..n][min..max]$  un tableau de booléens tel que \*/

/\*  $tab[i][v]=vrai$  ssi  $\exists S \subseteq \{x_1, \dots, x_i\}$  tel que  $S \neq \emptyset$  et  $(\sum_{k \in S} k) = v$  \*/

**début**

Initialiser tous les éléments de  $tab$  à *faux*

$tab[1][x_1] \leftarrow vrai$

**pour**  $i$  variant de 2 à  $n$  **faire**

**pour**  $v$  variant de  $min$  à  $max$  **faire**

$tab[i][v] \leftarrow tab[i-1][v] \parallel (x_i == v) \parallel tab[i-1][v-x_i]$

**retourner**  $tab[n][0]$

### Calculer la complexité de l'algorithme :

A-t-on montré que  $\mathcal{P} = \mathcal{NP}$  ?

**Non : L'algorithme est Pseudo-polynomial !**

## Pb fortement $\mathcal{NP}$ -complets et Complexité paramétrée

### Problèmes fortement $\mathcal{NP}$ -complets :

Problèmes qui restent  $\mathcal{NP}$ -complets même quand toutes les valeurs numériques en entrée sont bornées par un polynôme en la taille de l'entrée

### Problèmes paramétrés :

Problèmes dont un paramètre  $k$  en entrée est fixé

### Exemple : Somme sous-ensemble paramétré

- Entrées : un ensemble  $S$  de  $n$  entiers
- Paramètre :  $k = \max - \min$  avec  $\min = \sum_{i \in S, i < 0} i$  et  $\max = \sum_{i \in S, i > 0} i$
- Question : Existe-t'il  $X \subseteq S$  tel que  $X \neq \emptyset$  et  $(\sum_{i \in X} i) = 0$

### Problèmes *Fixed Parameter Tractable* (FPT) :

Problèmes pour lesquels il existe un algo en  $\mathcal{O}(f(k) \cdot n^l)$  où  $n$  = taille des entrées,  $k$  = paramètre fixé et  $l$  = constante indpdte des données en entrée

# Problèmes $\mathcal{NP}$ -difficiles

## Problèmes au moins aussi difficiles que ceux de $\mathcal{NP}$ :

- $X$  est  $\mathcal{NP}$ -difficile si :  $X \in \mathcal{P} \Rightarrow \mathcal{P} = \mathcal{NP}$   
 $\rightsquigarrow$  Vérifier qu'une solution de  $X$  est correcte peut être un pb difficile
- $\mathcal{NP}$ -complet  $\subset$   $\mathcal{NP}$ -difficile

## Exemple 1 : Problème du $k^{\text{ième}}$ sous-ensemble

- Entrées : un ensemble  $S$  de  $n$  entiers et 2 entiers  $k$  et  $l$
- Question : Existe-t'il  $k$  sous-ensembles distincts de  $S$  dont la somme des éléments soit supérieure à  $l$ ?

Ce problème appartient-il à  $\mathcal{NP}$  ?

## Exemple 2 : Problème de la clique exacte

- Entrées : un graphe  $G = (V, E)$  et un entier positif  $k$
- Question : La plus grande clique de  $G$  est-elle de taille  $k$  ?

Ce problème appartient-il à  $\mathcal{NP}$  ?

# La classe $\text{co-NP}$

## Problème complémentaire $\bar{X}$ d'un problème de décision $X$

- Entrées de  $\bar{X}$  = Entrées de  $X$
- Question de  $\bar{X}$  = Négation de la question de  $X$

## Exemple 1 : $\overline{\text{SAT}}$ = Complémentaire de SAT

- Entrées : un ensemble de variables  $X$  et une formule booléenne  $F$
- Question : Est-ce qu'aucune valuation de  $X$  satisfait  $F$  ?

## Exemple 2 : $\overline{\text{Clique}}$ = Complémentaire de Clique

- Entrées : un graphe  $G = (V, E)$  et un entier positif  $k$
- Question : Est-ce que toutes les cliques de  $G$  ont une taille  $\neq$  de  $k$  ?

## Classe $\text{co-NP}$ :

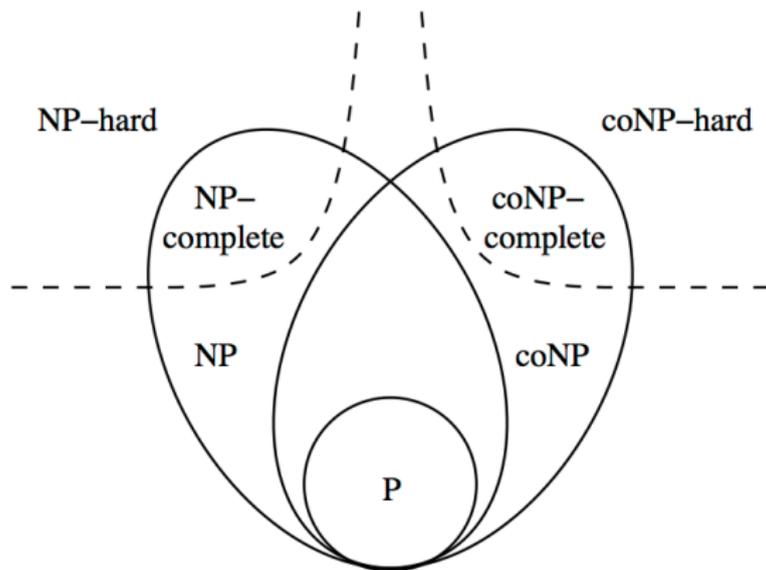
Classe des problèmes dont le complémentaire appartient à  $\text{NP}$

$\leadsto$   $\overline{\text{SAT}}$  et  $\overline{\text{Clique}}$  appartiennent à  $\text{co-NP}$

# Relations entre les classes $\mathcal{NP}$ et $\text{co-}\mathcal{NP}$

## Théorèmes :

- Si  $A \in \mathcal{P}$  Alors  $\bar{A} \in \mathcal{P}$
- S'il existe un pb  $\mathcal{NP}$ -complet  $A$  tq  $\bar{A} \in \mathcal{NP}$  alors  $\mathcal{NP} = \text{co-}\mathcal{NP}$
- Si  $A$  est  $\mathcal{NP}$ -complet alors  $\bar{A}$  est  $\text{co-}\mathcal{NP}$ -complet



# Problèmes indécidables

~ Problèmes pour lesquels il n'existe pas d'algo pour une machine de Turing

## Exemple 1 : Problème de l'arrêt

- Entrée : Un programme  $P$  (une suite de caractères)
- Question : Est-ce que l'exécution de  $P$  se termine en un temps fini ?

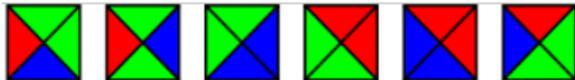
## Exemple 2 : Problème de Post

- Entrée : 2 listes finies  $\alpha_1, \alpha_2, \dots, \alpha_n$  et  $\beta_1, \beta_2, \dots, \beta_n$  de mots
- Question :  $\exists k$  indices  $i_1, i_2, \dots, i_k$  tq  $\alpha_{i_1} \alpha_{i_2} \dots \alpha_{i_n} = \beta_{i_1} \beta_{i_2} \dots \beta_{i_n}$  ?
- Exemple d'instance : Entrée =

$\alpha_1$	$\alpha_2$	$\alpha_3$	$\beta_1$	$\beta_2$	$\beta_3$
a	ab	bba	baa	aa	bb

## Exemple 3 : Problème de pavage

- Entrée : Un ensemble fini  $S$  de carrés aux arêtes coloriées
- Question : Peut-on paver n'importe quel cadre  $n \times n$  avec des copies de carrés de  $S$  de sorte que 2 arêtes adjacentes soient de même couleur ?
- Exemple d'instance : Entrée =



# Plan du cours

## 1 Validation théorique d'algorithmes

- Définitions préliminaires (rappels)
- Classes de complexité
- Illustration : Complexité des problèmes d'appariement de graphes

## 2 Evaluation expérimentale d'algorithmes

## 3 Ingénierie algorithmique

# Pourquoi appairer des graphes ?

## Les graphes sont utilisés pour modéliser :

- des images ;
- des objets 3D ;
- des molécules ;
- des réseaux d'interaction ;
- des réseaux sociaux ;
- des ontologies ;
- des documents (XML) ;
- des ressources sur le Web (RDF) ;
- ...

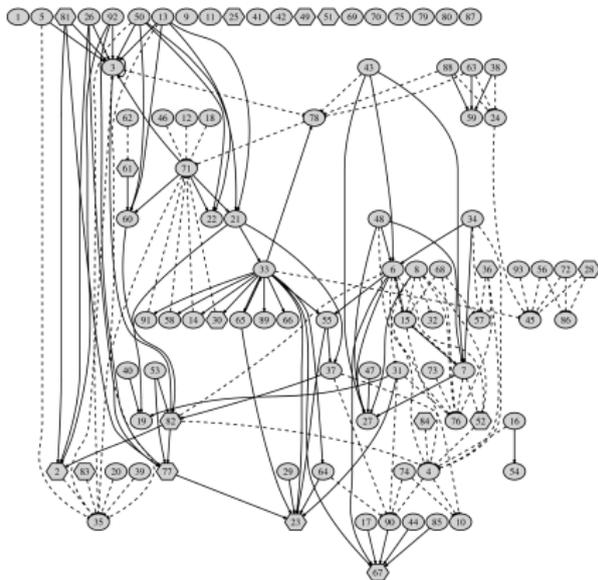
## Besoin d'appairer les graphes pour les comparer :

↪ Equivalence, Inclusion, Intersection, Distance, Similarité, ...

# Exemple 1 : Bio-informatique [LIRIS/Beagle]

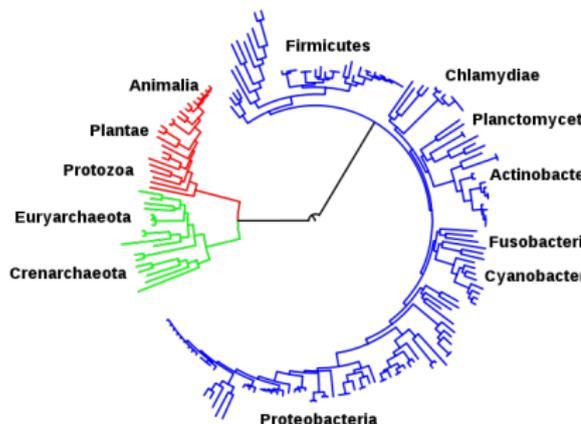
## Réseaux de régulation génétiques

- Sommets = gènes
- Arcs = influence entre gènes



## Arbres phylogénétiques

- Sommets = espèces
- Arêtes = parenté

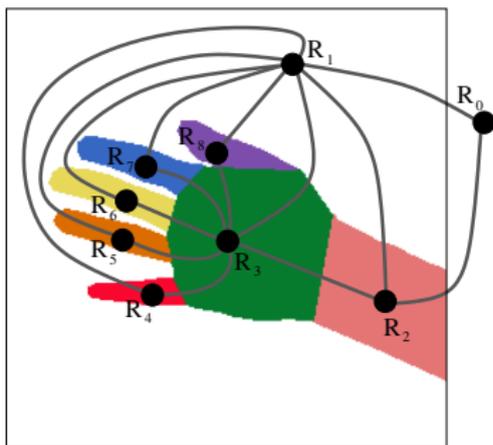




## Exemple 3 : Image [LIRIS/M2DisCo,Imagine]

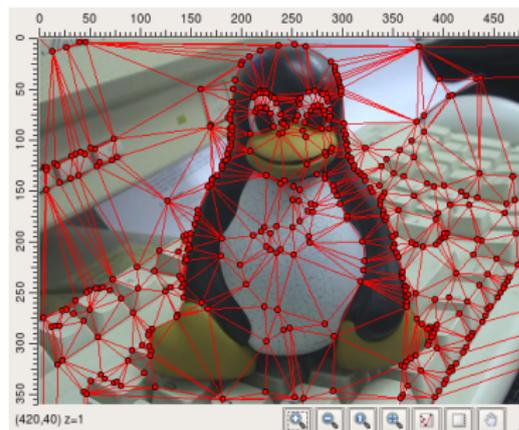
### Graphes d'adjacence de régions

- Sommets = régions
- Arêtes = relations d'adjacence



### Triangulations

- Sommets = points d'intérêt / mots
- Arêtes = triangulation



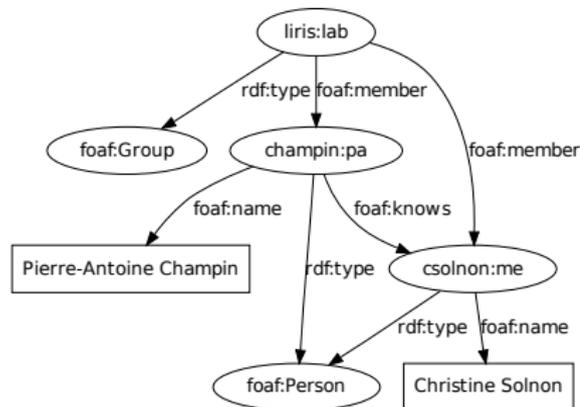
...mais aussi graphes de proximité, maillages, ...

## Exemple 4 : Web sémantique [LIRIS/BD, M2DisCo, SILEX]

### Resource Description Framework (RDF)

- Ensemble de recommandations du W3C
- Information représentée par des triplets (*sujet, prédicat, objet*)  
 ~> Arc de *sujet* vers *objet* étiqueté par *prédicat*

liris :lab	rdf :type	foaf :Group
liris :lab	foaf :member	champin :pa
liris :lab	foaf :member	csolnon :me
champin :pa	foaf :name	"P.-A. Champin"
champin :pa	rdf :type	foaf :Person
champin :pa	foaf :knows	csolnon :me
csolnon :me	rdf :type	foaf :Person
csolnon :me	foaf :name	"C. Solnon"

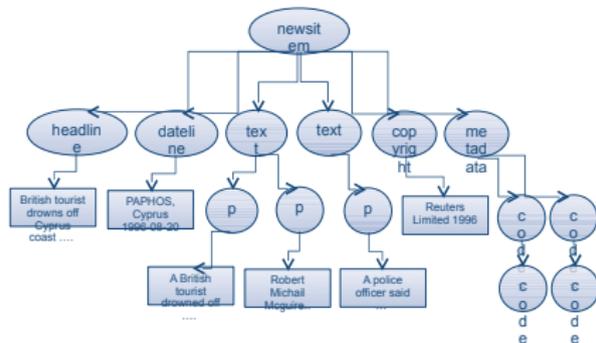


# Exemple 5 : Documents [LIRIS/DRIM]

## Arborescence d'un document XML

- Sommets internes = tags
- Feuilles = texte
- Arcs = relations d'inclusion

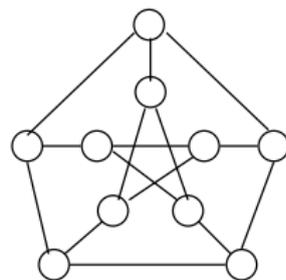
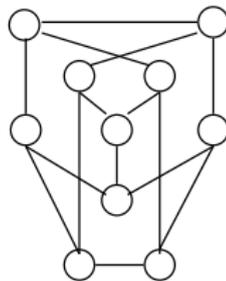
```
<newsitem class="GCAT" date="1996-08-20" id="root" itemid="3597"
xml:lang="en">
  <headline> British tourist drowns off Cyprus coast.</headline>
  <dateline>PAPHOS, Cyprus 1996-08-20</dateline>
  <text>
    <p> A British tourist drowned off the coast of Cyprus on Tuesday while
swimming with his wife and daughter, police in the resort town of Paphos
said.</p>
    <p> Robert Michail Mcguire, 47, from London, was staying with his
family at Venus Hotel in the west coast town of Paphos.</p>
  </text>
  <text>
    <p> A police officer said the sea was rough when the family went for
an evening swim. "When people see large waves they don't think of their
safety they're just concerned with enjoying themselves," the police officer
said.</p>
  </text>
  <copyright>(c) Reuters Limited 1996</copyright>
  <metadata>
    <codes class="bip:countries:1.0">
      <code code=" UK"></code></codes>
    <codes class="bip:topics:1.0">
      <code code="GCAT"></code></codes>
  </newsitem>
```



# Problèmes d'appariement de graphes

- Isomorphisme  $\leadsto$  Equivalence
- Sous-isomorphisme  $\leadsto$  Inclusion
- Plus grand sous-graphe commun  $\leadsto$  Intersection
- Distance d'édition  $\leadsto$  Coût de transformation

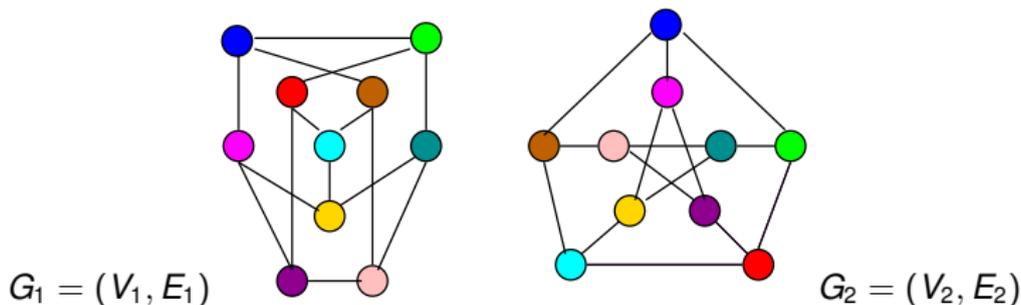
$G_1 = (V_1, E_1)$



$G_2 = (V_2, E_2)$

# Problèmes d'appariement de graphes

- Isomorphisme  $\leadsto$  Equivalence
- Sous-isomorphisme  $\leadsto$  Inclusion
- Plus grand sous-graphe commun  $\leadsto$  Intersection
- Distance d'édition  $\leadsto$  Coût de transformation



$\leadsto \exists$  bijection  $f : V_1 \rightarrow V_2$  tq  $\forall i, j \in V_1, i \neq j : (i, j) \in E_1 \Leftrightarrow (f(V_1), f(V_2)) \in E_2$

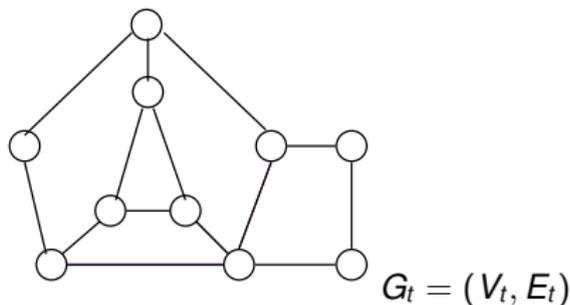
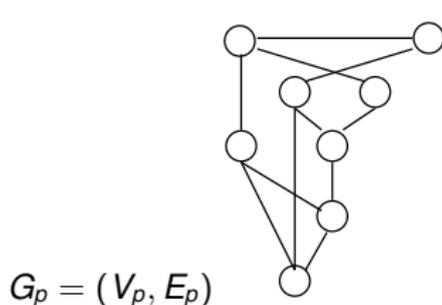
$\leadsto$  Complexité dans le cas général : Isomorphe-complet !

$\leadsto$  Cas particuliers :

- Algorithme polynomial : Graphes planaires, Graphes ordonnés, ...
- Algorithme FPT : Degré borné, Profondeur d'arbre bornée, ...

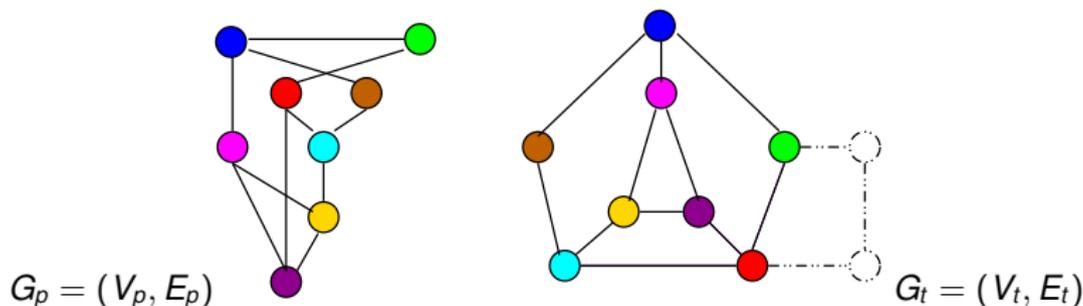
# Problèmes d'appariement de graphes

- Isomorphisme  $\leadsto$  Equivalence
- Sous-isomorphisme  $\leadsto$  Inclusion
- Plus grand sous-graphe commun  $\leadsto$  Intersection
- Distance d'édition  $\leadsto$  Coût de transformation



# Problèmes d'appariement de graphes

- Isomorphisme  $\leadsto$  Equivalence
- Sous-isomorphisme  $\leadsto$  Inclusion
- Plus grand sous-graphe commun  $\leadsto$  Intersection
- Distance d'édition  $\leadsto$  Coût de transformation



$\leadsto \exists$  injection  $f : V_p \rightarrow V_t$  telle que

- Cas induit :  $\forall i, j \in V_p, i \neq j : (i, j) \in E_p \Leftrightarrow (f(i), f(j)) \in E_t$
- Cas partiel :  $\forall i, j \in V_p, i \neq j : (i, j) \in E_p \Rightarrow (f(i), f(j)) \in E_t$

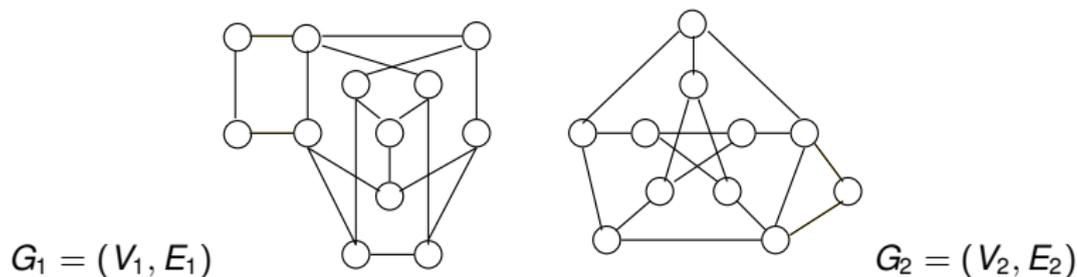
$\leadsto$  Complexité dans le cas général :  $\mathcal{NP}$ -complet

$\leadsto$  Cas particuliers :

- Algorithme polynomial : Arbres, Graphes outerplanar 2-connexes, ...
- Algorithme FPT :  $(|V_p|, \text{treeWidth}(G_t))$  borné

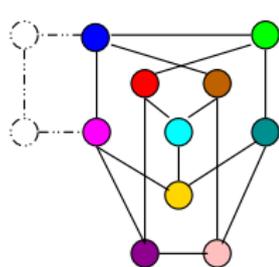
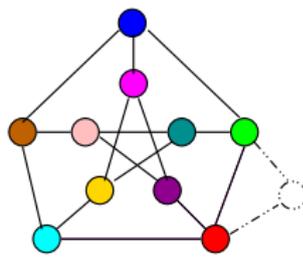
# Problèmes d'appariement de graphes

- Isomorphisme  $\leadsto$  Equivalence
- Sous-isomorphisme  $\leadsto$  Inclusion
- Plus grand sous-graphe commun  $\leadsto$  Intersection
- Distance d'édition  $\leadsto$  Coût de transformation



# Problèmes d'appariement de graphes

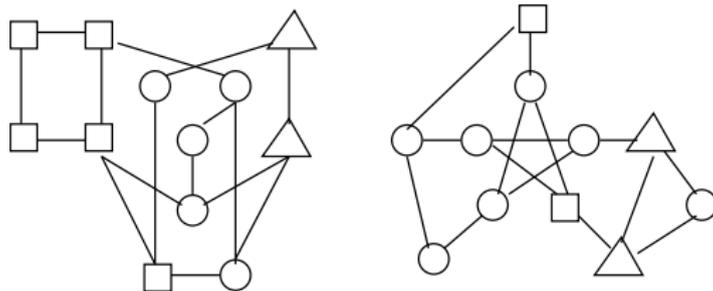
- Isomorphisme  $\leadsto$  Equivalence
- Sous-isomorphisme  $\leadsto$  Inclusion
- Plus grand sous-graphe commun  $\leadsto$  Intersection
- Distance d'édition  $\leadsto$  Coût de transformation


 $G_1 = (V_1, E_1)$ 

 $G_2 = (V_2, E_2)$ 

- $\leadsto \exists$  injection partielle  $f : V_1 \rightarrow V_2 \cup \{\epsilon\}$  telle que
  - Cas induit : Maximiser  $|\{i \in V_1, f(i) \neq \epsilon\}|$  sous la contrainte  $\forall i, j \in V_1, f(i) \neq \epsilon, f(j) \neq \epsilon, i \neq j : (i, j) \in E_1 \Leftrightarrow (f(i), f(j)) \in E_2$
  - Cas partiel : Maximiser  $|\{(i, j) \in E_1, (f(i), f(j)) \in E_2\}|$
- $\leadsto$  Complexité dans le cas général :  $\mathcal{NP}$ -difficile
- $\leadsto$  Cas particulier :
  - Algorithme polynomial : Arbres
  - Algorithme FPT : Degré borné pour les graphes outerplanar

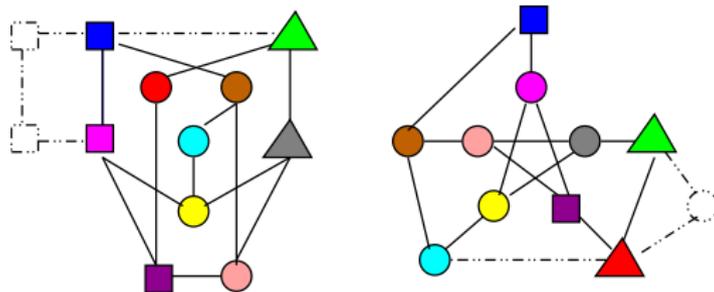
## Problèmes d'appariement de graphes

- Isomorphisme  $\leadsto$  Equivalence
- Sous-isomorphisme  $\leadsto$  Inclusion
- Plus grand sous-graphe commun  $\leadsto$  Intersection
- Distance d'édition  $\leadsto$  Coût de transformation



## Problèmes d'appariement de graphes

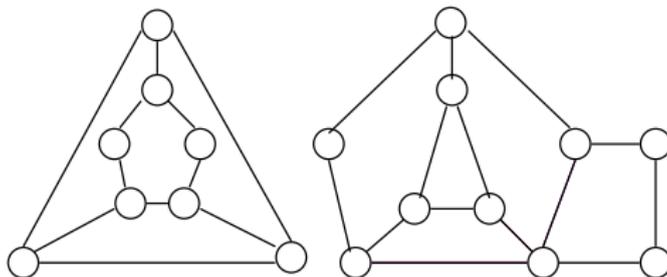
- Isomorphisme  $\leadsto$  Equivalence
- Sous-isomorphisme  $\leadsto$  Inclusion
- Plus grand sous-graphe commun  $\leadsto$  Intersection
- Distance d'édition  $\leadsto$  Coût de transformation



- $\leadsto$  Appariement qui minimise des coûts d'édition
- $\leadsto$  Complexité dans le cas général :  $\mathcal{NP}$ -difficile
- $\leadsto$  Algorithme polynomial : Chaînes et arbres ordonnés

## Zoom sur les graphes plans et la comparaison d'images

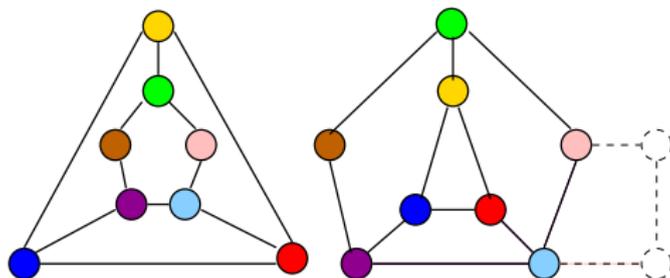
~ Projets SATTIC (ANR blanc 2007-2011) et Solstice (ANR blanc 2013-2017)



Le graphe de gauche est-il inclus dans celui de droite ?

# Zoom sur les graphes plans et la comparaison d'images

~ Projets SATTIC (ANR blanc 2007-2011) et Solstice (ANR blanc 2013-2017)



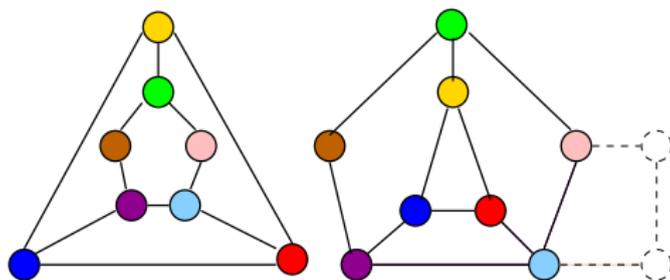
Le graphe de gauche est-il inclus dans celui de droite ?

Oui, mais est-ce qu'ils se ressemblent ?

- Les graphes modélisant des images sont plongés
- Comparons les plongements ~ Cartes combinatoires

# Zoom sur les graphes plans et la comparaison d'images

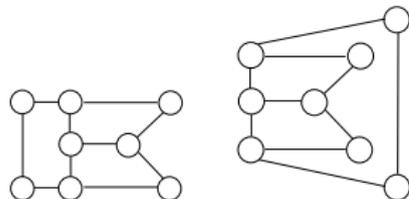
~ Projets SATTIC (ANR blanc 2007-2011) et Solstice (ANR blanc 2013-2017)



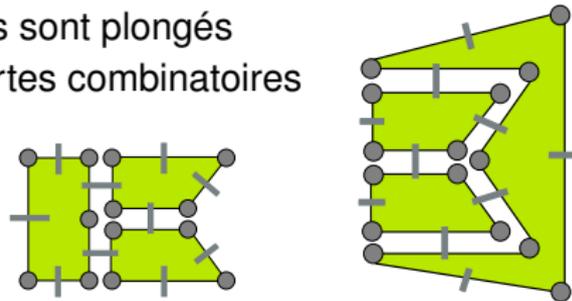
Le graphe de gauche est-il inclus dans celui de droite ?

Oui, mais est-ce qu'ils se ressemblent ?

- Les graphes modélisant des images sont plongés
- Comparons les plongements ~ Cartes combinatoires



2 graphes isomorphes



2 cartes non isomorphes

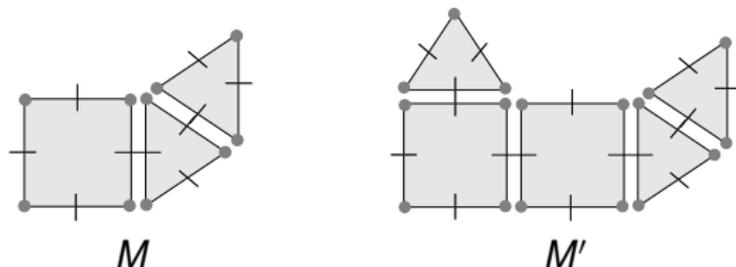
# Algorithm for Submap Isomorphism

## Basic idea :

- Choose a dart  $d$  in the pattern map  $M$
- For each dart  $d'$  in the target map  $M'$  :
  - Traverse  $M$  and  $M'$  in parallel and build a dart matching
  - If the dart matching is a subisomorphism then answer Yes
- Answer No

**Complexity :**  $\mathcal{O}(|darts(M)| \cdot |darts(M')|)$

**Precondition :** The pattern map  $M$  must be connected



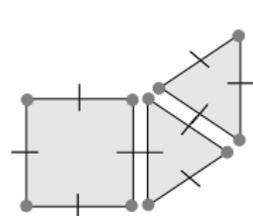
# Algorithm for Submap Isomorphism

## Basic idea :

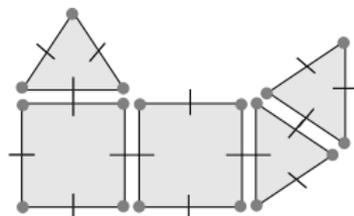
- Choose a dart  $d$  in the pattern map  $M$
- For each dart  $d'$  in the target map  $M'$  :
  - Traverse  $M$  and  $M'$  in parallel and build a dart matching
  - If the dart matching is a subisomorphism then answer Yes
- Answer No

**Complexity :**  $\mathcal{O}(|darts(M)| \cdot |darts(M')|)$

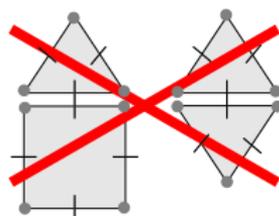
**Precondition :** The pattern map  $M$  must be connected



$M$



$M'$



# Submap isomorphism is $\mathcal{NP}$ -complete

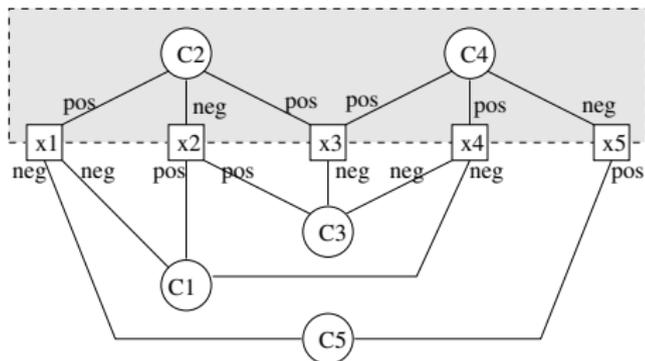
~ Proof by reduction of Separable Planar 3-SAT

## Definition of the $\mathcal{NP}$ -complete pb Separable Planar 3-SAT [Lichtenstein 82]

- Input : A CNF formula  $F$  over a set  $X$  of variables and a plane embedding of the formula graph  $G_{(X,F)}$  such that
  - Every clause of  $F$  contains 2 or 3 literals
  - For every variable  $x_i$  :  $d_{pos}^{\circ}(x_i) \geq 1$ ,  $d_{neg}^{\circ}(x_i) \geq 1$ ,  $d^{\circ}(x_i) \leq 3$
  - The cycle  $(x_1, x_2, \dots, x_n, x_1)$  separates the plane in two parts *in* and *out* such that for every variable  $x_i$  :  $\{pos(x_i), neg(x_i)\} = \{in(x_i), out(x_i)\}$
- Question : Does there exist a truth assignment for  $X$  which satisfies  $F$  ?

### Example of instance :

$$\begin{aligned}
 X &= \{x_1, x_2, x_3, x_4, x_5\} \\
 F &= (\neg x_1 \vee x_2 \vee \neg x_4) \\
 &\wedge (x_1 \vee \neg x_2 \vee x_3) \\
 &\wedge (x_2 \vee \neg x_3 \vee \neg x_4) \\
 &\wedge (x_3 \vee x_4 \vee \neg x_5) \\
 &\wedge (\neg x_1 \vee x_5)
 \end{aligned}$$



# Reduction of Sep. Planar 3-SAT to Submap Isomorphism

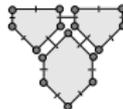
## Goal of the reduction :

- Let  $F$  be a boolean formula with  $v$  variables and  $c$  clauses
- Give a polynomial time algorithm for building two maps  $M$  and  $M'$  such that  $F$  is satisfiable iff  $M$  is a submap of  $M'$

## Basic idea : Associate gadgets with variables and clauses

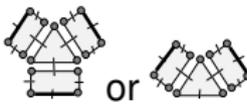
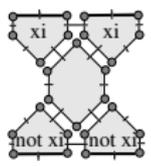
- The pattern map  $M$  is composed of

$c$  clause gadgets  and  $v$  variable gadgets



- The target map  $M'$  is derived from the plane graph associated with  $F$  :

- Replace vertices by gadgets :

clause gadgets  or  and variable gadgets 

- 2-sew patterns associated with adjacent vertices

Submap isomorphism  $\Leftrightarrow$  Selection of 1 satisfied literal for each clause

# Example

$$X = \{x_1, x_2, x_3, x_4, x_5\}$$

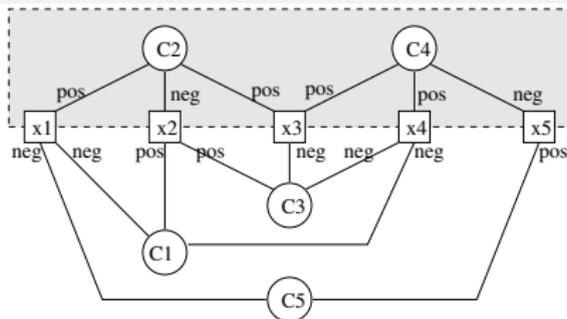
$$F = (\neg x_1 \vee x_2 \vee \neg x_4)$$

$$\wedge (x_1 \vee \neg x_2 \vee x_3)$$

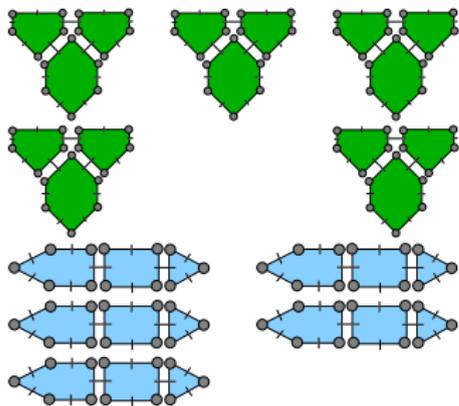
$$\wedge (x_2 \vee \neg x_3 \vee \neg x_4)$$

$$\wedge (x_3 \vee x_4 \vee \neg x_5)$$

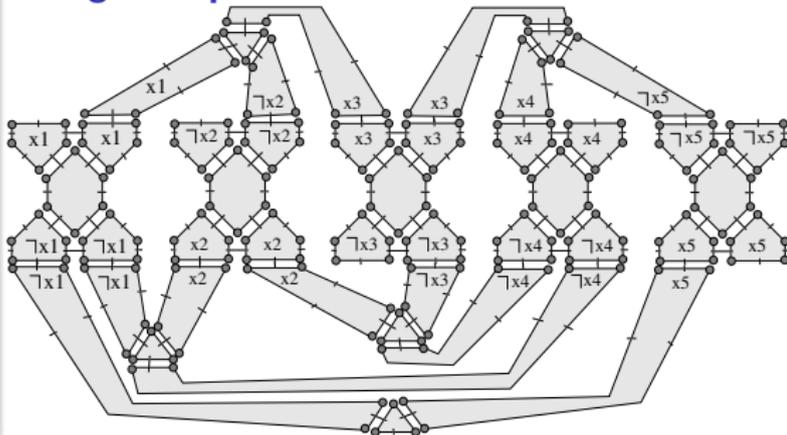
$$\wedge (\neg x_1 \vee x_5)$$



Pattern map  $M$  :



Target map  $M'$  :



# Example

$$X = \{x_1, x_2, x_3, x_4, x_5\}$$

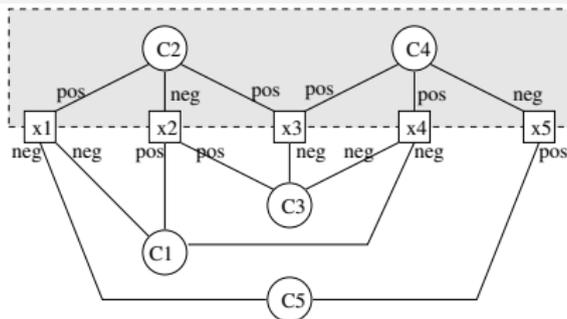
$$F = (\neg x_1 \vee x_2 \vee \neg x_4)$$

$$\wedge (x_1 \vee \neg x_2 \vee x_3)$$

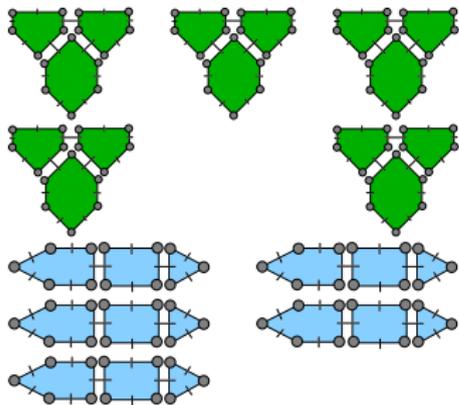
$$\wedge (x_2 \vee \neg x_3 \vee \neg x_4)$$

$$\wedge (x_3 \vee x_4 \vee \neg x_5)$$

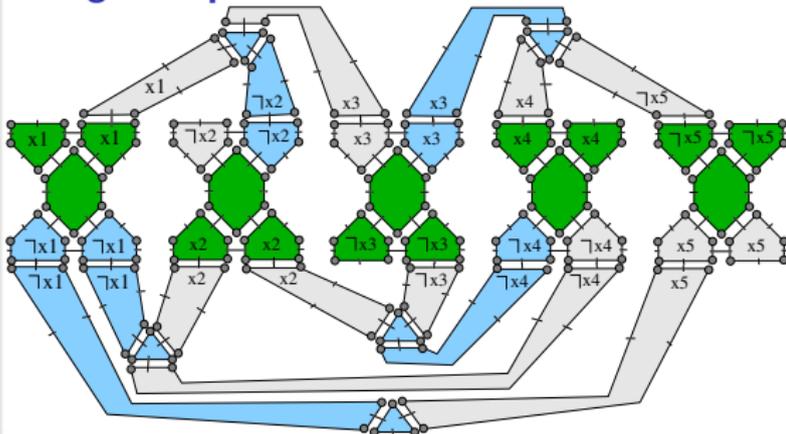
$$\wedge (\neg x_1 \vee x_5)$$



Pattern map  $M$  :



Target map  $M'$  :



# FPT algorithm for Submap isomorphism

## Parameterized submap isomorphism :

- Input : Two maps  $M$  and  $M'$
- Question : Does there exist a submap of  $M'$  which is isomorphic to  $M$  ?
- Parameter : The number  $k$  of connected components in  $M$

## FPT Algorithm :

Decompose  $M$  into its  $k$  connected components denoted  $M_1, \dots, M_k$

Let  $V$  and  $E$  be two empty sets

**for** each connected component  $M_i$  of  $M$  **do**

**for** each partial submap  $M'_x$  of  $M'$  which is isomorphic to  $M_i$  **do**  
         add  $(M'_x, i)$  to  $V$

**for** each  $(M'_x, i) \in V$  **do**

**for** each  $(M'_y, j) \in V$  such that  $i \neq j$  **do**  
         **if**  $Darts(M'_x) \cap (Darts'_y) = \emptyset$  **then** add  $((M'_x, i), (M'_y, j))$  to  $E$ ;

**if** the graph  $G = (V, E)$  has a clique of size  $k$  **then return** *True* **else return** *False*

## Complexity of the FPT algorithm :

Let  $d = |Darts(M)|$ ,  $d_i = |Darts(M_i)|$  and  $d' = |Darts(M')|$  :

- Decomposition in connected components by a traversal of  $M$  in  $\mathcal{O}(d)$
- Construction of  $V$  in  $\mathcal{O}(dd')$
- Construction of  $E$  in  $\mathcal{O}(dd'^2)$
- Search for a clique of size  $k$  in  $\mathcal{O}(d'^k)$

↪ Overall time complexity in  $\mathcal{O}(d'^k + dd'^2)$ .

## Conclusion :

The tractability of submap isomorphism depends on the number of connected components in the pattern map

# What about Maximum Common Submap (MCS) ?

## Decision problem (Common Submap) associated with MCS :

- Input : 2 maps  $M$  and  $M'$  and an integer  $k$
- Question : Does there exist a map  $M''$  such that
  - $M''$  is isomorphic to submaps of  $M$  and  $M'$
  - $|Darts(M'')| \geq k$

## Reduction of Submap Isomorphism to Common Submap :

To decide whether  $M$  is a submap of  $M'$  : Solve  $MCS(M, M', |Darts(M)|)$

## Complexity of Common Submap :

- $\mathcal{NP}$ - complete in the general case
- What if we add the constraint that  $M''$  must be connected ?  
     $\rightsquigarrow$  Reduction of Separable Planar 3-SAT

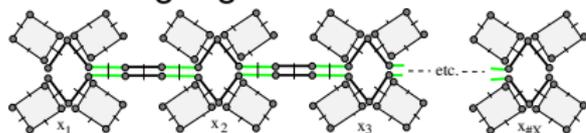
# Reduction of Separable Planar 3-SAT to MCS

## Goal of the reduction for a formula $F$ :

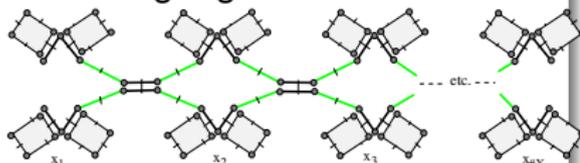
Polynomial algo for building  $k$ ,  $M$  and  $M'$  st  $F$  is satisfiable iff  $\exists$  a connected map  $M''$  isomorphic to submaps of  $M$  and  $M'$  with  $|Darts(M'')| \geq k$

## Basic idea : Associate gadgets with variables and clauses

- Variable gadget in  $M$  :



- Variable gadget in  $M'$  :



- Gadgets associated with clauses in  $M$  and  $M'$  :



Each clause gadget has  $D$  darts where  $D \geq \text{nb of darts in the var gadget}$

- Bound on the size  $k = c \cdot (D + 1)$  where  $c = \text{nb of clauses of } F$

$\exists$  common submap  $M''$  st  $|Darts(M'')| \geq k \Leftrightarrow c$  clauses are satisfied in  $F$

# Example

$$X = \{X_1, X_2, X_3, X_4, X_5\}$$

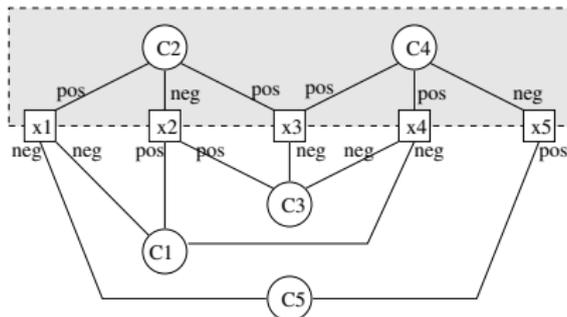
$$F = (\neg X_1 \vee X_2 \vee \neg X_4)$$

$$\wedge (X_1 \vee \neg X_2 \vee X_3)$$

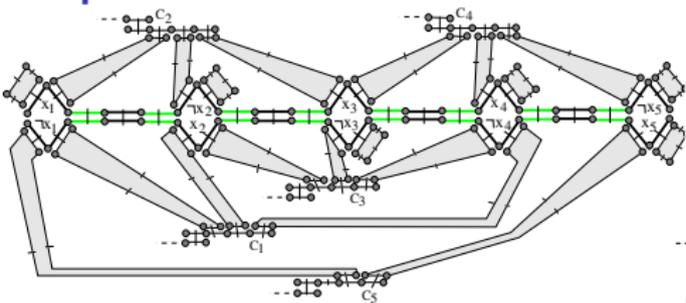
$$\wedge (X_2 \vee \neg X_3 \vee \neg X_4)$$

$$\wedge (X_3 \vee X_4 \vee \neg X_5)$$

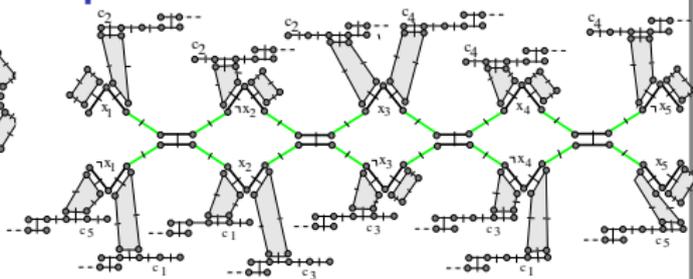
$$\wedge (\neg X_1 \vee X_5)$$



Map  $M$  :



Map  $M'$  :



# Solutions

## Instance SAT :

$$X = \{x_1, x_2, x_3, x_4, x_5\}$$

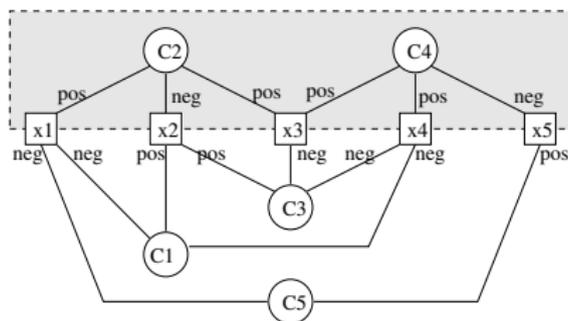
$$F = (\neg x_1 \vee x_2 \vee \neg x_4)$$

$$\wedge (x_1 \vee \neg x_2 \vee x_3)$$

$$\wedge (x_2 \vee \neg x_3 \vee \neg x_4)$$

$$\wedge (x_3 \vee x_4 \vee \neg x_5)$$

$$\wedge (\neg x_1 \vee x_5)$$

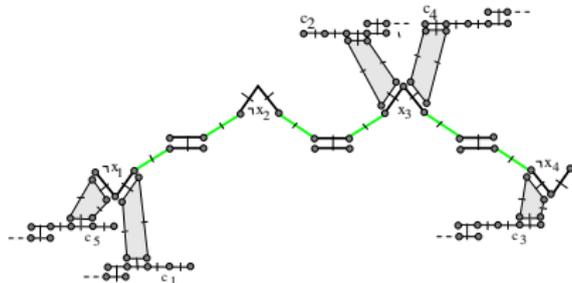


## Sous-carte commune de taille supérieure à $5 * D$ :

- Solution de l'instance SAT :

$$A = \{\neg x_1, \neg x_2, x_3, \neg x_4, x_5\} \text{ OU}$$

$$A = \{\neg x_1, \neg x_2, x_3, \neg x_4, \neg x_5\}$$



# Plan du cours

## 1 Validation théorique d'algorithmes

## 2 Evaluation expérimentale d'algorithmes

- Processus expérimental
- Constitution d'un benchmark
- Choisir les facteurs, *Design Points* et indicateurs de performance
- Analyser les résultats
- Illustration : Comparaison expérimentale de solveurs de contraintes

## 3 Ingénierie algorithmique

# Processus expérimental [McGeoch 2012]

## Etape 1 : Préparer l'expérimentation

- Formuler une question
- Constituer l'environnement de test :  
~ Programme(s), Instances, Outils d'analyse de données, ...
- Concevoir l'expérimentation :  
~ Quelles propriétés mesurer ? Sur quelles instances ? ...

## Etape 2 : Réaliser l'expérimentation

- Lancer les exécutions et collecter les résultats
- Analyser les résultats
  - Si question non répondue, alors retour à l'étape 1
  - Si question répondue et réponse intéressante, alors publier !

## Processus itératif ...

# Ex. : Evaluation d'un nouvel algo pour MaxSAT

## Rappel du problème MaxSAT :

- Entrée : formule booléenne  $F$  composée de  $m$  clauses et  $n$  variables
- Sortie : un entier  $k$
- Postrelation :  $k = \text{nb max de clauses de } F \text{ satisfiables simultanément}$   
(ou  $k = \text{borne inf. du nb max de clauses satisfiables si algo heuristique}$ )

## Exemples de questions :

- Temps moyen pour résoudre une instance ?
- Ecart moyen à la solution optimale / limite donnée de temps ?
- Influence de  $n$  et  $m$  sur les performances ?
- Influence de la structure des instances sur les performances ?
- Pour quelles instances l'algorithme est-il bon (resp. mauvais) ?
- Influence des paramètres de l'algorithme ?
- L'algorithme est-il compétitif avec l'état de l'art ?
- Quelles sont les meilleures struct. de données pour implémenter l'algo ?
- ...

# Deux types d'expérimentations

## Expérimentation exploratoire :

Identifier ce qu'il serait intéressant d'expérimenter :

- Questions pertinentes ?
- Paramètres qui influent sur les performances ?
- Instances difficiles ?
- Compétitif avec l'état de l'art ?
- ...

↪ Cycles rapides, en préparation à l'expérimentation intensive

## Expérimentation intensive :

- Mise en place d'un protocole expérimental efficace et automatisé
  - ↪ Objectifs bien définis
  - ↪ Cycles plus lents (pouvant aller jusque plusieurs mois...)

# Quelques principes généraux :

## Une expérimentation doit être reproductible

- Préciser toutes les infos nécessaires : instances considérées, paramètres, machine, système d'exploitation, nombre d'exécutions, ...
- Si possible : diffuser le code sous licence et/ou une machine virtuelle  
~> <http://www.recomputation.org/manifesto/>



## Une expérimentation doit être efficace

- Automatisation du processus d'expérimentation/analyse des résultats
- Pas d'expérimentations inutiles ...

## Une expérimentation doit être la plus générale possible

- Minimiser les dépendances (implémentation, matériel, ...)

## Une expé. doit produire des résultats intéressants et nouveaux !

# Le vocabulaire de l'expérimentation

**Métrique de performance** : Critère que l'on veut évaluer

↪ Durée d'exécution, Qualité des solutions, Mémoire, ...

**Indicateur de performance** : Qté mesurable pour évaluer une métrique

↪ Si métrique = Durée : Temps CPU, Nb d'op. dominantes, ...

**Paramètre** : Élément qui affecte la valeur d'un indicateur de performance

- Paramètres de l'algorithme

↪ Nb max d'itér. pour un algo de recherche locale, ...

- Paramètres de l'instance

↪ Nb de variables et nb de clauses d'une formule SAT, ...

- Paramètres de l'environnement

↪ Compilateur, OS, machine, ...

**Facteur** : Paramètre explicitement manipulé dans l'expérimentation

**Niveau** : Valeur affectée à un facteur pendant une expérimentation

**Design point** : Combinaison de niveaux à tester pendant une expé.

**Run** : 1 execution sur un *design point* produisant 1 mesure de l'indicateur de performance

# How not to do it [Gent et al 1997] (1/2)

## Getting started

- Don't trust yourself
- Do use different hardware
- Do make it fast enough
- Do report important implementation details
- Do use version control

## Experimental design

- Do measure with many instruments
- Do vary all relevant factors
- Don't change two things at once
- Do measure CPU time
- Do collect all data possible
- Do be paranoid
- Do check your solutions
- Do it all again
- Do use the same instances
- Don't ignore crashes
- Do it often and do it big
- Don't kill your machines
- Do look for scaling results
- Do be stupid

# How not to do it [Gent et al 1997] (2/2)

## Problems with random numbers

- Don't trust your source of random numbers
- Do understand your instance generator
- Do control sources of variation

*Random numbers should not be generated with a method chosen at random*

D. Knuth

## Presentation of results

- Do present statistics
- Do report negative results
- Don't push deadlines
- Do check your references

## Data analysis

- Do look at the raw data
- Do look for good views
- Don't discard data
- Do face up to the consequences of your results
- Don't reject the obvious

# Plan du cours

## 1 Validation théorique d'algorithmes

## 2 Evaluation expérimentale d'algorithmes

- Processus expérimental
- Constitution d'un benchmark
- Choisir les facteurs, *Design Points* et indicateurs de performance
- Analyser les résultats
- Illustration : Comparaison expérimentale de solveurs de contraintes

## 3 Ingénierie algorithmique

# Constitution d'un benchmark

## Le choix des instances dépend des objectifs de l'expérimentation :

- Vérifier que le programme est correct  
~> *Stress-test instances (boundary instances, happy path, ...)*
- Evaluer les performances dans le pire des cas  
~> *Worst-case, bad-case instances*
- Evaluer le passage à l'échelle par rapport à un paramètre des instances  
~> *Random instances*
- Evaluer les performances pour une application réelle  
~> *Real-world instances*
- Comparer le programme avec l'état de l'art  
~> *Public benchmark/testbed*

## Quelle diversité dans les benchmarks ?

- Pour des résultats plus généraux, privilégier des benchmarks variés
- Pour faciliter l'analyse des résultats, privilégier des bench. homogènes  
~> Décomposer le benchmark en classes homogènes pour l'analyse

# Difficulté des instances

## Attention aux effets plancher/plafond !

- Les instances extrêmes ne permettent pas de comparer des algos
  - Trop faciles : tout le monde les résout très vite
  - Trop difficiles : personne ne les résout dans la limite de temps
- Eliminer les instances trop faciles ou trop difficiles
- Faire varier la difficulté des instances

## Facteurs influant sur la difficulté d'une instance :

- Taille de l'instance
- Structure de l'instance
  - ↪ Paramètres structurels (largeur d'arbre, degré max., planarité, ...)
- Constrainedness (pour les problèmes de décision)
  - ↪ Transition de phase
- Distribution des optima globaux (pour les problèmes d'optimisation)
  - ↪ Paysage de recherche

# Transition de phases (1/4)

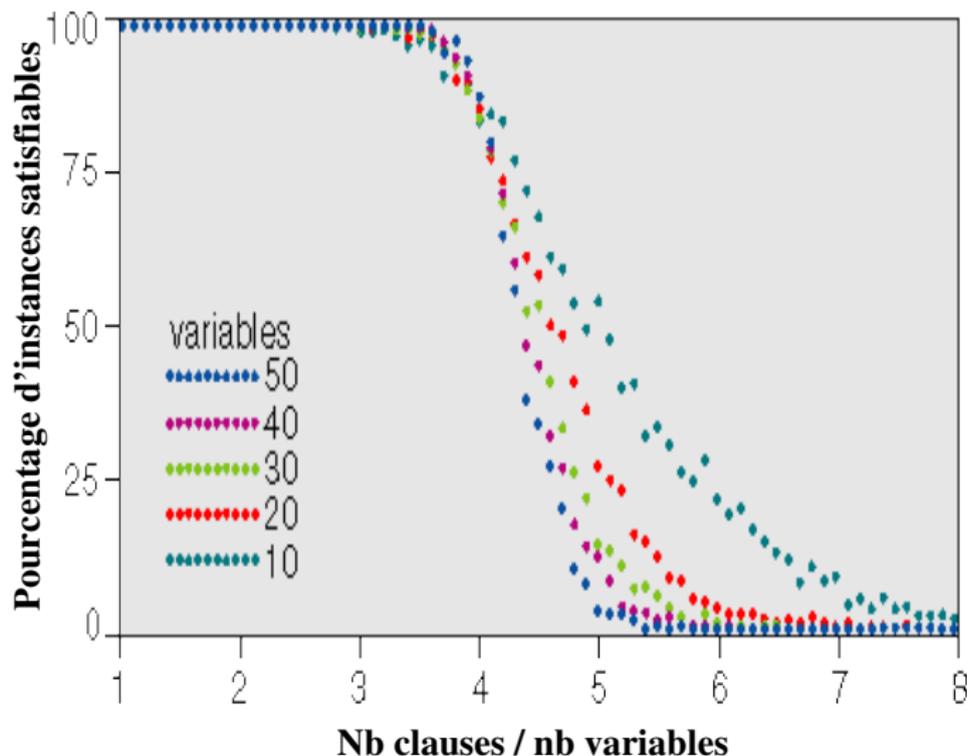
## Exemple : Le problème SAT

- La difficulté dépend du nombre  $n$  de variables...
- ... mais aussi du ratio entre le nombre  $p$  de clauses et  $n$ 
  - Peu de clauses  $\Rightarrow$  Instance sous-contrainte  
 $\leadsto$  Facile (sauf rares exceptions !)
  - Beaucoup de clauses  $\Rightarrow$  Instance sur-contrainte  
 $\leadsto$  Facile
  - Entre les 2  $\Rightarrow$  ça se complique !!!

## Etude expérimentale :

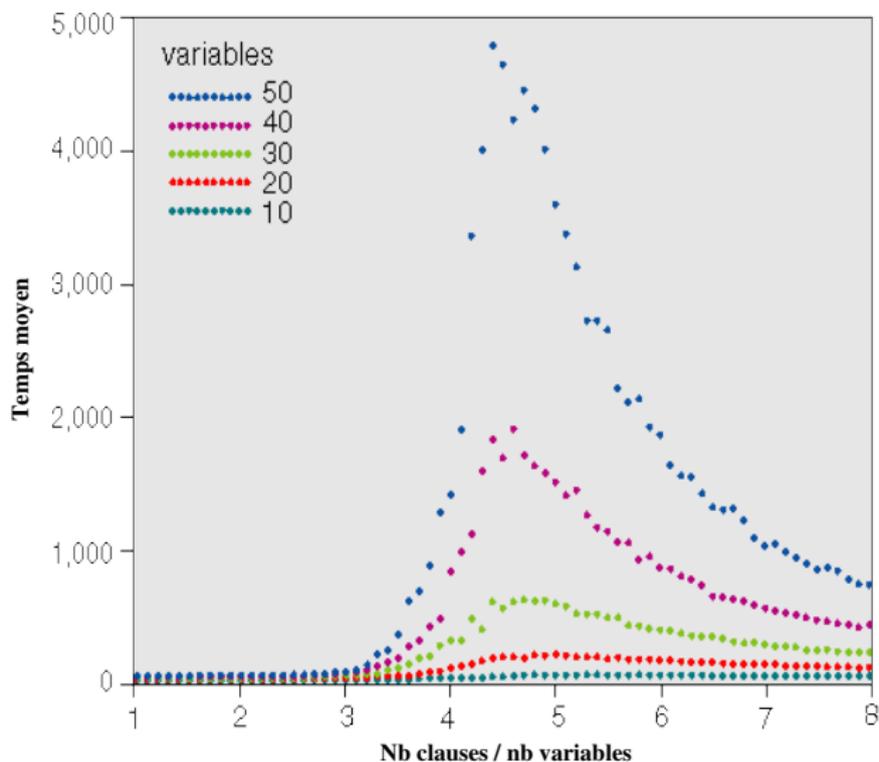
- Génération aléatoire d'instances, en fonction
  - du nombre  $n$  de variables ( $n \in \{10, 20, 30, 40, 50\}$ )
  - du nombre  $p$  de clauses ( $n \leq p \leq 8n$ ) $\leadsto$  Distribution uniforme des variables dans les clauses
- Mesure du taux d'instances satisfiables en fct du ratio  $p/n$
- Mesure du temps de résolution en fct du ratio  $p/n$

## Transition de phases (2/4)



(image empruntée à des transparents de Toby Walsh)

# Transition de phases (3/4)



(image empruntée à des transparents de Toby Walsh)

# Transition de phases (4/4)

## Qu'est-ce qu'une transition de phase ?

- Changmt abrupte de la réponse aux instances en fct de paramètres  
     $\rightsquigarrow$  Pour 3-SAT : quand  $p/n = 4.3$
- Correspond à un pic de difficulté pour la résolution
- Indépendant de l'algorithme utilisé

## Où se trouve la transition de phases ?

- Là où la probabilité d'avoir une réponse positive est de 50%
- Mesurer le taux de constrainedness :  $\kappa = 1 - \frac{\log_2(p)}{n}$  de l'instance, avec
  - $2^n$  = Taille de l'espace de recherche de l'instance
  - $p$  = Nombre estimé de solutions de l'instance
- La transition de phases se trouve là où  $\kappa = 1$  :
  - Si  $\kappa \approx 0$  alors l'instance est sous-contrainte
  - Si  $\kappa \approx 1$  alors l'instance est critique
  - Si  $\kappa \approx \infty$  alors l'instance est sur-contrainte

# Paysage de recherche (1/4)

## Définition (très générale) d'un problème d'optimisation

- Entrée : un ensemble  $S$  de combinaisons et une fct  $f : S \rightarrow \mathbb{R}$
- Sortie : une combinaison  $s^* \in S$  telle que  $\forall s \in S, f(s) \leq f(s^*)$

## Graphe de voisinage $G = (S, N)$ associé à un algorithme $A$

- Sommets :  $S =$  ensemble des combinaisons à explorer
- Arcs :  $N = \{(s_i, e_j) \in S \times S, A \text{ peut visiter } s_j \text{ à partir de } s_i\}$   
 $\rightsquigarrow s_j$  est une combinaison voisine de  $s_i$
- Notation : voisinage de  $s_i = N(s_i) = \{s_j / (s_i, s_j) \in N\}$

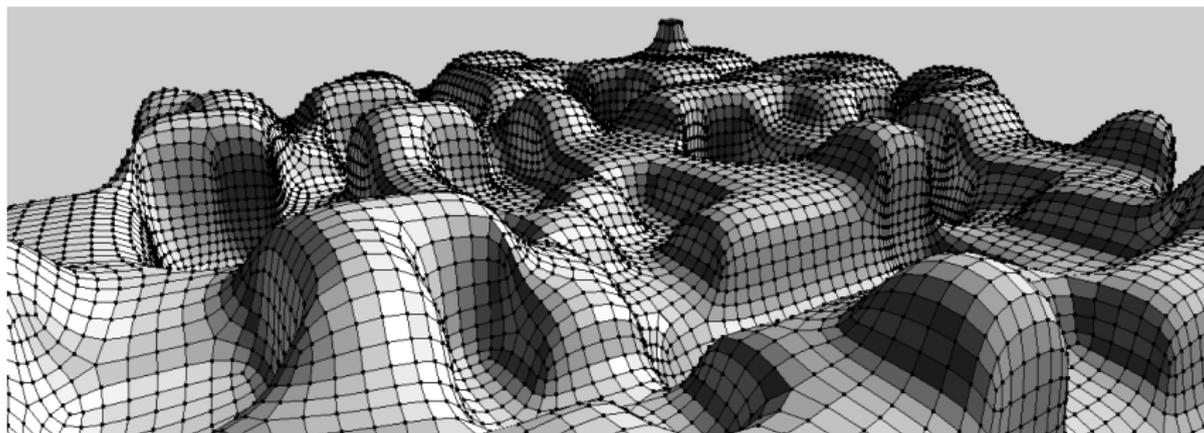
## Paysage de recherche =

- un problème d'optimisation  $P = (S, f)$
- + un graphe de voisinage  $G = (S, N)$

# Paysage de recherche (2/4)

## Représentation graphique d'un paysage de recherche

- Chaque configuration de l'espace de recherche  $S = 1$  point
- La fonction objectif  $f$  donne l'altitude des points
- Le voisinage  $N$  positionne les points dans les autres dimensions



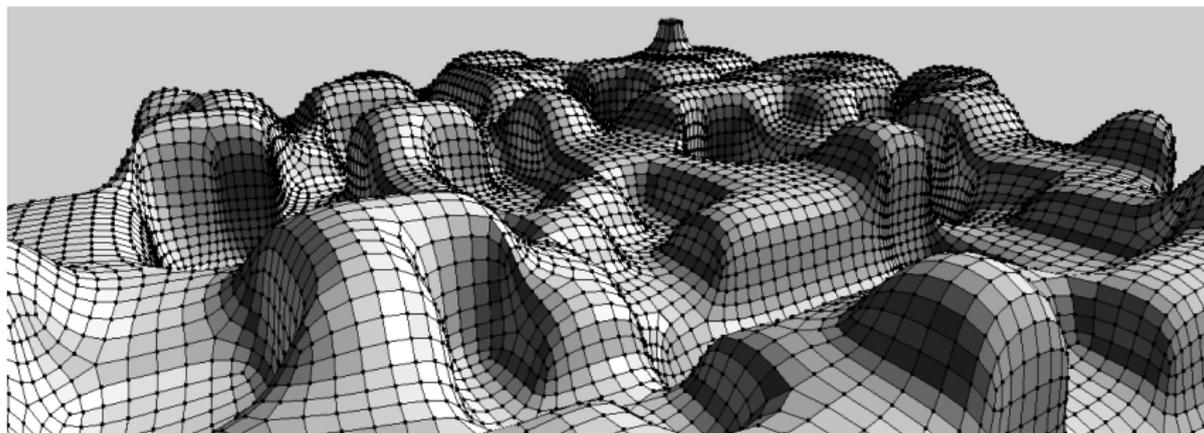
# Paysage de recherche (3/4)

## Optima locaux, plateaux et bassins d'attraction :

- Optimum local = point dont tous les voisins sont moins bons

$$s_i \in S \text{ tel que } \forall s_j \in N(s_i), f(s_j) < f(s_i)$$

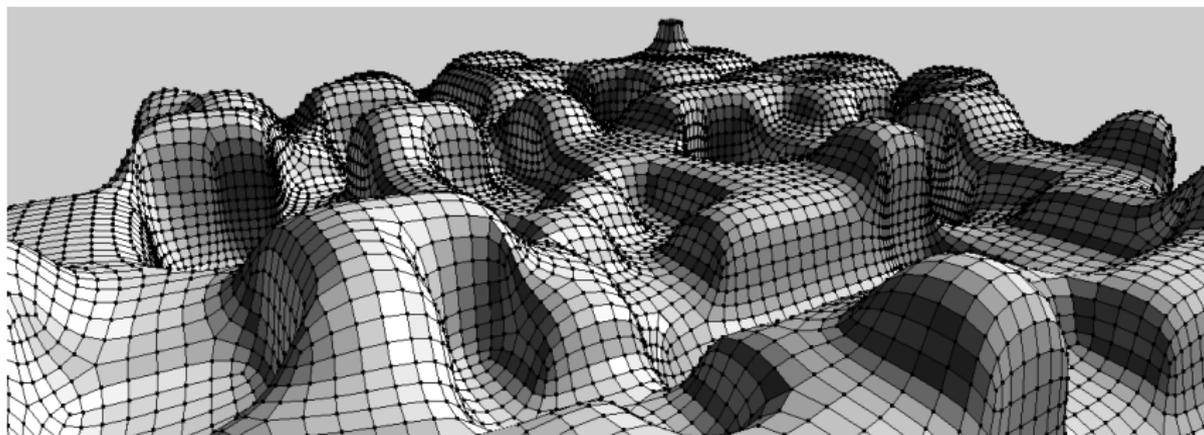
- Plateau = ensemble de points connexes dans le graphe  $G = (S, N)$  ayant tous la même valeur pour  $f$
- Bassin d'attraction d'un optimum local  $s_i$   
→ points que l'on peut atteindre depuis  $s_i$  sans jamais monter



# Paysage de recherche (4/4)

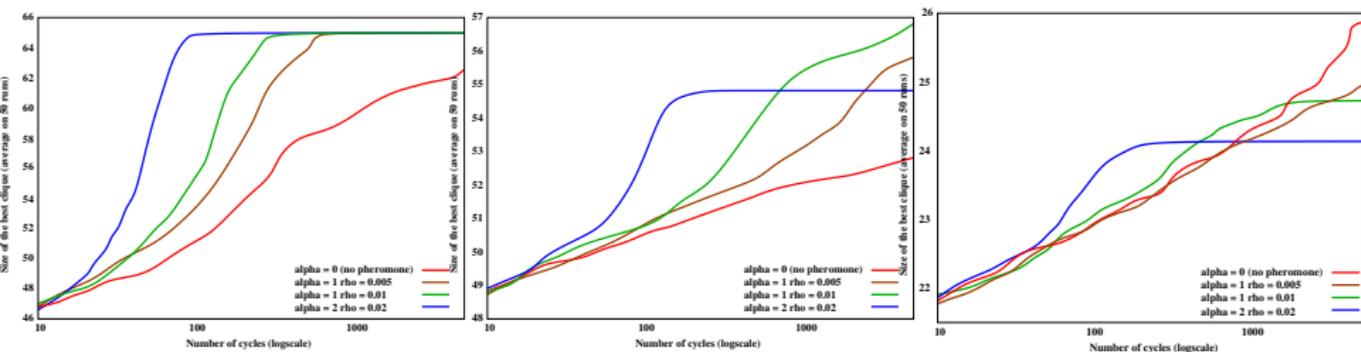
## Influence du paysage sur les performances

- Paysage avec 1 seul optimum local (= opt. global) et pas de plateau  
~> Un algorithme glouton sera très performant
- Paysage rugueux = nombreux optima + distrib. uniforme  
~> Pas de corrélation entre qualité et distance à l'opt. global  
~> Un algorithme purement aléatoire sera probablement performant
- Paysage de type "massif central"  
~> Performances dpdent du compromis intensification/diversification



# Illustration sur un problème de recherche de cliques

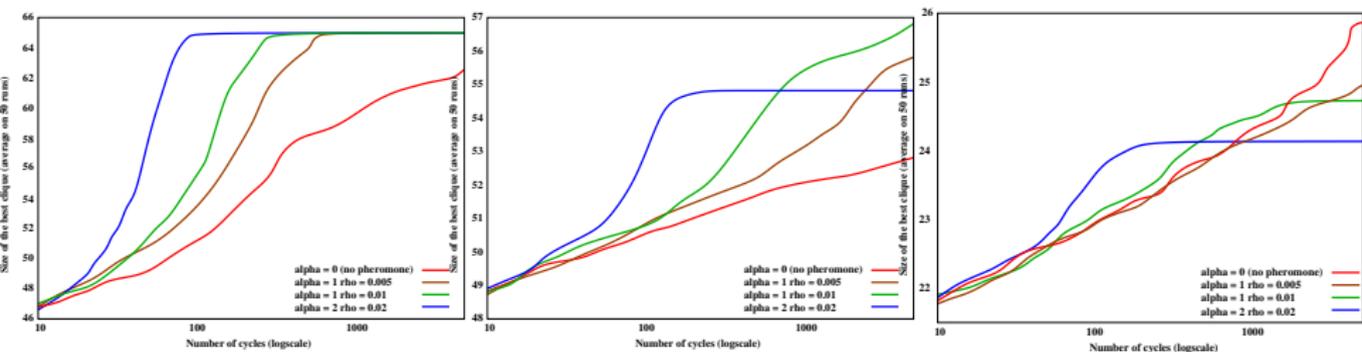
- Comparaison des performances de 4 algorithmes sur 3 instances



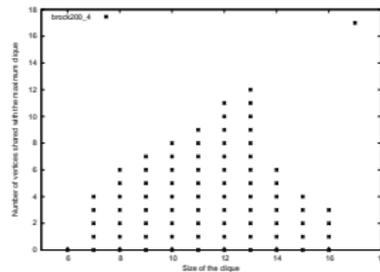
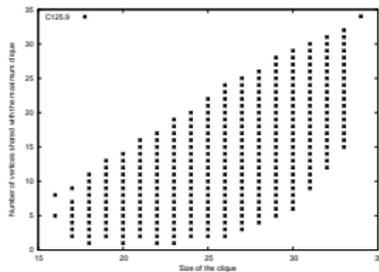
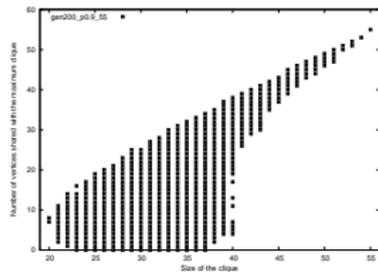
- Corrélation entre la taille d'une clique et sa distance à la clique max :

# Illustration sur un problème de recherche de cliques

- Comparaison des performances de 4 algorithmes sur 3 instances



- Corrélation entre la taille d'une clique et sa distance à la clique max :



# Plan du cours

## 1 Validation théorique d'algorithmes

## 2 Evaluation expérimentale d'algorithmes

- Processus expérimental
- Constitution d'un benchmark
- Choisir les facteurs, *Design Points* et indicateurs de performance
- Analyser les résultats
- Illustration : Comparaison expérimentale de solveurs de contraintes

## 3 Ingénierie algorithmique

# Choisir les facteurs et *Design Points*

## Facteurs : Choisir les paramètres qui influent sur les performances

~> Exploiter la littérature + connaissances sur l'algo + Expés exploratoires

## Niveaux : Identifier les valeurs pertinentes pour chaque facteur

- Facteurs symboliques : 1 niveau par valeur
- Facteurs numériques :
  - Identifier les intervalles de valeurs pertinentes
  - Echantillonner en utilisant une progression exponentielle  
~> 10, 20, 40, 80, ... ou 10, 100, 1000, ...

## *Design Points* :

- Full factorial design = toutes les combinaisons Facteur/Niveau possibles  
~> Permet d'identifier tous les effets des facteurs :
  - Effet principal : dépendant d'un seul facteur à la fois
  - Effets d'interaction : dépendant de plusieurs facteurs à la fois
- Si trop de facteurs et/ou de niveaux par facteur  
~> Choisir des combinaisons représentatives et complémentaires

# Indicateurs de performance

## Trois principales métriques de performance :

- Qualité de la solution
- Durée de l'exécution
- Consommation de mémoire

Les trois sont souvent dépendants :

↪ La qualité augmente quand on augmente la durée

↪ La durée diminue quand on augmente la consommation de mémoire

## Indicateurs de performance :

- Qualité de la solution :
  - % d'écart à la solution optimale (ou meilleure solution connue)
  - % de runs ayant trouvé la solution optimale (ou meilleure sol. )
  - ↪ Considérer plusieurs limites de temps / progression exponentielle
- Durée de l'exécution :
  - Nombre d'opérations dominantes
  - Temps CPU

# Indicateurs de performance pour mesurer la durée

## Nombre d'opérations dominantes :

- Identifier l'opération la plus fréquente (par ex., nb de comparaisons / tri)
- Ajouter un compteur à cette opération

Mesure indépendante du langage, du programmeur, de l'OS, ...  
mais pas toujours représentative de la durée

## Temps :

- Temps réel : Non fiable car trop dépendant de la charge de la machine
- Temps CPU : Pas vraiment fiable non plus !

Sur un HP avec 8 cœurs :

Compil.	Nb proc.	CPU	Réel
gcc	1	43.0	43.0
gcc -O3	1	27.9	28.2
gcc -O3	9	[36.0; 37.6]	[43.4; 43.6]

Sur un MAC avec 2 cœurs :

Compil.	Nb proc.	CPU	Réel
gcc	1	108	115
gcc -O3	1	67	79
gcc -O3	9	[97; 100]	[630; 649]

(Données issues de [Mcgeoch 2012])

... sans parler de l'impact de la mémoire sur les temps d'exécution !

# Plan du cours

## 1 Validation théorique d'algorithmes

## 2 Evaluation expérimentale d'algorithmes

- Processus expérimental
- Constitution d'un benchmark
- Choisir les facteurs, *Design Points* et indicateurs de performance
- Analyser les résultats
- Illustration : Comparaison expérimentale de solveurs de contraintes

## 3 Ingénierie algorithmique

# Analyse de données

## Objectif :

Transformer des données brutes en une information

## Outils de base pour l'analyse :

- Statistique descriptive :
  - ~> Description concise des propriétés essentielles
- Analyse de données graphique :
  - ~> Visualisation mettant en évidence les propriétés des données
- Tests statistiques :
  - ~> Procédure permettant de rejeter ou pas une hypothèse statistique

~> Cf cours d'introduction à la statistique fait en 4IF

# Statistique descriptive (rappels)

## Mesures de tendance centrale :

- Moyenne :  $\bar{X} = \frac{\sum x_i}{n}$   
~> Tient compte de toutes les valeurs (sensible aux *outliers*)
- Médiane : Valeur au milieu de la séquence triée de valeurs  
~> Ne tient pas compte de toutes les valeurs (insensible aux *outliers*)
- Mode : Valeur la plus fréquente  
~> Utilisé pour décrire des données non numériques

Dans une distribution normale, ces 3 mesures sont souvent proches

## Mesures de dispersion

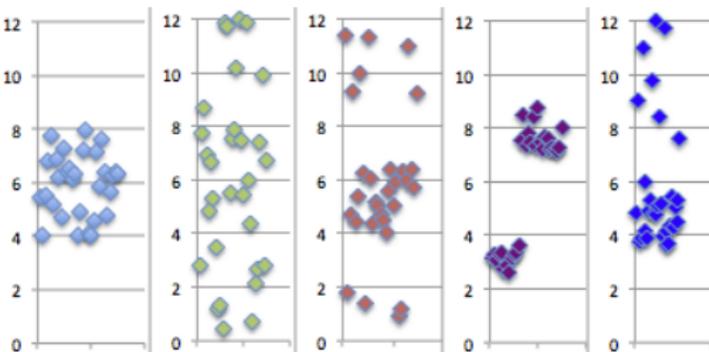
- Ecart type :  $\sigma = \sqrt{\frac{\sum (x_i - \bar{X})^2}{n}}$   
~> Ecart à la moyenne
- Ecart-inter-quartile :  $EIQ = Q3 - Q1$  avec
  - $Q1$  = valeur à la fin du premier quart de la séquence triée
  - $Q3$  = valeur à la fin du troisième quart de la séquence triée

## Regarder les données brutes avant de les analyser !

	Moyenne	Médiane	Ecart type	EIQ
Ciel	5,94	6,24	1,16	1,90
Vert	6,12	6,34	3,49	4,40
Rose	5,84	5,55	2,83	1,81
Violet	5,83	7,19	2,26	4,21
Bleu	5,88	4,91	2,52	2,21

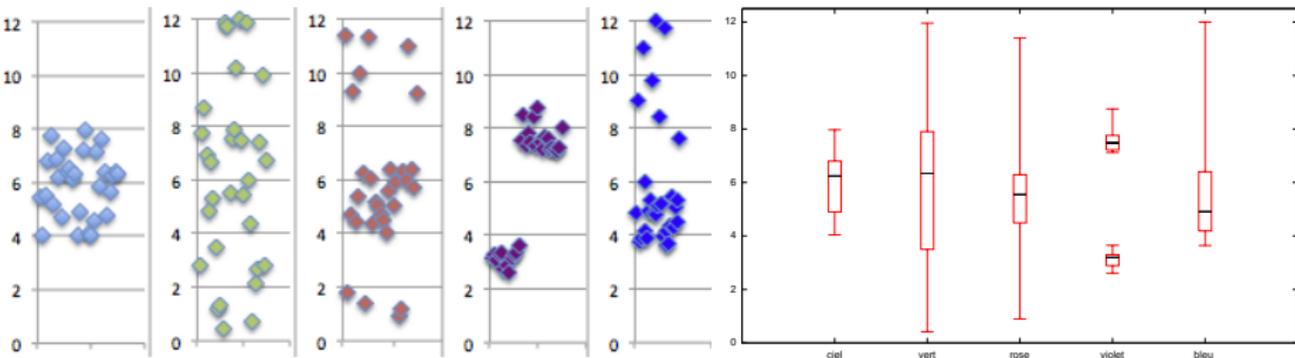
# Regarder les données brutes avant de les analyser !

	Moyenne	Médiane	Ecart type	EIQ
Ciel	5,94	6,24	1,16	1,90
Vert	6,12	6,34	3,49	4,40
Rose	5,84	5,55	2,83	1,81
Violet	5,83	7,19	2,26	4,21
Bleu	5,88	4,91	2,52	2,21



# Regarder les données brutes avant de les analyser !

	Moyenne	Médiane	Ecart type	EIQ
Ciel	5,94	6,24	1,16	1,90
Vert	6,12	6,34	3,49	4,40
Rose	5,84	5,55	2,83	1,81
Violet	5,83	7,19	2,26	4,21
Bleu	5,88	4,91	2,52	2,21



... et utiliser les boîtes à moustaches !

# Générer des données faciles à analyser !

*"If your experiment needs statistics then you ought to have done a better experiment."*

E. Rutherford

- Mémoriser toutes les données pertinentes
- Agrandir les plages de niveaux pour amplifier l'impact des facteurs  
~> Choisir les bonnes échelles
- Utiliser des techniques de réduction de variance (VRT) :
  - Même benchmark de référence pour toutes les expérimentations
  - Si benchmark hétérogène alors découper en classes homogènes  
~> Analyser les résultats pour chaque classe séparément
  - Normaliser les données avant de les analyser  
~> % d'écart à la solution optimale plutôt que valeur absolue
  - Faire plus de runs

# Comment comparer des algorithmes non déterministes ?

## Algorithme déterministe :

→ Plusieurs exécutions indépendantes retournent le même résultat

## Attention au déterminisme “forcé” :

- Soient  $i1$  et  $i2$  deux instances équivalentes à des permutations ou des renommages près
- Si l'exécution sur  $i1$  est différente de l'exécution sur  $i2$  alors le déterminisme a été forcé par des choix arbitraires

## Comparaison d'algorithmes non déterministes :

- Plusieurs runs pour chaque couple (instance/algo)
  - Déterminer le nb de runs en fct de la puissance du test. . . et du tps !
- Comparer les résultats pour chaque instance
  - $a1$  est-il significativement meilleur que  $a2$  sur  $i3$  ?
- Comparer les algorithmes
  - Corrélation entre caractéristiques des instances et perf. des algos ?

# Quelques outils et conseils

- Ne pas coder les paramètres en dur dans le programme :
  - ↪ Les lire dans un fichier (par ex. xml) ou les passer en ligne de cde
- Collecter toutes les données possibles
  - ↪ Formater les sorties pour faciliter l'analyse
- Lancement des expérimentations :

- Utiliser un langage de script (Shell)

```
for niveauInstance in `cat designPointInstance.txt`; do
  for niveauFacteur in `cat designPointFacteur.txt`; do
    for seed in `cat seeds.txt`; do
      echo -n "$seed $niveauInstance $niveauFacteur " » res.txt
      ./genInst -n $niveauInstance | ./prog -n $niveauFacteur -s $seed » res.txt
    done; done; done
```

- Mémoriser tous les scripts (avec les dates) dans un fichier texte
- Préférer plusieurs petits runs indépendants à un gros run
- Pour mesurer le temps CPU : time, gprof, getrusage()
- Analyse des données :
  - Filtres Shell (awk, sed, grep) pour parser, reformater, ... les sorties
  - R (<http://www.r-project.org/>) pour faire des statistiques
  - Gnuplot pour visualiser des courbes

# Plan du cours

## 1 Validation théorique d'algorithmes

## 2 Evaluation expérimentale d'algorithmes

- Processus expérimental
- Constitution d'un benchmark
- Choisir les facteurs, *Design Points* et indicateurs de performance
- Analyser les résultats
- Illustration : Comparaison expérimentale de solveurs de contraintes

## 3 Ingénierie algorithmique

# Contexte

## Problèmes de satisfaction de contraintes (CSP)

- Entrées :
  - Un ensemble  $X$  de  $n$  variables (inconnues)
  - Une fct  $D$  associant un domaine  $D(x_i)$  à chaque var.  $x_i \in X$
  - Un ensemble  $C$  de contraintes définies sur  $X$
- Question : Existe t'il une affectation affectant une valeur de  $D(x_i)$  à chaque variable  $x_i$  satisfaisant toutes les contraintes de  $C$  ?

↪ Généralisation de SAT à tous types de variables et de contraintes

## Solveurs de contraintes considérés

Exploration par construction d'un arbre/graphes de recherche :

- 6 stratégies de recherche différentes : CBT, DBT, CBJ, CBJR, BTD, DR
- 2 algorithmes de propagation différents : FC, MAC
- 2 heuristiques de choix de variables différentes : d, w

↪ 24 algorithmes différents

# Benchmark

On se restreint ici à des instances dont toutes les contraintes sont binaires

- 1092 instances réparties en 6 classes :

Class	#Instances	#Variables			#Values			#Constraints			Constraint tightness		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
ACAD	75	10	116	500	2	146	2187	45	691	4950	0.001	0.692	0.998
PATT	238	16	263	1916	3	66	378	48	4492	65390	0.002	0.795	0.996
QRND	80	50	220	315	7	11	20	451	2968	4388	0.122	0.578	0.823
RAND	206	23	37	59	8	36	180	84	282	753	0.095	0.613	0.984
REAL	193	200	628	1000	2	152	802	1235	6394	17447	0.0	0.519	1.0
STRUCT	300	150	257	500	20	23	25	617	1641	3592	0.544	0.647	0.753

- Certaines classes sont artificielles et regroupent des instances très différentes

## Question 1 : Mesurer le temps ou la taille de l'arbre ?

### Avantages de chacun :

- Le temps CPU tient compte de toutes les opérations effectuées
- La taille de l'arbre de recherche est très stable  
 ↪ Indppte de la machine, du compilo, du programmeur, du langage, ...

### Comparaison du nombre de noeuds explorés par seconde :

● (CBT,FC,d) : 78 947	● (DBT,MAC,d) : 2 584	● (CBJR,FC,d) : 21 873
● (CBT,FC,w) : 23 339	● (DBT,MAC,w) : 2 598	● (CBJR,FC,w) : 14 442
● (CBT,MAC,d) : 9 263	● (CBJ,FC,d) : 26 791	● (BTD,FC,d) : 2 590
● (CBT,MAC,w) : 6 898	● (CBJ,FC,w) : 19 182	● (BTD,FC,w) : 2 549
● (DBT,FC,d) : 23 164	● (CBJ,MAC,d) : 2 560	● (BTD,MAC,d) : 1 743
● (DBT,FC,w) : 9 512	● (CBJ,MAC,w) : 2 696	● (BTD,MAC,w) : 1 199

### D'où viennent ces différences ?

- Instances considérées ?
- Machine ? Codage ?
- Complexité des opérations à réaliser à chaque nœud ?

## Question 1 : Mesurer le temps ou la taille de l'arbre ?

### Avantages de chacun :

- Le temps CPU tient compte de toutes les opérations effectuées
- La taille de l'arbre de recherche est très stable  
 ~> Indppte de la machine, du compilateur, du programmeur, du langage, ...

### Comparaison du nombre de noeuds explorés par seconde :

● (CBT,FC,d) : 78 947	● (DBT,MAC,d) : 2 584	● (CBJR,FC,d) : 21 873
● (CBT,FC,w) : 23 339	● (DBT,MAC,w) : 2 598	● (CBJR,FC,w) : 14 442
● (CBT,MAC,d) : 9 263	● (CBJ,FC,d) : 26 791	● (BTD,FC,d) : 2 590
● (CBT,MAC,w) : 6 898	● (CBJ,FC,w) : 19 182	● (BTD,FC,w) : 2 549
● (DBT,FC,d) : 23 164	● (CBJ,MAC,d) : 2 560	● (BTD,MAC,d) : 1 743
● (DBT,FC,w) : 9 512	● (CBJ,MAC,w) : 2 696	● (BTD,MAC,w) : 1 199

### D'où viennent ces différences ?

- Instances considérées ? ~> **NON**
- Machine ? Codage ? ~> **NON**
- Complexité des opérations à réaliser à chaque nœud ? ~> **OUI**

~> Mesurer le temps CPU !

## Question 2 : Combien d'exécutions ?

### Les algorithmes comparés ne sont pas déterministes :

- Ordre aléatoire des valeurs dans les domaines
- L'heuristique de choix de variable laisse des ex-aequo

### Critères à évaluer pour déterminer le nombre d'exécutions :

- Nombre d'algorithmes et d'instances ?  
~>  $24 \times 1092$
- Temps d'une exécution ?
  - Algorithmes de complexité exponentielle  
~> Ajouter une limite de temps... Compromis à trouver !
  - ~> 30 minutes
- Puissance (power) du test statistique  
~> Probabilité de rejeter l'hypothèse nulle sachant qu'elle est fausse

~> 15 runs : c'est peu pour un test... mais on est limité par nos ressources !

### Question 3 : Quel algorithme résout le plus d'instances ?

#### Indicateur de performance :

↪ Pourcentage de runs ayant terminé dans la limite de temps fixée

#### Quelle limite de temps choisir ?

Pourcentage d'instances résolues pour 3 algos et 3 limites différentes :

Limite de temps :	1 s	10 s	1800 s
(CBJ,FC,d) :	<b>41.4</b>	55.2	88.0
(CBJ,MAC,w) :	39.7	57.6	<b>95.1</b>
(BTD,MAC,w) :	37.4	<b>61.9</b>	94.2

## Question 3 : Quel algorithme résout le plus d'instances ?

### Indicateur de performance :

↪ Pourcentage de runs ayant terminé dans la limite de temps fixée

### Quelle limite de temps choisir ?

Pourcentage d'instances résolues pour 3 algos et 3 limites différentes :

Limite de temps :	1 s	10 s	1800 s
(CBJ,FC,d) :	<b>41.4</b>	55.2	88.0
(CBJ,MAC,w) :	39.7	57.6	<b>95.1</b>
(BTD,MAC,w) :	37.4	<b>61.9</b>	94.2

- Etudier l'évolution du pourcentage de succès en fonction du temps
- Considérer une échelle logarithmique pour le temps
  - ↪ Permet de comparer pour des temps à différents ordres de grandeur

% solved instances			1	5	10	50	100	500	1000	1800
CBT	FC	d	37.0	45.2	47.6	52.7	55.3	60.0	61.1	61.7
		w	41.8	51.7	56.8	65.9	69.4	77.8	81.5	83.2
	MAC	d	43.0	51.7	56.7	65.5	69.1	75.3	76.5	77.7
		w	<b>47.1</b>	<b>61.5</b>	<b>68.3</b>	<b>80.5</b>	<b>85.2</b>	<b>92.3</b>	<b>94.3</b>	<b>95.4</b>
CBJ	FC	d	41.3	50.4	55.2	66.9	70.5	81.9	85.6	88.0
		w	39.6	51.0	55.0	67.8	72.6	84.0	88.1	91.0
	MAC	d	38.0	50.2	54.3	68.2	74.2	85.3	88.8	90.4
		w	39.7	53.1	57.6	73.7	79.6	90.7	93.5	95.1
CBJR	FC	d	39.9	49.4	53.3	63.3	66.9	75.8	78.1	79.5
		w	39.1	50.5	55.0	67.5	72.6	84.2	88.3	90.9
	MAC	d	29.2	37.3	41.1	46.4	48.5	53.9	55.4	56.5
		w	31.6	40.1	44.9	55.0	58.9	67.7	69.4	70.7
DBT	FC	d	33.8	38.0	38.8	40.8	41.5	43.9	45.8	46.7
		w	37.7	47.4	50.5	61.9	66.5	77.0	80.3	83.7
	MAC	d	35.8	46.2	49.4	56.6	60.0	66.6	68.0	69.3
		w	37.9	49.5	54.1	68.6	74.5	85.7	89.5	91.8
DR	FC	d	32.7	37.5	39.2	41.9	42.6	44.0	44.6	45.0
		w	35.1	44.4	48.1	55.4	59.8	71.9	76.3	79.4
	MAC	d	32.5	41.6	45.1	51.9	53.8	59.0	60.8	62.3
		w	34.4	44.3	48.7	57.8	62.5	75.2	80.3	84.5
BTD	FC	d	31.4	45.1	52.2	65.1	69.2	76.1	77.3	78.0
		w	33.5	48.0	55.5	70.2	74.9	82.1	83.9	84.5
	MAC	d	32.8	45.1	53.9	70.1	75.8	84.6	86.0	87.1
		w	37.4	51.3	61.9	77.6	83.3	91.9	93.6	94.2

## Question 4 : Quel algo est le meilleur pour le plus d'instances ?

CBT				CBJ				CBJR				DBT				DR				BTD			
FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC	
d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w
27.7	4.5	11.3	9.2	2.0	1.2	1.0	0.7	0.7	1.1	1.6	0.9	1.4	0.4	0.7	0.1	1.0	3.4	0.5	0.2	14.2	12.3	0.7	3.1

### Pour chaque algorithme A :

- 1 % d'instances pour lesquelles A est le meilleur (1092 instances)
- 2 % d'instances non triviales pour lesquelles A est le meilleur (622 instances non triviales)
- 3 % d'instances non triviales pour lesquelles A est bon (bon = meilleur ou pas significativement moins bon que le meilleur)
- 4 % d'instances non triviales pour lesquelles A est significativement meilleur que tous les autres

## Question 4 : Quel algo est le meilleur pour le plus d'instances ?

CBT				CBJ				CBJR				DBT				DR				BTD			
FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC	
d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w
27.7	4.5	11.3	9.2	2.0	1.2	1.0	0.7	0.7	1.1	1.6	0.9	1.4	0.4	0.7	0.1	1.0	3.4	0.5	0.2	14.2	12.3	0.7	3.1
17.8	2.3	8.8	11.1	1.1	0.3	0.3	1.1	0.8	1.6	0.0	0.8	0.8	0.2	0.5	0.0	0.5	2.7	0.5	0.2	23.3	18.8	1.3	5.1

### Pour chaque algorithme A :

- 1 % d'instances pour lesquelles A est le meilleur (1092 instances)
- 2 % d'instances **non triviales** pour lesquelles A est le meilleur (622 instances non triviales)
- 3 % d'instances **non triviales** pour lesquelles A est **bon** (bon = meilleur ou pas significativement moins bon que le meilleur)
- 4 % d'instances **non triviales** pour lesquelles A est **significativement meilleur que tous les autres**

## Question 4 : Quel algo est le meilleur pour le plus d'instances ?

CBT				CBJ				CBJR				DBT				DR				BTD			
FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC	
d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w
27.7	4.5	11.3	9.2	2.0	1.2	1.0	0.7	0.7	1.1	1.6	0.9	1.4	0.4	0.7	0.1	1.0	3.4	0.5	0.2	14.2	12.3	0.7	3.1
17.8	2.3	8.8	11.1	1.1	0.3	0.3	1.1	0.8	1.6	0.0	0.8	0.8	0.2	0.5	0.0	0.5	2.7	0.5	0.2	23.3	18.8	1.3	5.1
27.7	9.8	12.9	19.9	4.0	3.4	2.7	7.1	2.3	4.8	2.3	2.3	1.3	5.5	2.7	2.7	2.7	7.1	2.9	2.9	42.1	26.5	5.3	10.5

### Pour chaque algorithme A :

- 1 % d'instances pour lesquelles A est le meilleur (1092 instances)
- 2 % d'instances **non triviales** pour lesquelles A est le meilleur (622 instances non triviales)
- 3 % d'instances **non triviales** pour lesquelles A est **bon** (bon = meilleur ou pas significativement moins bon que le meilleur)
- 4 % d'instances **non triviales** pour lesquelles A est **significativement meilleur que tous les autres**

## Question 4 : Quel algo est le meilleur pour le plus d'instances ?

CBT				CBJ				CBJR				DBT				DR				BTD			
FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC	
d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w
27.7	4.5	11.3	9.2	2.0	1.2	1.0	0.7	0.7	1.1	1.6	0.9	1.4	0.4	0.7	0.1	1.0	3.4	0.5	0.2	14.2	12.3	0.7	3.1
17.8	2.3	8.8	11.1	1.1	0.3	0.3	1.1	0.8	1.6	0.0	0.8	0.8	0.2	0.5	0.0	0.5	2.7	0.5	0.2	23.3	18.8	1.3	5.1
27.7	9.8	12.9	19.9	4.0	3.4	2.7	7.1	2.3	4.8	2.3	2.3	1.3	5.5	2.7	2.7	2.7	7.1	2.9	2.9	42.1	26.5	5.3	10.5
6.4	0.3	5.5	5.8	0	0	0.2	0.5	0	1.1	0	0	0	0	0	0	0	1.9	0	0	14.0	7.2	0.2	1.3

### Pour chaque algorithme A :

- 1 % d'instances pour lesquelles A est le meilleur (1092 instances)
- 2 % d'instances **non triviales** pour lesquelles A est le meilleur (622 instances non triviales)
- 3 % d'instances **non triviales** pour lesquelles A est **bon** (bon = meilleur ou pas significativement moins bon que le meilleur)
- 4 % d'instances **non triviales** pour lesquelles A est **significativement meilleur que tous les autres**

# Conclusion ?

## Le “meilleur algorithme” dépend de la limite de temps considérée

- Faire varier cette limite
- Considérer une échelle logarithmique

## Le “meilleur algorithme” dépend du benchmark considéré :

- On peut (presque) toujours trouver des instances pour lesquelles on est meilleur : (CBT,FC,d) est 21ème en % d'instances résolues mais il est le meilleur pour plus de 27% des instances
- Certains algorithmes ne sont pas très bons en moyenne, mais sont les seuls à savoir bien résoudre certaines instances : (BTD,FC,d) est 15ème en % d'instances résolues mais il est significativement meilleur que tous les autres pour 14% des instances non triviales

- ↪ Parler aussi des instances pour lesquelles on est moins bon
- ↪ Utiliser la configuration automatique (cf fin du cours)

# Plan du cours

- 1 Validation théorique d'algorithmes
- 2 Evaluation expérimentale d'algorithmes
- 3 Ingénierie algorithmique

*In almost every computation, a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them for the purposes of a Calculating Engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.*

Ada Byron, 1843

### 3 niveaux possibles d'optimisation (tuning) :

- Algorithme  $\rightsquigarrow$  Paradigmes algorithmiques, Structures de données, ...
- Code  $\rightsquigarrow$  Boucles, Procédures, Gestion de la mémoire, ...
- Paramètres  $\rightsquigarrow$  Meilleures valeurs pour une instance / classe d'instances

### Objectifs :

- Réduire le nb d'instructions à exécuter ou le tps d'exec des instructions
- Ne change pas toujours les performances théoriques (asymptotiques),  
Mais peut permettre de gagner plusieurs facteurs en pratique !

# Plan du cours

1 **Validation théorique d'algorithmes**

2 **Evaluation expérimentale d'algorithmes**

3 **Ingénierie algorithmique**

- Optimisation d'algorithmes et de code
- Outils pour optimiser algorithmes et code
- Optimisation de paramètres
- Illustration : Sélection de solveurs de contraintes

# Optimisation d'algorithmes : Quelques règles générales

## Utiliser de la mémoire pour économiser des instructions

- Faire de la "Memoization"
- Maintenir incrémentalement plutôt que calculer "from scratch"

## Utiliser les bonnes structures de données

- Connaître les propriétés des différentes structures de données  
~ Tables de hachage, Arbres, Tas, Disjoint-sets, Sparse-sets, ...
- Etudier la fréquence des opérations pour choisir la bonne structure

## Sortir des boucles au plus tôt

Exemples : Dijkstra, Bellman-Ford, ...

## Elaguer les arbres de recherche :

- Calcul de bornes sur la fonction objectif ~ Branch & Bound
- Propagation de contraintes ~ Branch & Propagate
- Précalcul (rapide) de bonnes bornes

## Illustration sur le TSP : Faster and faster and faster yet [Bentley 97]

- Algo1 : Enumère  $n!$  permutations + calcule longueur
- Algo2 : Fixe le premier sommet
- Algo3 : Calcule le coût incrémentalement
- Algo4 : Mémorise les distances dans une matrice  $n \times n$
- Algo5 : Coupe branche si coût courant  $\geq$  meilleur coût connu

### Comparaison temps :

	ALGORITHM				
N	1	2	3	4	5
7	.07				
8	.63	.08	.04		
9	6.31	.70	.31	.06	
10	69.68	6.97	2.81	.57	.10
11		76.21	29.12	5.74	.40
12				63.08	1.91
13					13.71
14					74.86

### Tps Algo5 / type du graphe :

N	11	12	13	14
Uniform	.04	1.91	13.71	74.86
Circle	.34	.97	4.24	14.29
Matrix	.33	.49	1.10	3.80

## Illustration sur le TSP (suite)

- Algo6 : Coupe branche si coût ACM + coût courant  $\geq$  meilleur coût connu
- Algo7 : Utilise table de hachage pour mémoriser les ACM
- Algo8 : Plus proches voisins en premier

**Algorithm 8 On Three Distributions**

N	20	21	22	23	24
Matrix	.73	2.14	3.16	45.07	203.46
Uniform	.13	1.05	.60	1.09	24.14
Circle	.14	.12	.13	.12	.12

	ALGORITHM			
N	5	6	7	8
11	.40	.04		
12	1.91	.07		
13	13.71	.08	.02	.01
14	74.86	.43	.07	.04
15		1.15	.14	.02
16		1.94	.21	.04
17		7.19	.69	.14
18		10.82	.92	.36
19		27.27	2.14	.76
20		5.35	.42	.13
21		70.56	4.88	1.05
22		52.31	3.16	.60
23		49.52	2.61	1.09
24			50.38	24.14
25			166.64	86.20
26			36.20	23.80
27			92.85	60.42
28				193.00
29				137.61
30				137.92

# Code Tuning

## Différence d'échelle par rapport à l'optimisation d'algorithmes :

- Boucles et procédures plutôt que paradigmes algorithmiques
- Gestion de la mémoire plutôt que structures de données

## Quelques règles

- Sortir certaines instructions des boucles
- Utiliser des sentinelles pour économiser des tests
- Fusionner des boucles
- Déplier des boucles ou des procédures
- ...

↪ Gains souvent plus réduits... et perte de lisibilité/généralité potentielle !

↪ Beaucoup de ces optim. sont faites par le compilateur (option -O3 de gcc)

*We should forget about small efficiencies, say about 97% of the time :  
premature optimization is the root of all evil.*

D. Knuth

## Exemple 1 : Tri par insertion

```
pour i variant de 1 à  $n - 1$  faire  
  /* Invariant :  $tab[0..i - 1]$  est trié                               */  
   $j \leftarrow i$   
  tant que  $j > 0$  et  $tab[j] > tab[j - 1]$  faire  
    échanger( $tab[j]$ ,  $tab[j - 1]$ )  
     $j \leftarrow j - 1$ 
```

### Optimisations possibles ?

	n=40000	n=80000	n=160000
Code original	0.65	2.66	10.59

## Exemple 1 : Tri par insertion

```

pour i variant de 1 à  $n - 1$  faire
  /* Invariant :  $tab[0..i - 1]$  est trié                               */
   $j \leftarrow i$ 
  tant que  $j > 0$  et  $tab[j] > tab[j - 1]$  faire
    échanger( $tab[j], tab[j - 1]$ )
     $j \leftarrow j - 1$ 

```

### Optimisations possibles ?

- Opt1 : Sortir certaines instructions des boucles  
 ↪ Mémoriser  $tab[i]$  avant la boucle **tant que**

	n=40000		n=80000		n=160000	
Code original	0.65		2.66		10.59	
Opt1	0.40	-38%	1.61	-40%	6.43	-39%

## Exemple 1 : Tri par insertion

**pour**  $i$  variant de 1 à  $n - 1$  **faire**

  /\* Invariant :  $tab[0..i - 1]$  est trié \*/

$j \leftarrow i$

**tant que**  $j > 0$  et  $tab[j] > tab[j - 1]$  **faire**

    échanger( $tab[j]$ ,  $tab[j - 1]$ )

$j \leftarrow j - 1$

### Optimisations possibles ?

- Opt1 : Sortir certaines instructions des boucles  
 ~> Mémoriser  $tab[i]$  avant la boucle **tant que**
- Opt2 : Réduire le nombre de comparaisons  
 ~> Ajouter une sentinelle dans  $tab[0]$

	n=40000	n=80000	n=160000
Code original	0.65	2.66	10.59
Opt1	0.40    -38%	1.61    -40%	6.43    -39%
Opt2	0.67	2.72	10.91

## Exemple 1 : Tri par insertion

```

pour  $i$  variant de 1 à  $n - 1$  faire
  /* Invariant :  $tab[0..i - 1]$  est trié                               */
   $j \leftarrow i$ 
  tant que  $j > 0$  et  $tab[j] > tab[j - 1]$  faire
    échanger( $tab[j], tab[j - 1]$ )
     $j \leftarrow j - 1$ 

```

### Optimisations possibles ?

- Opt1 : Sortir certaines instructions des boucles  
 ~> Mémoriser  $tab[i]$  avant la boucle **tant que**
- Opt2 : Réduire le nombre de comparaisons  
 ~> Ajouter une sentinelle dans  $tab[0]$

	n=40000		n=80000		n=160000	
Code original	0.65		2.66		10.59	
Opt1	0.40	-38%	1.61	-40%	6.43	-39%
Opt2	0.67		2.72		10.91	
Opt1+Opt2	0.30	-54%	1.22	-54%	4.88	-54%

## Exemple 2 : Enumérer toutes les permutations d'un tableau

---

```
permut(int* tab, int k, int n)
```

début

```
  si  $k = n - 1$  alors afficher( $tab, n$ );
```

```
  sinon
```

```
    pour  $i$  de  $k$  à  $n - 1$  faire
```

```
      échanger( $tab[k], tab[i]$ )
```

```
      permut( $tab, k + 1, n$ )
```

```
      échanger( $tab[k], tab[i]$ )
```

---

### Optimisations possibles ?

n	Code original	Opt1	Opt2	Opt1+Opt2
11	0.89			
13	143.64			

## Exemple 2 : Enumérer toutes les permutations d'un tableau

---

```
permut(int* tab, int k, int n)
```

début

```
  si  $k = n - 1$  alors afficher( $tab, n$ );
```

```
  sinon
```

```
    pour  $i$  de  $k$  à  $n - 1$  faire
```

```
      échanger( $tab[k], tab[i]$ )
```

```
      permut( $tab, k + 1, n$ )
```

```
      échanger( $tab[k], tab[i]$ )
```

---

### Optimisations possibles ?

- Opt1 : Déplier *échanger* (inlining procedure call)

n	Code original	Opt1	Opt2	Opt1+Opt2
11	0.89	0.63		
13	143.64	97.60		

## Exemple 2 : Enumérer toutes les permutations d'un tableau

```
permut(int* tab, int k, int n)
```

début

```
    si  $k = n - 1$  alors afficher( $tab, n$ );
```

sinon

```
    pour  $i$  de  $k$  à  $n - 1$  faire
```

```
        échanger( $tab[k], tab[i]$ )
```

```
        permut( $tab, k + 1, n$ )
```

```
        échanger( $tab[k], tab[i]$ )
```

```
si  $k = n - 1$  alors afficher( $tab, n$ );
```

sinon

```
    pour  $i$  de  $k$  à  $n - 1$  faire
```

```
        échanger( $tab[k], tab[i]$ )
```

```
        pour  $j$  de  $k + 1$  à  $n - 1$  faire
```

```
            échanger( $tab[k + 1], tab[i]$ )
```

```
            permut2( $tab, k + 2, n$ )
```

```
            échanger( $tab[k + 1], tab[i]$ )
```

```
        échanger( $tab[k], tab[i]$ )
```

### Optimisations possibles ?

- Opt1 : Déplier *échanger* (inlining procedure call)
- Opt2 : Diviser par 2 le nombre d'appels récursifs (Hypothèse :  $n$  impair)

n	Code original	Opt1	Opt2	Opt1+Opt2
11	0.89	0.63	0.82	
13	143.64	97.60	126.21	

## Exemple 2 : Enumérer toutes les permutations d'un tableau

```
permut(int* tab, int k, int n)
```

début

```
    si  $k = n - 1$  alors afficher( $tab, n$ );
```

sinon

```
    pour  $i$  de  $k$  à  $n - 1$  faire
```

```
        échanger( $tab[k], tab[i]$ )
```

```
        permut( $tab, k + 1, n$ )
```

```
        échanger( $tab[k], tab[i]$ )
```

```
si  $k = n - 1$  alors afficher( $tab, n$ );
```

sinon

```
    pour  $i$  de  $k$  à  $n - 1$  faire
```

```
        échanger( $tab[k], tab[i]$ )
```

```
        pour  $j$  de  $k + 1$  à  $n - 1$  faire
```

```
            échanger( $tab[k + 1], tab[i]$ )
```

```
            permut2( $tab, k + 2, n$ )
```

```
            échanger( $tab[k + 1], tab[i]$ )
```

```
        échanger( $tab[k], tab[i]$ )
```

### Optimisations possibles ?

- Opt1 : Déplier *échanger* (inlining procedure call)
- Opt2 : Diviser par 2 le nombre d'appels récursifs (Hypothèse :  $n$  impair)

n	Code original	Opt1	Opt2	Opt1+Opt2
11	0.89	0.63	0.82	0.56
13	143.64	97.60	126.21	85.20

## Exemple 2 : Enumérer toutes les permutations d'un tableau

```
permut(int* tab, int k, int n)
```

début

```
    si k = n - 1 alors afficher(tab, n);
```

sinon

```
        pour i de k à n - 1 faire
```

```
            échanger(tab[k], tab[i])
```

```
            permut(tab, k + 1, n)
```

```
            échanger(tab[k], tab[i])
```

```
si k = n - 1 alors afficher(tab, n);
```

sinon

```
    pour i de k à n - 1 faire
```

```
        échanger(tab[k], tab[i])
```

```
        pour j de k + 1 à n - 1 faire
```

```
            échanger(tab[k + 1], tab[i])
```

```
            permut2(tab, k + 2, n)
```

```
            échanger(tab[k + 1], tab[i])
```

```
        échanger(tab[k], tab[i])
```

### Optimisations possibles ?

- Opt1 : Déplier *échanger* (inlining procedure call)
- Opt2 : Diviser par 2 le nombre d'appels récursifs (Hypothèse :  $n$  impair)

n	Code original		Opt1		Opt2		Opt1+Opt2	
11	0.89	0.35	0.63	0.34	0.82	0.28	0.56	0.28
13	143.64	52.92	97.60	52.87	126.21	44.28	85.20	44.27

Résultats avec l'option -O3 de gcc

↪ Parfois, il vaut mieux laisser le compilateur faire les optimisations !

# Plan du cours

1 **Validation théorique d'algorithmes**

2 **Evaluation expérimentale d'algorithmes**

3 **Ingénierie algorithmique**

- Optimisation d'algorithmes et de code
- Outils pour optimiser algorithmes et code
- Optimisation de paramètres
- Illustration : Sélection de solveurs de contraintes

# Outils pour l'optimisation d'algorithmes et de code

## Outils de profilage

- gprof (Unix)
- Cachegrind (Unix)
- Instruments (MacOS X)
- ...

Temps passé par procédure (pourcentage et valeur absolue)

~> Pas toujours compatible avec les optimisations du compilateur !

## Outils pour l'évaluation expérimentale et l'analyse de données

Vérifiez expérimentalement que vos optimisations optimisent effectivement votre programme !

# Illustration : AntClique

## Problème MaxClique (rappel)

- Entrée : un graphe  $G = (V, E)$
- Sortie : la plus grande clique de  $G$   
(clique = sous-ensemble  $C \subseteq V$  tel que  $\forall i, j \in C, i \neq j \Rightarrow (i, j) \in E$ )

## AntClique :

- Algorithme incomplet : calcule une borne inférieure
- Basé sur la méta-heuristique Ant Colony Optimization (ACO)

# Basic Idea

- initialize pheromone trails
- repeat
  - ① each ant builds a clique
  - ② update pheromone trails
- until optimal clique found or stagnation

# Basic Idea

- **initialize pheromone trails**
- repeat
  - 1 each ant builds a clique
  - 2 update pheromone trails
- until optimal clique found or stagnation

## Pheromone trails :

Pheromone is laid on pairs of vertices

$\leadsto \tau(i, j) =$  desirability of selecting both  $i$  and  $j$  in a same clique

# Basic Idea

- initialize pheromone trails
- repeat
  - 1 **each ant builds a clique**
  - 2 update pheromone trails
- until optimal clique found or stagnation

## Greedy randomized construction of a clique $C$

- Start from an empty clique and iteratively add vertices to it
- Let  $C =$  partial clique and  $cand = \{j \in V \setminus C, \forall i \in C, (i, j) \in E\}$
- Vertex  $v_j$  added to  $C$  randomly chosen in  $cand$  wrt probability

$$p(v_j) = \frac{[\sum_{i \in C} \tau(i, j)]^\alpha}{\sum_{k \in cand} [\sum_{i \in C} \tau(i, k)]^\alpha}$$

where  $\alpha =$  pheromone weight (parameter)

# Basic Idea

- initialize pheromone trails
- repeat
  - 1 each ant builds a clique
  - 2 **update pheromone trails**
- until optimal clique found or stagnation

## Pheromone updating step

- Evaporation : multiply pheromone trails by  $(1 - \rho)$   
 $\rightsquigarrow \rho = \text{evaporation rate } (0 \leq \rho \leq 1)$
- Reward : add pheromone on all pairs of the best clique

- Profiling de `main`

60650.0ms	99.9%	1,0	▼main essai
58457.0ms	96.3%	22,0	▶buildClique essai
2122.0ms	3.4%	2122,0	updatePheromoneTrails essai
69.0ms	0.1%	6,0	▶createGraph essai
1.0ms	0.0%	1,0	initPhero essai

- 96.3% du temps pour construire les cliques

~> Zoom sur `buildClique`

```

clique[0] = getNextRand(G->nbVertices);
cliqueSize = 1;
nbCandidates = selectCandidates(cliqueSize, clique, candidates, G);
while (nbCandidates>0){
    computeProba(nbCandidates, candidates, cliqueSize, clique, alpha, G, p);
    clique[cliqueSize++] = candidates[chooseNextVertex(p, nbCandidates)];
    nbCandidates = selectCandidates(cliqueSize, clique, candidates, G);
}

```

- Profiling de `buildClique`

58780.0ms	96.3%	12,0	▼buildClique essai
45996.0ms	75.4%	17453,0	▶selectCandidates essai
12311.0ms	20.1%	5984,0	▶computeProba essai
218.0ms	0.3%	198,0	▶chooseNextVertex essai

- 75.4% du temps pour sélectionner les candidats

~> Opt1 : Maintenir la liste des candidats de façon incrémentale

- Code de `buildClique2` :

```

clique[0] = getNextRand(G->nbVertices);
cliqueSize = 1;
nbCandidates = G->degree[clique[0]];
for (i=0; i<nbCandidates; i++) candidates[i] = G->succ[clique[0]][i];
while (nbCandidates>0){
    computeProba(nbCandidates, candidates, cliqueSize, clique, alpha, G, p);
    v = candidates[chooseNextVertex(p, nbCandidates)];
    clique[cliqueSize++] = v;
    for (i=0; i<nbCandidates; i++){
        if (!isEdge(v, candidates[i], G)){
            candidates[i] = candidates[--nbCandidates];
            i--;
        }
    }
}

```

- Profiling de `buildClique2`

13407.0ms	85.7%	463,0		▼buildClique2	essai
12080.0ms	77.2%	5893,0		▶computeProba	essai
604.0ms	3.8%	604,0		isEdge	essai
221.0ms	1.4%	211,0		▶chooseNextVertex	essai

- 77.2% du temps pour calculer les probabilités

↪ Opt2 : Maintenir le facteur phéromonal de façon incrémentale

- Code de `buildClique3` :

```

clique[0] = getNextRand(G->nbVertices);
cliqueSize = 1;
nbCandidates = G->degree[clique[0]];
for (i=0; i<nbCandidates; i++){
    candidates[i] = G->succ[clique[0]][i];
    tauClique[candidates[i]] = (G->tauE)[clique[0]][candidates[i]];
}
while (nbCandidates>0){
    computeProba3(tauClique, nbCandidates, candidates, cliqueSize, clique, alpha, G, p);
    v = candidates[chooseNextVertex(p, nbCandidates)];
    clique[cliqueSize++] = v;
    for (i=0; i<nbCandidates; i++){
        if (!isEdge(v, candidates[i], G)){
            candidates[i] = candidates[--nbCandidates];
            i--;
        }
        else tauClique[candidates[i]] += (G->tauE)[v][candidates[i]];
    }
}
}

```

- Profiling de `buildClique3`

8614.0ms	79.3%	642,0		▼buildClique3	essai
7080.0ms	65.1%	810,0		▼computeProba3	essai
6270.0ms	57.7%	6270,0		0x7fff88363e00	libsystem_m.dylib
617.0ms	5.6%	617,0		isEdge	essai
237.0ms	2.1%	215,0		►chooseNextVertex	essai

- 57.7% du temps pour `pow` (de la bibliothèque `math.h`)  
 ~> Opt3 : remplacer `pow` par `myPow` !

- Profiling de `buildClique4`

2482.0ms	52.6%	684,0		▼ <code>buildClique4</code> essai
954.0ms	20.2%	587,0		▼ <code>computeProba4</code> essai
367.0ms	7.7%	367,0		<code>myPow</code> essai
644.0ms	13.6%	644,0		<code>isEdge</code> essai
197.0ms	4.1%	176,0		▶ <code>chooseNextVertex</code> essai

- 20.2% du temps pour `computeProba4`

↪ Opt4 : fusionner la boucle calculant les probas avec le filtrage

- Profiling de `buildClique5`

2314.0ms	50.8%	1119,0		▼ <code>buildClique5</code> essai
603.0ms	13.2%	603,0		<code>myPow</code> essai
385.0ms	8.4%	385,0		<code>isEdge</code> essai
205.0ms	4.5%	170,0		▶ <code>chooseNextVertex</code> essai

- Une dernière optimisation ?

↪ Opt5 : remplacer la recherche séquentielle de `chooseNextVertex` par une recherche dichotomique

- Profiling de `buildClique6`

2228.0ms	49.7%	1114,0		▼ <code>buildClique6</code> essai
617.0ms	13.7%	617,0		<code>myPow</code> essai
391.0ms	8.7%	391,0		<code>isEdge</code> essai
105.0ms	2.3%	84,0		▶ <code>chooseDicho</code> essai ↻

Les gains sont de plus en plus réduits... Est-ce bien utile ?

# Comparaison expérimentale (avec l'option -O3 !)

- Code initial
- Opt1 : Maintien incrémental des candidats
- Opt2 : Maintien incrémental du facteur phéromonal
- Opt3 Remplacement de `pow` de `math.h` par une fonction ad-hoc
- Opt4 : Fusion des boucles
- Opt5 : Recherche dichotomique du sommet sélectionné

	Code init
C125.9	3.61
C250.9	8.34
C500.9	23.53
C1000.9	111.30
C2000.9	347.15

# Comparaison expérimentale (avec l'option -O3 !)

- Code initial
- Opt1 : Maintien incrémental des candidats
- Opt2 : Maintien incrémental du facteur phéromonal
- Opt3 Remplacement de `pow` de `math.h` par une fonction ad-hoc
- Opt4 : Fusion des boucles
- Opt5 : Recherche dichotomique du sommet sélectionné

	Code init	Opt1
C125.9	3.61	1.79
C250.9	8.34	3.65
C500.9	23.53	8.49
C1000.9	111.30	28.94
C2000.9	347.15	88.18

# Comparaison expérimentale (avec l'option -O3 !)

- Code initial
- Opt1 : Maintien incrémental des candidats
- Opt2 : Maintien incrémental du facteur phéromonal
- Opt3 Remplacement de `pow` de `math.h` par une fonction ad-hoc
- Opt4 : Fusion des boucles
- Opt5 : Recherche dichotomique du sommet sélectionné

	Code init	Opt1	Opt2
C125.9	3.61	1.79	1.29
C250.9	8.34	3.65	2.78
C500.9	23.53	8.49	5.84
C1000.9	111.30	28.94	13.52
C2000.9	347.15	88.18	27.32

# Comparaison expérimentale (avec l'option -O3 !)

- Code initial
- Opt1 : Maintien incrémental des candidats
- Opt2 : Maintien incrémental du facteur phéromonal
- Opt3 Remplacement de `pow` de `math.h` par une fonction ad-hoc
- Opt4 : Fusion des boucles
- Opt5 : Recherche dichotomique du sommet sélectionné

	Code init	Opt1	Opt2	Opt3
C125.9	3.61	1.79	1.29	0.32
C250.9	8.34	3.65	2.78	0.71
C500.9	23.53	8.49	5.84	1.59
C1000.9	111.30	28.94	13.52	4.06
C2000.9	347.15	88.18	27.32	10.64

# Comparaison expérimentale (avec l'option -O3 !)

- Code initial
- Opt1 : Maintien incrémental des candidats
- Opt2 : Maintien incrémental du facteur phéromonal
- Opt3 Remplacement de `pow` de `math.h` par une fonction ad-hoc
- Opt4 : Fusion des boucles
- Opt5 : Recherche dichotomique du sommet sélectionné

	Code init	Opt1	Opt2	Opt3	Opt4
C125.9	3.61	1.79	1.29	0.32	0.24
C250.9	8.34	3.65	2.78	0.71	0.51
C500.9	23.53	8.49	5.84	1.59	1.21
C1000.9	111.30	28.94	13.52	4.06	3.32
C2000.9	347.15	88.18	27.32	10.64	9.14

# Comparaison expérimentale (avec l'option -O3 !)

- Code initial
- Opt1 : Maintien incrémental des candidats
- Opt2 : Maintien incrémental du facteur phéromonal
- Opt3 Remplacement de `pow` de `math.h` par une fonction ad-hoc
- Opt4 : Fusion des boucles
- Opt5 : Recherche dichotomique du sommet sélectionné

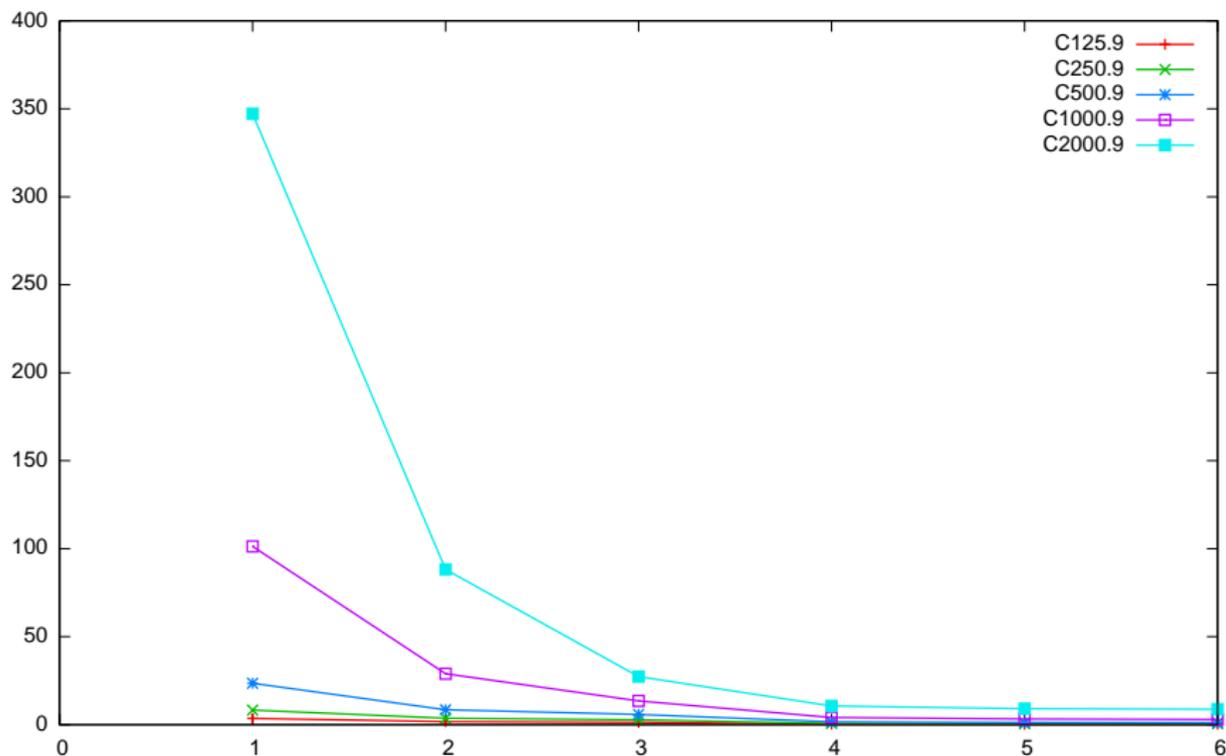
	Code init	Opt1	Opt2	Opt3	Opt4	Opt5
C125.9	3.61	1.79	1.29	0.32	0.24	0.24
C250.9	8.34	3.65	2.78	0.71	0.51	0.51
C500.9	23.53	8.49	5.84	1.59	1.21	1.11
C1000.9	111.30	28.94	13.52	4.06	3.32	2.98
C2000.9	347.15	88.18	27.32	10.64	9.14	8.81

# Comparaison expérimentale (avec l'option -O3 !)

- Code initial
- Opt1 : Maintien incrémental des candidats
- Opt2 : Maintien incrémental du facteur phéromonal
- Opt3 Remplacement de `pow` de `math.h` par une fonction ad-hoc
- Opt4 : Fusion des boucles
- Opt5 : Recherche dichotomique du sommet sélectionné

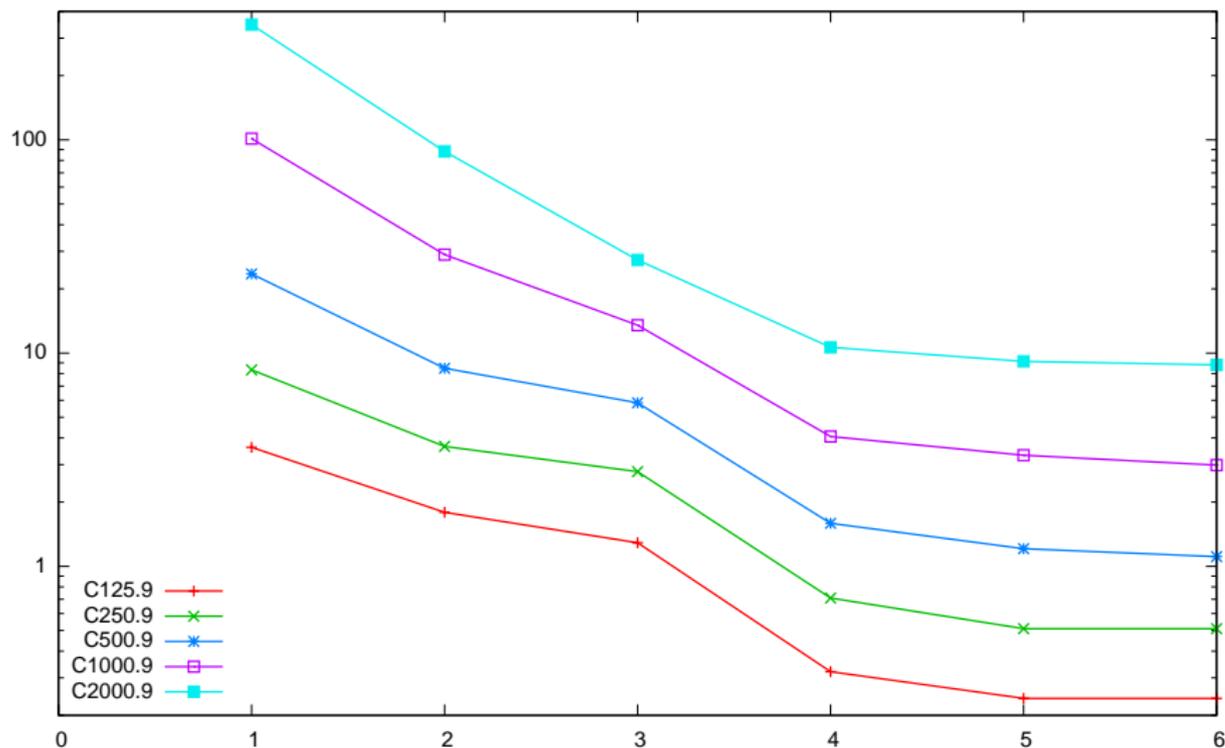
	Code init	Opt1	Opt2	Opt3	Opt4	Opt5
C125.9	3.61	1.79	1.29	0.32	0.24	0.24
C250.9	8.34	3.65	2.78	0.71	0.51	0.51
C500.9	23.53	8.49	5.84	1.59	1.21	1.11
C1000.9	111.30	28.94	13.52	4.06	3.32	2.98
C2000.9	347.15	88.18	27.32	10.64	9.14	8.81
C2000.5	33.64	11.37	6.23	4.32	4.15	4.03
C4000.5	85.53	30.50	18.73	14.75	14.17	14.02

# Comparaison des 6 variantes sur une courbe



Pas très lisible  $\rightsquigarrow$  Utiliser une échelle logarithmique !

# Comparaison avec une échelle logarithmique



# Plan du cours

1 **Validation théorique d'algorithmes**

2 **Evaluation expérimentale d'algorithmes**

3 **Ingénierie algorithmique**

- Optimisation d'algorithmes et de code
- Outils pour optimiser algorithmes et code
- Optimisation de paramètres
- Illustration : Sélection de solveurs de contraintes

# Optimisation de paramètres

## Différents types de paramètres :

- Valeurs numériques fixant des seuils, poids, fréquences, ...  
~> Changent les performances de l'algorithme
- Choix de conception ou de programmation (Hyper-paramètres)  
~> Changent l'algorithme

## Programming by Optimization [Hoos 2012]

*Developers specify a potentially large design space of programs that accomplish a given task, from which versions of the program optimized for various use contexts are generated automatically.*

# Configuration vs Sélection automatique d'algorithmes

## Données en entrée des problèmes d'optimisation de paramètres :

- Un algo  $A$  paramétré par un ensemble de (hyper-)paramètres  $P$
- Pour chaque paramètre  $p_i \in P$ , l'ens. des valeurs possibles de  $p_i$
- Un ensemble  $E$  d'instances d'apprentissage (training)
- Un indicateur de performance  $I$

## Problème de configuration :

Chercher les valeurs de  $P$  optimisant  $A$  pour l'indic.  $I$  sur les instances de  $E$

↪ Meilleur paramétrage pour toute une classe d'instances

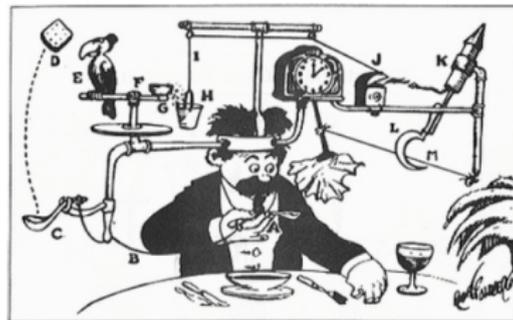
## Problème de sélection :

Construire un modèle pour choisir les valeurs de  $P$  / nouvelle instance

↪ Meilleur paramétrage pour chaque nouvelle instance

↪ Alternative : Meilleur planning (séquentiel ou parallèle)

**Oftentimes, a solver's behavior can be adjusted by carefully tweaking its parameters**



**But even as experts, our intuitions about “correct” values may be wrong and we want to avoid manually trying every possible combination**

**Solution: Algorithm Configuration**

image extraite de [Kotthoff et al - IJCAI 2013]

# A New Algorithm is Born

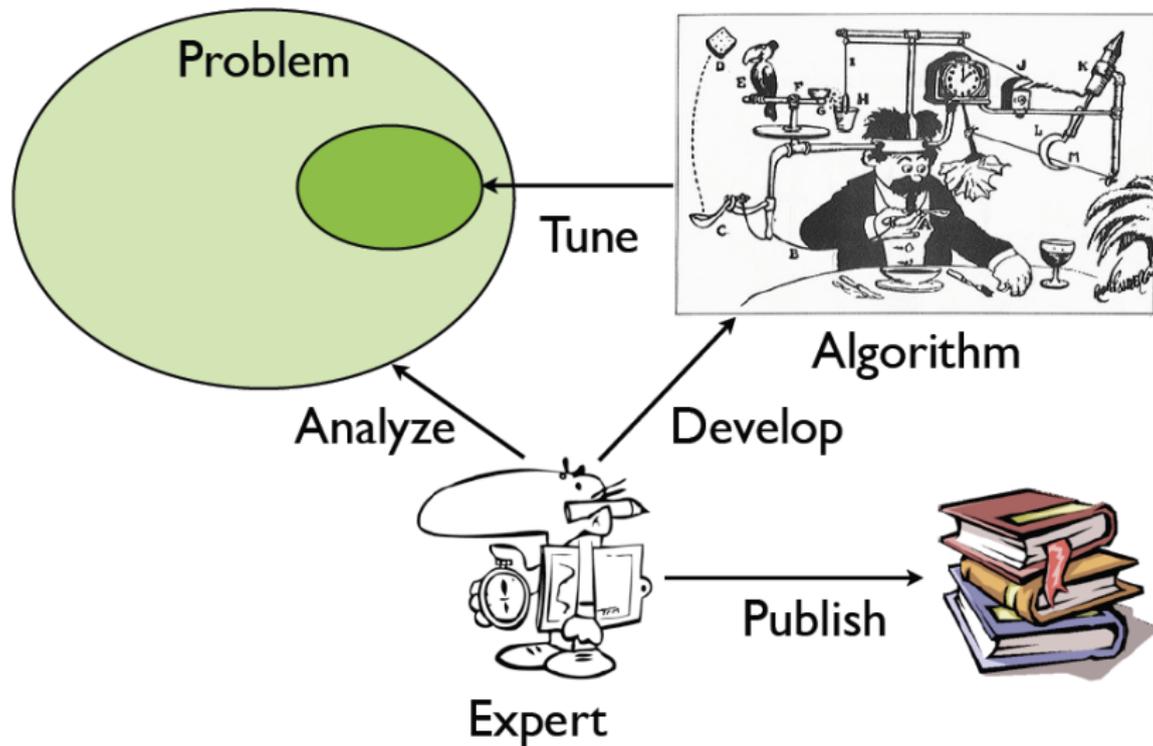


image extraite de [Kotthoff et al - IJCAI 2013]

# Using the New Algorithm

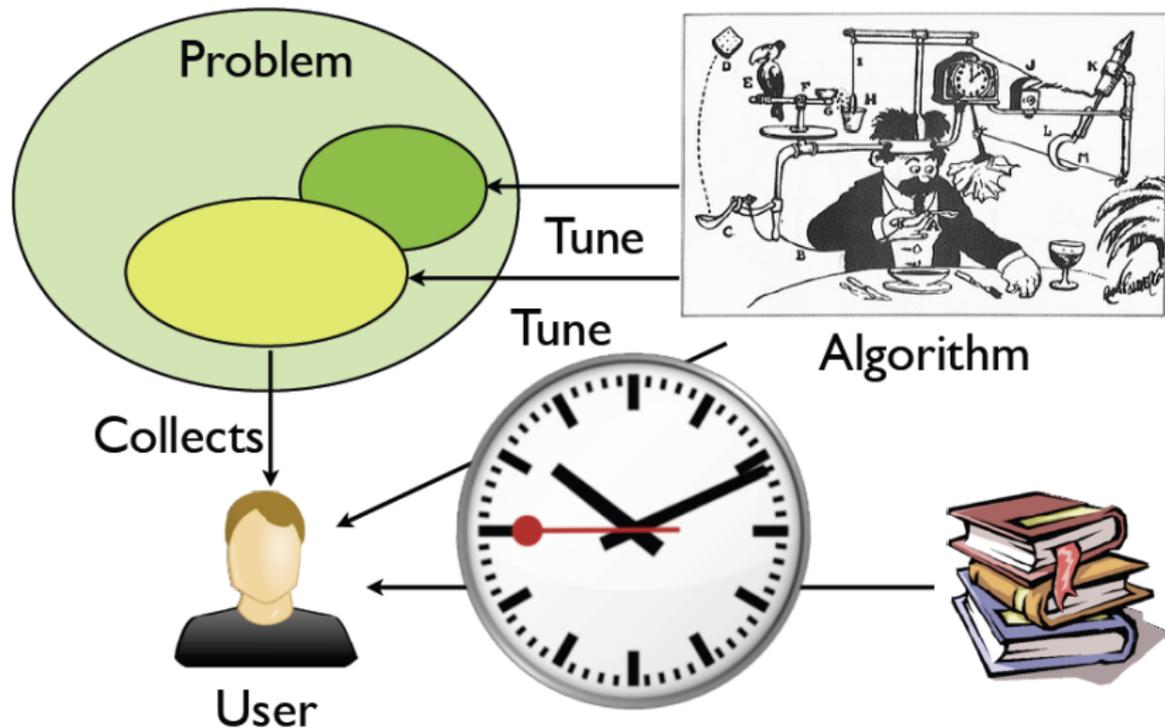


image extraite de [Kotthoff et al - IJCAI 2013]

# Using the New Algorithm

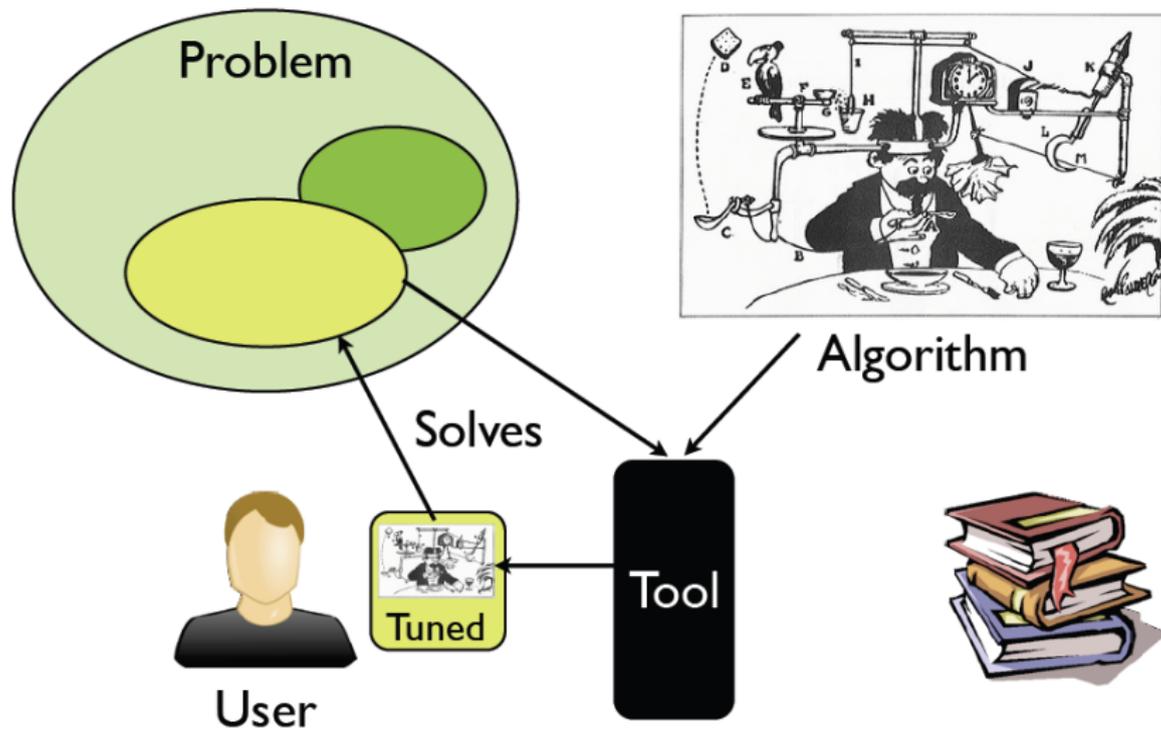


image extraite de [Kotthoff et al - IJCAI 2013]

# Instances d'apprentissage

## Hypothèse forte :

- Les instances à résoudre ont des caractéristiques similaires  
     $\leadsto$  Un bon paramétrage pour un ss-ens. doit être bon pour l'ensemble
- Si ce n'est pas le cas :
  - Diviser l'ensemble d'instances en sous-ensembles homogènes  
     $\leadsto$  Clustering
  - Appliquer la configuration automatique sur chaque sous-ensemble  
     $\leadsto$  cf Sélection d'algorithmes

## Nombre d'instances d'apprentissage utilisées pour configurer l'algo :

Trouver le bon compromis :

- Pas assez d'instances  $\leadsto$  Over-fitting
- Trop d'instances  $\leadsto$  Processus de configuration trop long

# Espace de recherche

## Espace de recherche :

Ensemble des configurations possibles des paramètres

~> Généralement impossible à explorer exhaustivement

## Choix des configurations :

Préférer un échantillonnage aléatoire plutôt que régulier

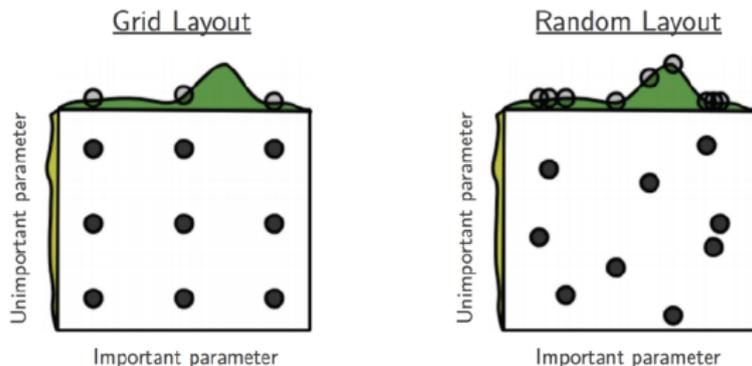


image extraite de [Kotthoff et al - IJCAI 2013]

# Configuration automatique d'algo : Principales approches

**Iterated Local Search:** Check the neighboring parameterizations of the current best setting, performing gradient descent. Restart randomly or when a local optima is found.

- BasicILS, ParamILS, FocusedILS

**ParamILS: An Automatic Algorithm Configuration Framework**  
Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Thomas Stutzle. JAIR (2009)

**Population Based:** Use genetic algorithms to pick the most fit parameter settings in a population, allowing for crossovers and mutations to create new entries to the population.

- GGA

**A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms**  
Carlos Ansótegui, Meinolf Sellmann, Kevin Tierney. CP (2009)

**SMBO:** Use regression to predict the performance of parameters on a set of instances. The next parameters to try are those where the regression predicts the most improvement.

- SMAC

**Sequential Model-Based Optimization for General Algorithm Configuration**  
Frank Hutter, Holger H. Hoos and Kevin Leyton-Brown. LION (2011)

**Racing:** Select a sample of parameterizations, race the instances on each instance in parallel, removing parameterizations that are statistically inferior to the others.

- F-Race, Iterated F-Race

**Automated Algorithm Tuning using F-races: Recent Developments**  
Mauro Birattar, Zhi Yuan, Prasanna Balaprakash, Thomas Stutzle. MIC (2009)

# Problème de sélection

**There are times when there are many approaches available to solve a particular problem**



**But we know there is no silver bullet solver that's perfect on everything, so how do we choose the best approach for a particular instance?**

**Solution: Algorithm Selection**

image extraite de [Kotthoff et al - IJCAI 2013]

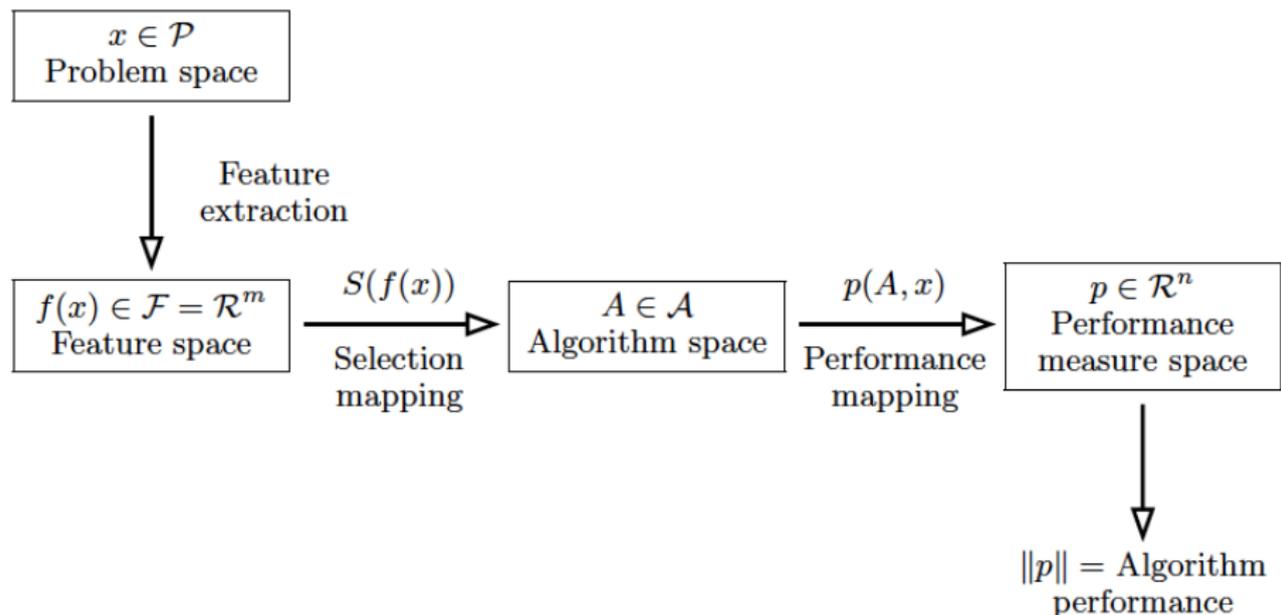


image extraite de [Kotthoff et al - IJCAI 2013]

# Instances d'apprentissage

## Constitution d'un jeu d'apprentissage :

- Choisir des instances variées et représentatives

## Caractérisation des instances par des features :

- Features syntaxiques / statiques : collectées par analyse de l'instance
  - Nb de variables, de valeurs, de contraintes (min/max/moy)
  - Analyse du graphe des contraintes
  - ...
- Features dynamiques : collectées lors de sondages (probing)
  - Nombre de noeuds explorés
  - Profondeur de l'arbre
  - Nombre de valeurs filtrées à chaque noeud
  - ...

# Apprentissage de modèles

## Régression [SATzilla 2009] :

- Modèle appris = temps d'exécution d'un algo sur une instance  
  ~> Un modèle par algorithme
- Utilisation pour résoudre une nouvelle instance  $i$   
  ~> Estimer le temps de résolution de  $i$  pour chaque algo  
  ~> Choisir l'algo pour lequel l'estimation est la plus petite

## Classification supervisée [Gent et al 2010] :

- Modèle appris = Classifieur (où classe = meilleur algorithme)
- Utilisation pour résoudre une nouvelle instance  $i$   
  ~> Calculer la classe de  $i$  et choisir l'algorithme correspondant

## Clustering [Kadioglu et al. 2010] :

- Modèle appris = partitionnement des instances en clusters homogènes
- Utilisation pour résoudre une nouvelle instance  $i$   
  ~> Chercher à quel cluster  $i$  appartient  
  ~> Choisir l'algorithme le mieux adapté en fonction du cluster

# Sélection vs planification d'algorithmes

## Sélection d'algorithmes :

- Etant donné une instance, choix d'un seul algorithme

## Planification d'algorithmes :

- Etant donné une instance, choix de plusieurs algorithmes
- Allocation de temps CPU à ces algorithmes :
  - En séquence
  - En parallèle

# Plan du cours

1 Validation théorique d'algorithmes

2 Evaluation expérimentale d'algorithmes

3 Ingénierie algorithmique

- Optimisation d'algorithmes et de code
- Outils pour optimiser algorithmes et code
- Optimisation de paramètres
- Illustration : Sélection de solveurs de contraintes

# Retour sur la comparaison de solveurs de contraintes

	CBT				CBJ				CBJR				DBT				DR				BTD			
	FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC		FC		MAC	
	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w	d	w
1	17.8	2.3	8.8	11.1	1.1	0.3	0.3	1.1	0.8	1.6	0.0	0.8	0.8	0.2	0.5	0.0	0.5	2.7	0.5	0.2	23.3	18.8	1.3	5.1
2	27.7	9.8	12.9	19.9	4.0	3.4	2.7	7.1	2.3	4.8	2.3	2.3	1.3	5.5	2.7	2.7	2.7	7.1	2.9	2.9	42.1	26.5	5.3	10.5
3	6.4	0.3	5.5	5.8	0	0	0.2	0.5	0	1.1	0	0	0	0	0	0	0	1.9	0	0	14.0	7.2	0.2	1.3
4	6.4	0.6	6.3	7.2	1.6	-	0.2	1.0	-	1.4	-	-	-	-	-	-	-	2.4	-	-	14.6	7.9	0.3	1.3

Pour chaque algorithme  $A$  :

- 1 % d'instances pour lesquelles  $A$  est le meilleur
- 2 % d'instances pour lesquelles  $A$  est bon
- 3 % d'instances pour lesquelles  $A$  est signif. meilleur que les autres
- 4 % d'instances pour lesquelles  $A$  est signif. meilleur que les autres sans les algo marqués par '-'

## Sélection automatique d'algorithmes :

- Constituer un portfolio d'algorithmes  $\rightsquigarrow$  13 algorithmes
- Apprendre un modèle capable de choisir un bon algo pour chaque instance

# Constitution d'un portfolios d'algorithmes complémentaires

## Résolution d'un *Set Covering Problem* :

- Entrée : entier  $1 \leq k \leq 13$
- Sortie : sous-ensemble  $S_k$  de  $k$  algos maximisant le nombre d'instances pour lesquelles  $S$  contient un bon algo

Ensemble d'algorithmes	Taux de succès d'un <i>virtual best algorithm</i>				
	1	10	100	1000	1800
$S_1 = \{(BTD, FC, d)\}$	31.4%	52.2%	69.2%	77.3%	78.0%
$S_2 = S_1 \cup \{(CBT, MAC, w)\}$	52.7%	77.2%	92.3%	98.2%	98.6%
$S_3 = S_2 \cup \{(BTD, FC, w)\}$	53.2%	77.8%	92.8%	98.4%	98.8%
$S_4 = S_3 \cup \{(CBT, FC, d)\}$	54.7%	78.5%	93.0%	98.4%	98.8%
$S_5 = S_4 \cup \{(CBT, MAC, d)\}$	55.8%	78.7%	93.0%	98.4%	98.8%
$S_6 = S_5 \cup \{(DR, FC, w)\}$	55.9%	79.0%	93.6%	98.8%	99.3%
$S_7 = S_6 \cup \{(CBJ, MAC, w)\}$	56.0%	79.0%	93.9%	99.2%	99.6%
$S_8 = S_7 \cup \{(BTD, MAC, w)\}$	56.1%	79.2%	93.9%	99.2%	99.6%
$S_9 = S_8 \cup \{(CBJ, FC, d)\}$	56.4%	79.3%	94.1%	99.2%	99.6%
$S_{10} = S_9 \cup \{(CBJR, FC, w)\}$	56.5%	79.5%	94.2%	99.4%	99.7%
$S_{11} = S_{10} \cup \{(CBT, FC, w)\}$	56.5%	79.6%	94.2%	99.4%	99.7%
$S_{12} = S_{11} \cup \{(BTD, MAC, d)\}$	56.5%	79.6%	94.2%	99.4%	99.7%
$S_{13} = S_{12} \cup \{(CBJ, MAC, d)\}$	56.6%	79.6%	94.3%	99.4%	99.7%

# Classification avec Weka

## Protocole expérimentale :

- Leave-one-out cross-validation
- Mesure du pourcentage d'instances bien classées (Rang=1)  
 ~ Si  $i$  mal classée : rang = rang de l'algorithme sélectionné pour  $i$
- Mesure du pourcentage d'instances tq l'algo sélectionné est bon

Rang	1	2	3	4	5	6	$\geq 7$	# bon
$S_2$	84.6	15.4						54.5
$S_3$	77.3	16.2	6.4					56.4
$S_4$	75.7	15.8	6.8	1.8				61.4
$S_5$	75.1	14.6	5.9	3.2	1.1			67.2
$S_6$	71.5	13.8	7.1	5.0	2.3	0.3		66.4
$S_7$	71.2	11.9	5.5	3.9	4.7	2.6	0.3	67.8
$S_8$	70.7	10.5	5.8	5.8	3.1	1.9	2.3	69.0
$S_9$	67.7	9.8	5.6	7.1	3.9	2.4	3.5	67.2
$S_{10}$	66.1	10.9	5.9	4.0	3.2	3.5	6.3	66.5
$S_{11}$	66.7	10.5	6.1	3.9	2.7	2.9	7.2	66.6
$S_{12}$	65.8	10.5	6.8	3.7	3.2	2.1	8.1	65.6
$S_{13}$	65.4	9.8	6.8	3.4	3.9	1.8	8.9	65.4

## Evolution du pourcentage d'instances résolues / temps

k	1	10	100	1000	1800
1	47.1	68.3	85.2	94.3	95.4
2	47.1	73.9	90.2	96.3	96.8
3	47.1	73.8	90.1	96.2	96.6
4	47.1	74.6	89.9	96.3	96.8
5	47.1	75.1	89.9	96.5	96.9
6	47.1	75.0	89.7	95.9	96.4
7	47.1	74.5	89.4	95.9	96.5
8	47.1	74.7	89.7	95.7	96.3
9	47.1	74.6	89.4	95.5	96.1
10	47.1	73.9	88.9	95.2	95.7
11	47.1	74.3	88.8	95.3	95.7
12	47.1	74.4	89.2	95.7	96.2
13	47.1	74.5	89.2	95.7	96.3

# Conclusions ?

- Validation théorique d'algorithmes
  - Etudier la complexité du problème avant de concevoir un algorithme
  - Prouver la correction des algorithmes
  - Etudier leur complexité théorique
- Evaluation expérimentale d'algorithmes
  - Bien choisir les jeux d'essai, facteurs, *Design Points*, et indicateurs de performance
  - Analyser soigneusement les résultats
- Ingénierie algorithmique
  - *Algorithm tuning vs code tuning*
    - ↪ Trouver le bon compromis entre efficacité et lisibilité
  - *Parameter tuning*
    - ↪ Utiliser des outils pour le paramétrage automatique

# Le(s) mot(s) de la fin par Donald Knuth

*If you find that you're spending almost all your time on theory, start turning some attention to practical things ; it will improve your theories. If you find that you're spending almost all your time on practice, start turning some attention to theoretical things ; it will improve your practice.*

*Computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better.*

*We should continually be striving to transform every art into a science : in the process, we advance the art.*

*An algorithm must be seen to be believed.*

