



# Techniques d'optimisation

hepia HES-SO

Guido Bologna

Paul Albuquerque

Michel Vinckenbosch

# Sujets abordés

- Introduction aux algorithmes de recherche
- Satisfaction de contraintes
- Apprentissage statistique
- Réseaux de neurones artificiels
- Recuit simulé
- Algorithmes génétiques
- Programmation linéaire

# But du cours

- Maîtriser les modèles et algorithmes étudiés de telle sorte à pouvoir déterminer pour un problème d'optimisation donné (défini par exemple par un futur employeur) quel est la technique la plus appropriée à appliquer

# Algorithme de recherche : puzzle-8

8	2	
3	4	7
5	1	6

État initial

1	2	3
4	5	6
7	8	

État final

État: n'importe quel arrangement des 8 plaquettes numérotées et de la case vide sur un damier 3x3

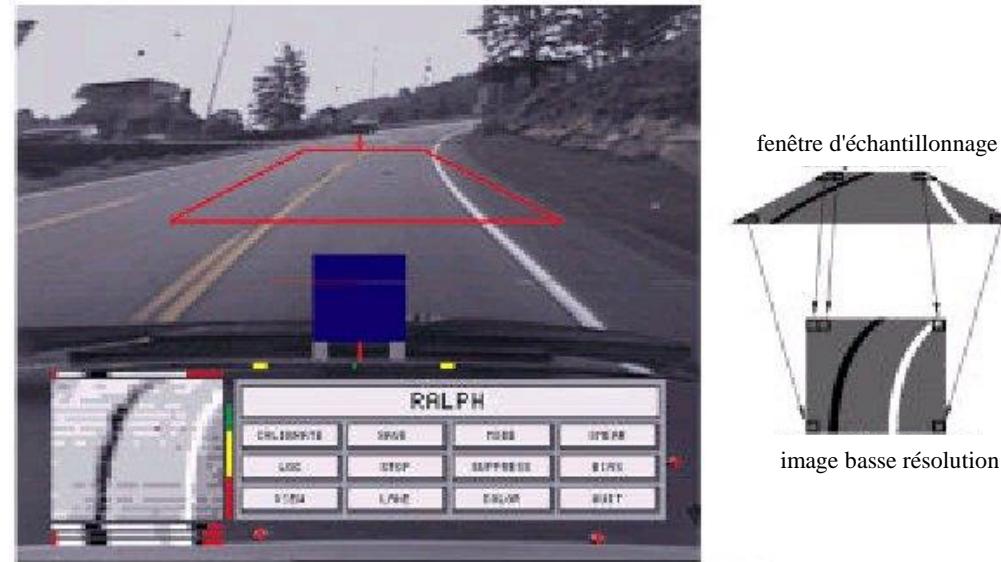
# Exemple: coloration de carte par propagation des contraintes



- Les **solutions** sont des affectations satisfaisant toutes les contraintes (couleurs différentes entre les territoires voisins)

# Ralph: un agent de conduite automatique obtenu par apprentissage statistique

- échantillonne l'image de la route devant le véhicule,
- détermine le rayon de courbure de la route,
- détermine le décalage latéral du véhicule par rapport à la ligne médiane de la route,
- sélectionne l'action de braquage calculée à partir de la courbure de la route et de la position estimée du véhicule sur la route.



Rapidly Adapting Lateral Position Handler

# No hands across USA !



- 2850 miles entre Washington DC et San Diego, sur autoroutes.
- défis du voyage: conduite de nuit, sous la pluie, sur des routes au marquage effacé ou en construction, etc.
- mode d'évaluation: pourcentage de la distance totale parcourue sous le contrôle du système *Ralph*,
- résultat: *Ralph* a conduit le véhicule sur 2796 des 2850 miles du trajet (98.1%):
  - 10 miles de nouvelle route sans marque (ligne centrale, bordures),
  - conduite en ville avec des marquages manquants ou cachés par d'autres véhicules.

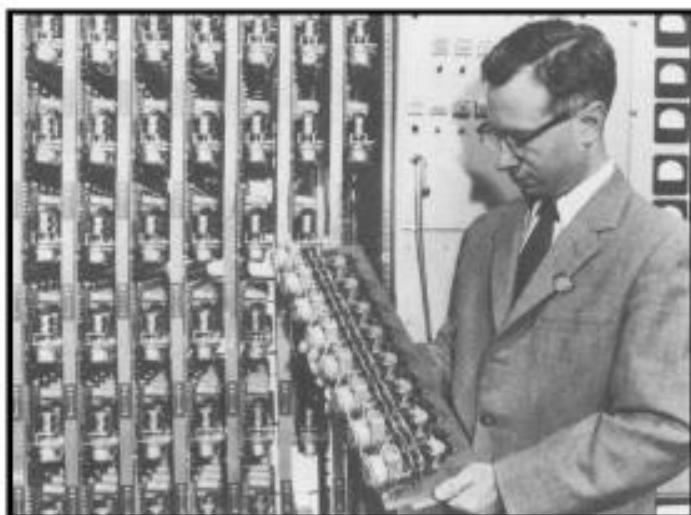
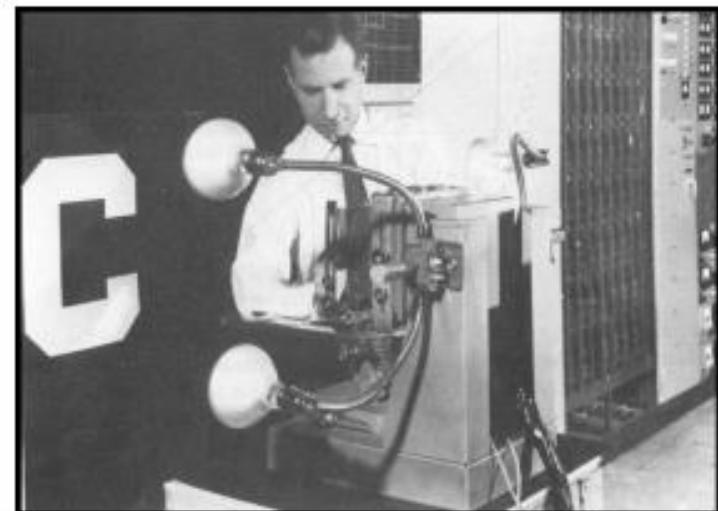
# Réseaux de neurones

- Différents types de réseaux de neurones artificiels
  - Perceptron, perceptron multicouche, réseaux de Hopfield, cartes de Kohonen
- Inspirés des neurones biologiques
- Processeurs parallèles et distribués
- Perceptron multicouche
  - apprentissage par rétropropagation des erreurs

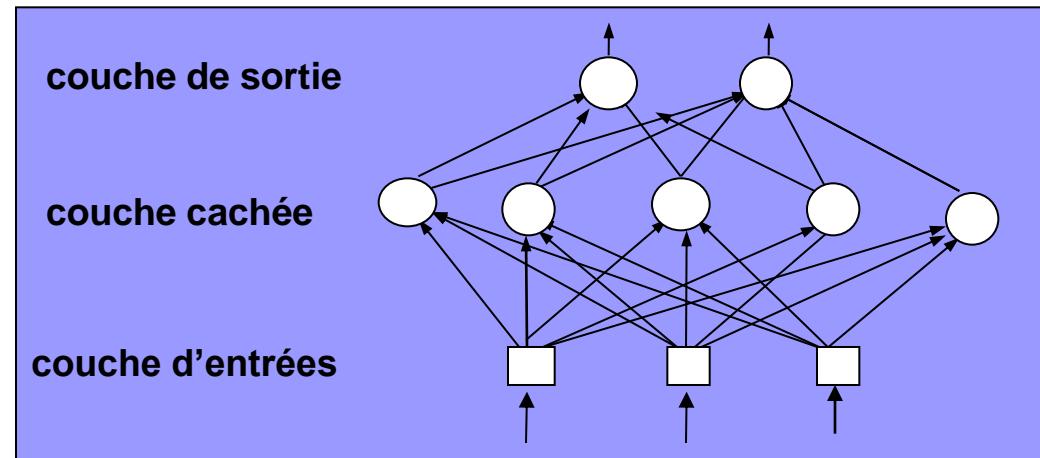
# *Le Mark I Perceptron*



*F. Rosenblatt*



# Réseau multicouche



- Propriétés des réseaux de neurones formels
  - approximation universelle
  - apprentissage
  - généralisation

# Réseaux de neurones

## Applications

- classification (reconnaissance de caractères)
- approximation de fonction
- traitement du signal, d'images
- commande de processus non linéaires
- prédiction de phénomènes: physiques, économiques
- robotique
- mémoire associative

# Formulation et terminologie

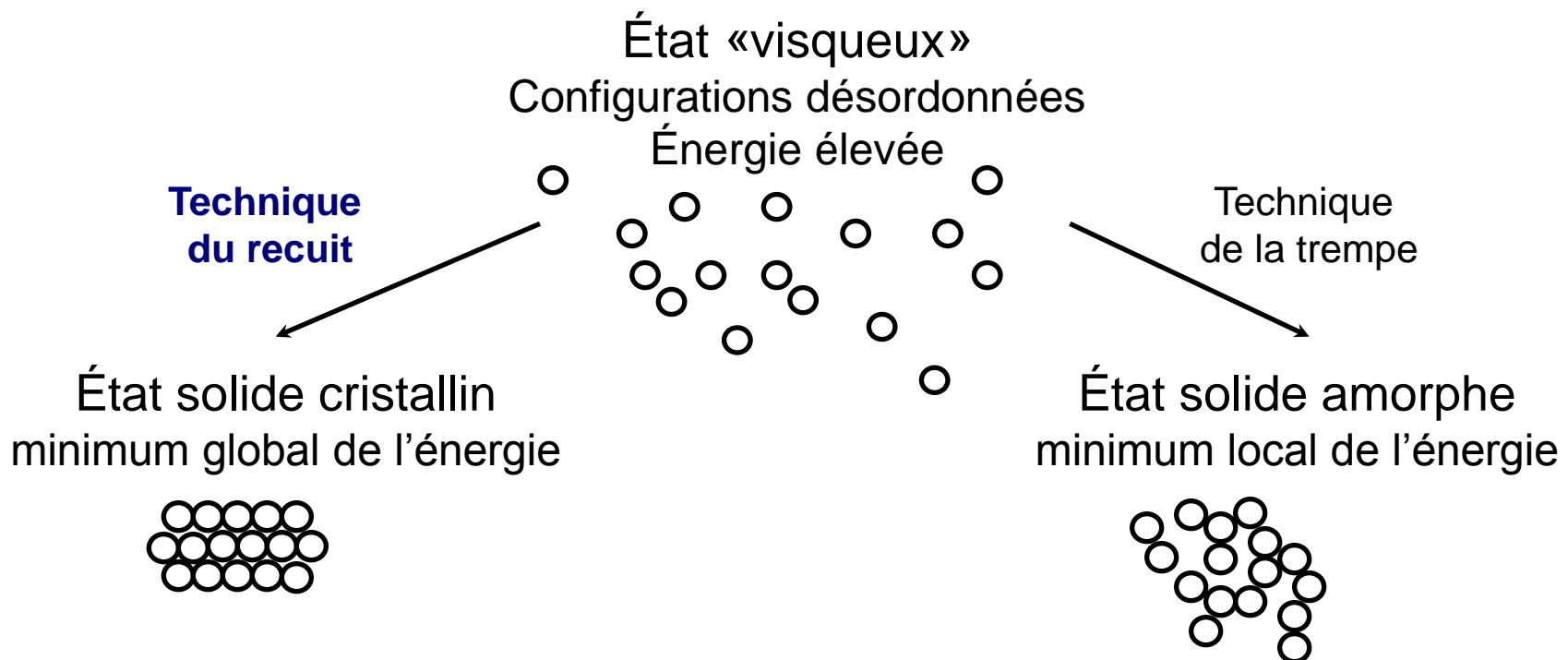
- Description générale d'un problème d'optimisation
  - $\Omega$  : **ensemble de configurations** (ou de solutions) du problème
  - $C$  : **fonction de coût** (ou objectif)
  - Trouver la configuration  $x' \in \Omega$  qui minimise le coût minimal

$$C(x') = \min \left\{ C(x) \mid x \in \Omega \right\}$$

- $x'$  n'est pas obligatoirement unique;
- la **fonction de coût** est aussi appelée ***fitness function***
- $\Omega$  est aussi appelé l'**espace de recherche**

# Le recuit simulé

- Origine du recuit simulé (*simulated annealing*)
  - Recuit métallurgique ou cristallisation d'un liquide



# Principe du recuit

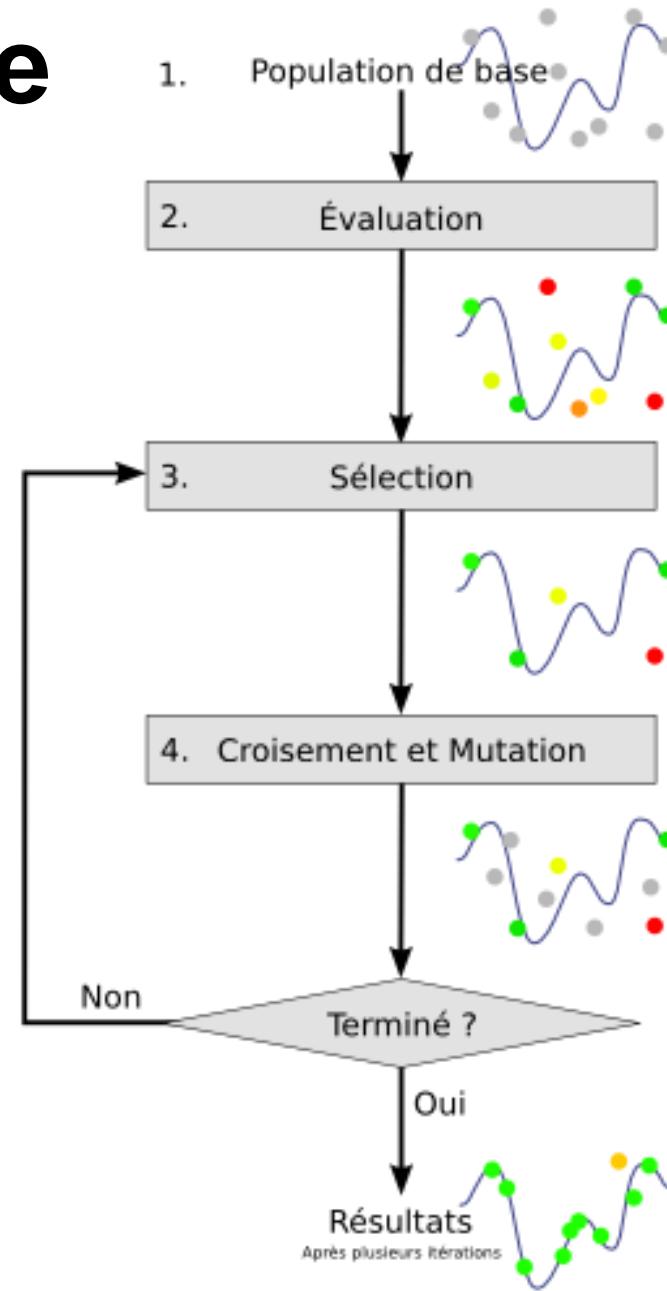
- Initialement le métal est porté à haute température
- Ensuite on le refroidit progressivement:
  - à haute température les atomes sont agités (configurations atomiques équiprobables)
  - à basse température les atomes s'organisent selon une structure atomique parfaite, proche de l'état d'énergie minimale
- Contrainte
  - Le refroidissement doit être lent pour ne pas rester bloqué dans un minimum local
  - un refroidissement trop rapide aboutirait à une configuration sous-optimale

# Algorithme génétique (AG)

## Principes

- Individu
  - une configuration (solution admissible) du problème à traiter
- Fonction d'évaluation / adaptation (*fitness function*)
  - Fonction objectif à optimiser, permettant d'évaluer l'adaptation d'un individu
- Population
  - ensemble d'individus évoluant simultanément
- Génération
  - itération de la boucle de base: sélection, croisement, mutation

# Algorithme génétique

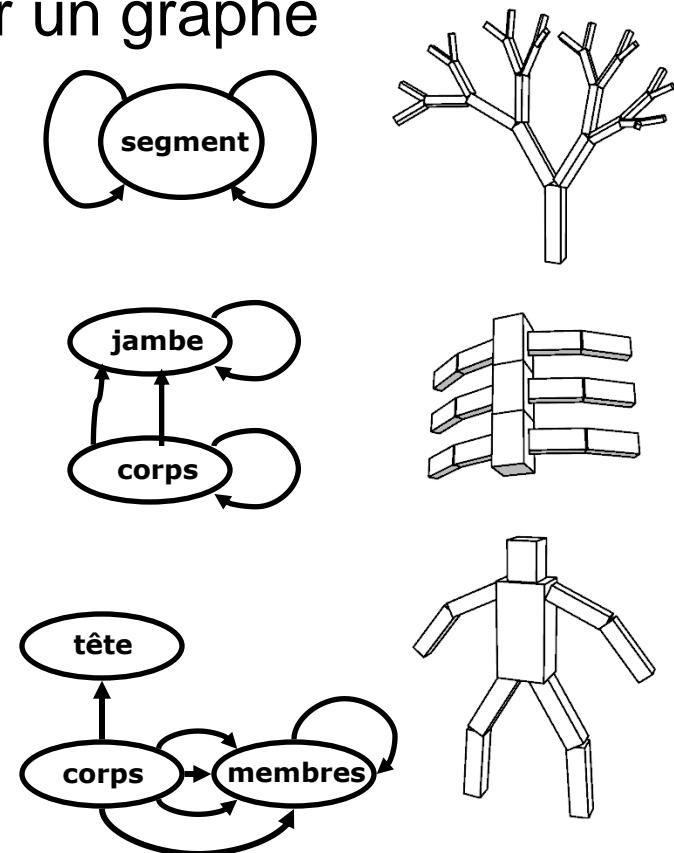


# Résoudre le Soduku par AG

8			4	6			7
					4		
1					6	5	
5	9		3		7	8	
			7				
4	8		2		1		3
5	2				9		
1							
3			9	2			5

# Evolved Virtual Creatures

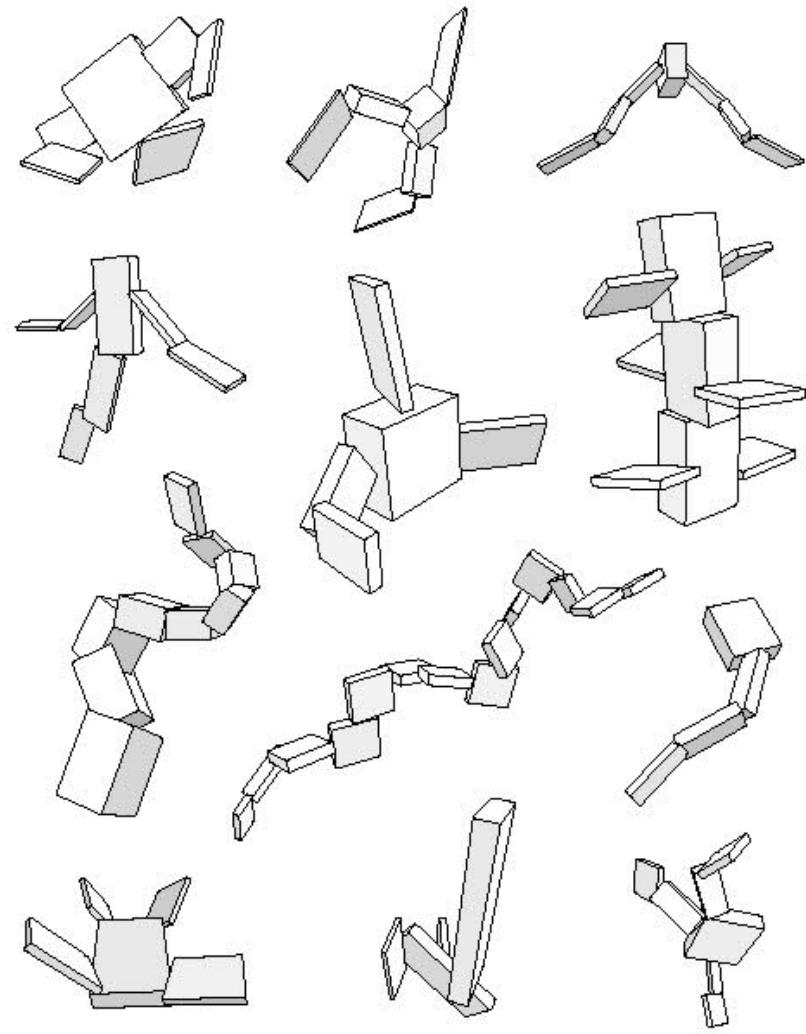
- Chaque « créature » est définie par un graphe
  - Un nœud = un « bloc corporel »
  - Un lien = une articulation
  - Liens récursifs = structures répétitives
- Les nœuds et les liens sont valués
  - Dimensions
  - Limites articulaires
  - position/orientation
  - limites de récursivité
  - contrôle de l 'articulation



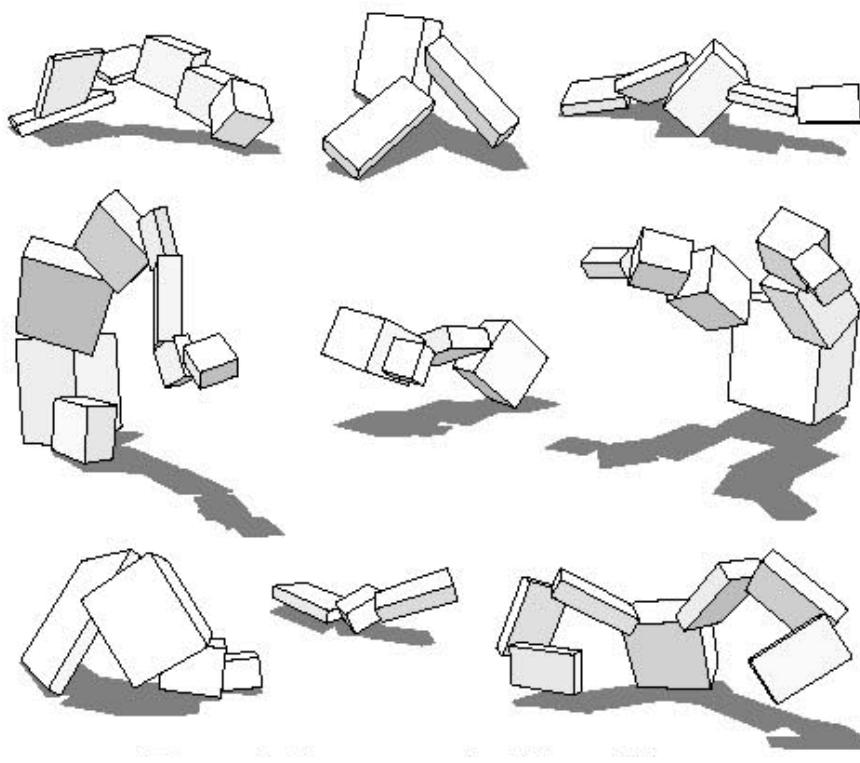
↳ Les régularités structurelles sont « prévues »

# Evolving Virtual Swimmers (Karl Sims)

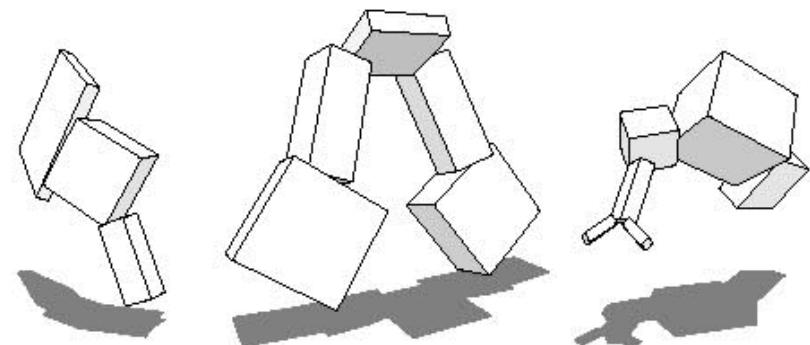
- Environnement :
  - Visqueux sans gravité
- Fitness :
  - Distance parcourue
- Trois types de formes :
  - Rames + gouvernail
  - Nageoires symétriques
  - « serpents marins »



# Evolving Virtual Walkers/Jumpers (Karl Sims)



- Environnement :
  - « Grave » sans viscosité
- Fitness :
  - distance parcourue
  - hauteur atteinte
- Formes très variées :
  - déplacement basé sur le saut
  - reptation
  - « début de bipédie »



# Programmation linéaire: exemple

- $H$  hectares de terres, semés de blé et de maïs.
- Quantités totales  $E$  d'engrais et  $I$  d'insecticide.
  - Quantité par hectare:  $E_1$ , d'engrais et  $I_1$ , d'insecticide pour le blé,  
 $E_2$  et  $I_2$  pour le maïs
  - prix de vente du blé  $P_1$ ; nombre d'hectares à planter en blé  $x_1$ ;  
 $P_2$ ,  $x_2$  pour le maïs
- Nombre optimal d'hectares à planter en blé et en maïs?
  - maximiser  $P_1x_1 + P_2x_2$  (maximiser le revenu net)
  - sous contraintes
    - $x_1 + x_2 \leq H$       borne sur le nombre total d'hectares
    - $E_1x_1 + E_2x_2 \leq E$       borne sur la quantité d'engrais
    - $I_1x_1 + I_2x_2 \leq I$       borne sur la quantité d'insecticide
    - $x_1 \geq 0, x_2 \geq 0$

# Optimisation de systèmes linéaires

## ■ Programmation linéaire

- En mathématiques, les problèmes de **programmation linéaire** (PL) sont des problèmes d'optimisation où la **fonction de coût** et les **contraintes** sont **linéaires**.
- La PL est un domaine utile de l'optimisation, car les problèmes de PL sont les problèmes d'optimisation les plus faciles.

# Optimisation combinatoire

## Problème du voyageur de commerce

- Quel est le plus court circuit pour visiter n villes en passant une seule fois par ville ?
- $(n-1)!/2$  chemins possibles

Nb villes	Nb possibilités	Temps (estimation)
5	12	12 microsecondes
10	181440	0.18 seconde
15	43 milliards	12 heures
20	6 E+15	1928 ans
25	310 E+21	9.8 milliards d'années

# Exemple : Optimisation combinatoire ; problème du sac à dos (knapsack)

- Remplir un sac à dos sans excéder sa capacité en maximisant la valeur de son contenu
  - capacité du sac  $M = 9$  (poids max. entier)
  - objets  $x_i \in \{0,1\}$  (présent, absent),  $i \in [1..n]$
  - poids de  $x_i$  :  $p_i$
  - valeur de  $x_i$  :  $v_i$

Objet x	v	p
Conserve	15	4
Sucre	10	3
Pain	7	2

Variables :  $x_c, x_s, x_p \in \{0,1\}$

$$4x_c + 3x_s + 2x_p \leq 9$$

$$15x_c + 10x_s + 7x_p \geq 30$$

# Formulation du problème du sac à dos

- Peut-on emporter une valeur totale  $V$  supérieure à 30 sans dépasser la capacité
- Trouver une solution  $(x_1, \dots, x_n)$  telle que

la fonction  $V = \sum_{i=1}^n x_i v_i$  est maximale

sous contraintes  $\sum_{i=1}^n x_i p_i \leq 9$

$$\sum_{i=1}^n x_i v_i \geq 30$$

# Heuristiques et mét-heuristiques

- Une **heuristique** est une méthode pour résoudre de manière approchée un type de problème particulier, dont la solution optimale exacte ne peut être obtenue (car, par exemple, son obtention nécessiterait un temps de calcul d'ordinateur, de plusieurs milliers d'années)
- Une **métaheuristique** est une méthode, ou plus précisément, un canevas de méthodes, pour résoudre de manière approchée tous les problèmes dont la solution optimale ne peut être obtenue. La méthode ne dépend donc plus du type de problème auquel on est confronté

# Références

Métaheuristiques pour l'optimisation difficile

Johann Dréo – Alain Pétrowski

Patrick Siarry – Eric taillard

Eyrolles 2003

Stuart J. Russell, Peter Norvig

*Artificial Intelligence : A Modern Approach (3<sup>ème</sup> édition)*

Prentice Hall, 2010

# Exigences

Partie pratique : 2 TP notés + 2 TP à rendre (non notés)

Partie théorique : une épreuve écrite

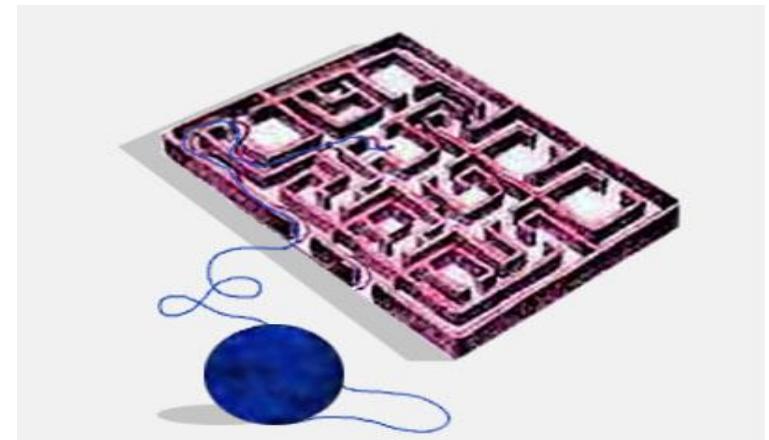
Moyenne = (Note pratique + Note théorique) / 2

# RECHERCHE AVEUGLE ET HEURISTIQUE

# *La recherche exploratoire est une vieille idée*

## **Le labyrinthe et le Fil d'Ariane**

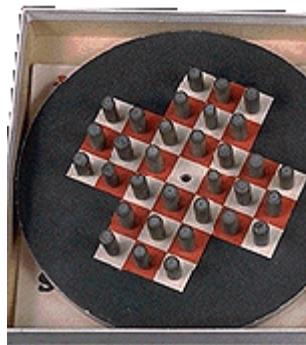
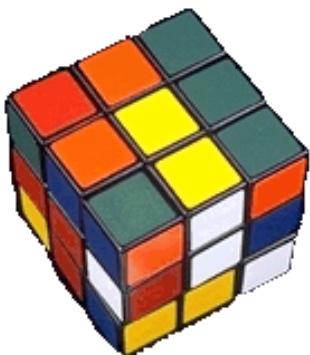
Selon la mythologie grecque, Thésée vint en Crète pour tuer le Minotaure, un monstre qui vivait dans un labyrinthe. Ariane donna à Thésée un fil qu'il déroula en parcourant de labyrinthe. Après avoir tué le Minotaure, Thésée retrouva son chemin vers l'entrée en suivant le fil, rejoignit Ariane et s'échappa de Crète.



# *Recherche exploratoire*

L'histoire des civilisations montre que les puzzles et les jeux qui demandent d'explorer des solutions alternatives ont fasciné les hommes et ont été considérés comme un défi pour l'intelligence humaine:

- Les échecs: originaire de Perse et des Indes il y a environ 4000 ans
- Le jeu de dames apparaît il y a 3600 ans sur des fresques égyptiennes
- Le jeu de Go apparaît en Chine il y a plus de 3000 ans



## Exemple: puzzle-8

8	2	
3	4	7
5	1	6

État initial

1	2	3
4	5	6
7	8	

État final

État: n'importe quel arrangement des 8 plaquettes numérotées et de la case vide sur un damier 3x3

# Espace d'états pour le puzzle-8

8	2	
3	4	7
5	1	6

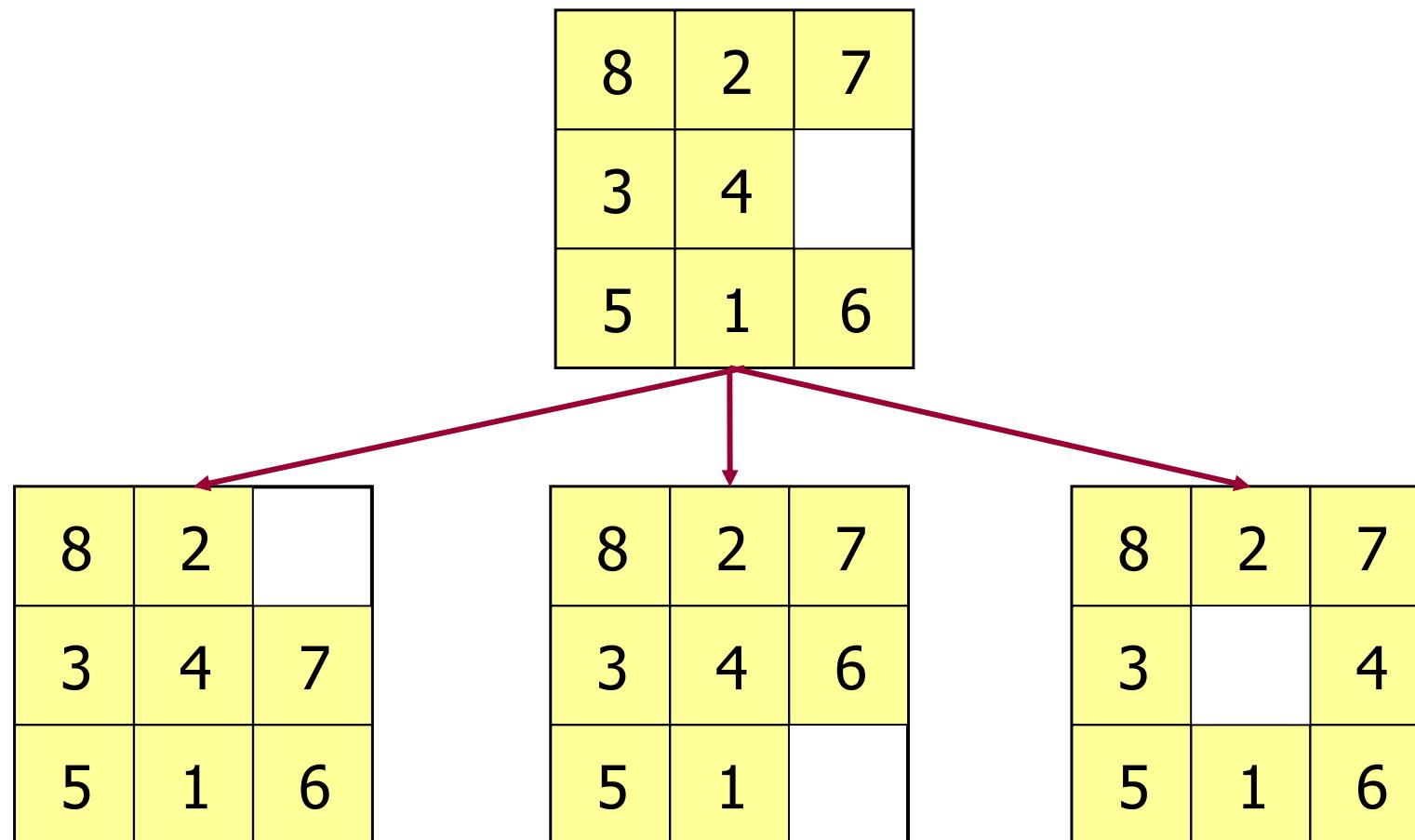
8	2	7
3	4	
5	1	6



8		2
3	4	7
5	1	6

	8	2
3	4	7
5	1	6

## Fonction "successeur" du puzzle-8



# Introduction aux algorithmes de recherche

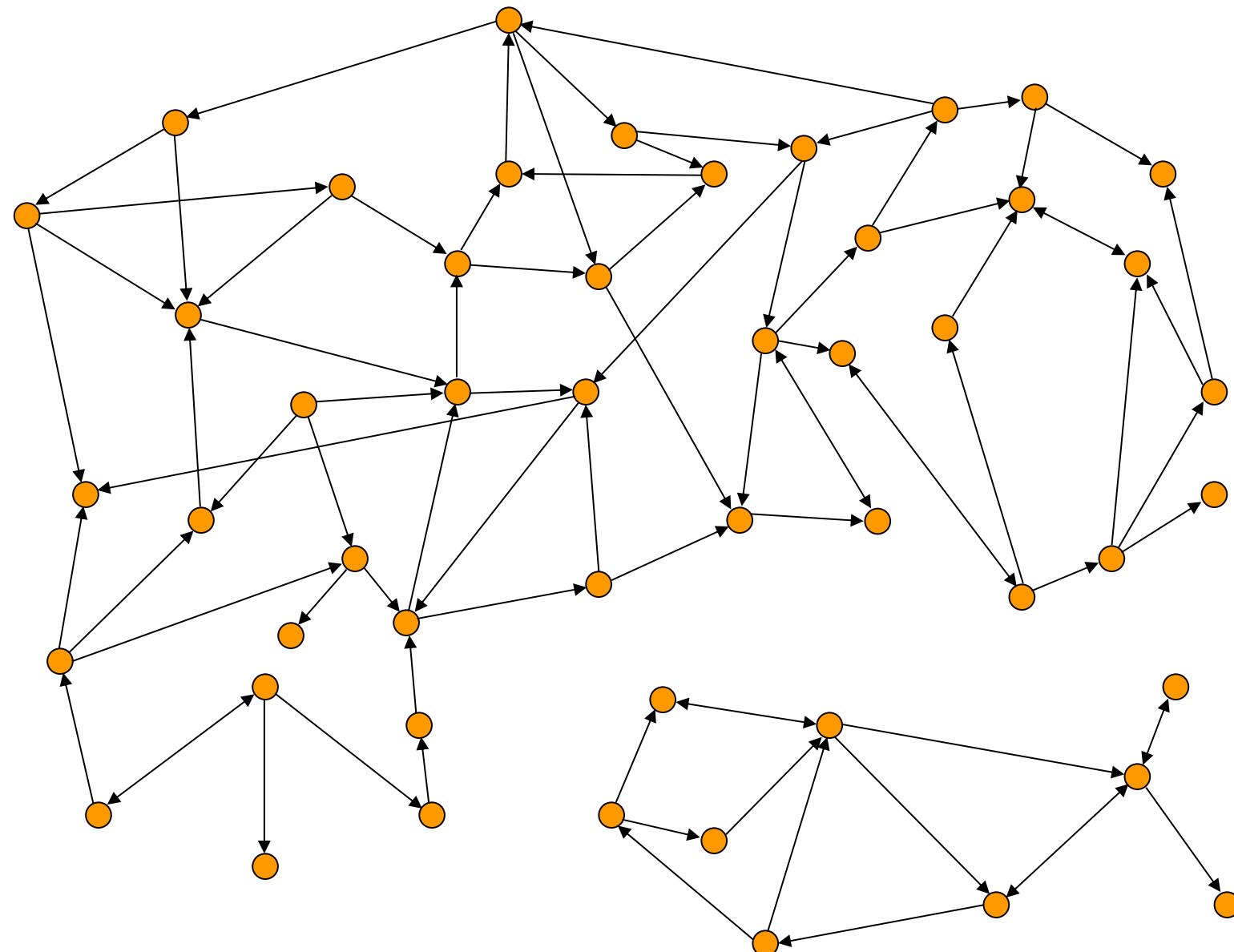
---

- Les algorithmes de recherche sont une technique générale de résolution de problèmes qui:
  - évolue dans un espace appelé *espace d'états*
  - explore systématiquement toutes les *alternatives*
  - trouve la *séquence d'étapes* menant à la solution
- "Problem Space Hypothesis"  
(Allen Newell, SOAR: An Architecture for General Intelligence, 1987)
  - toute recherche orientée vers un but se déroule dans *l'espace du problème (espace d'états)*

## Graphe d'états

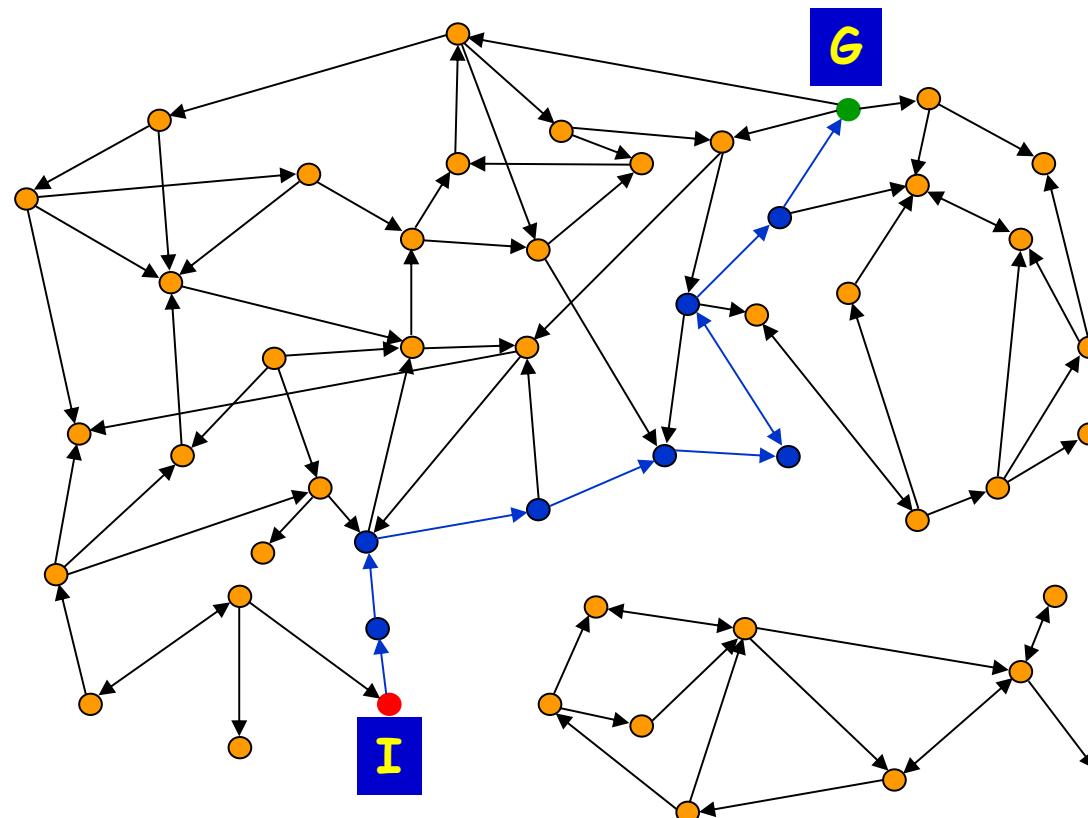
- Défini comme:
  - Chaque état est représenté par un nœud distinct
  - Un arc (ou arrête) relie un nœud  $s$  à un nœud  $s'$  si
$$s' \in \text{SUCCESEURS}(s)$$
- Le graphe d'états peut contenir plus d'une composante connexe

# Graphe d'états



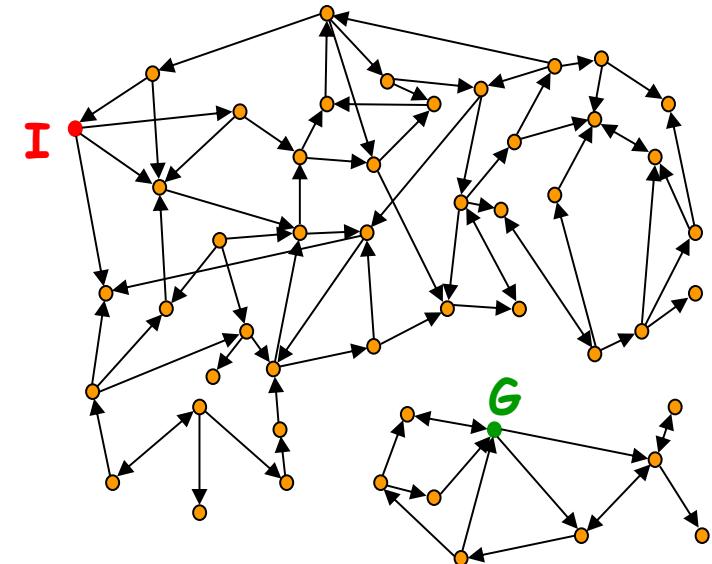
# Solution à un problème de recherche

- Une **solution** est un chemin reliant l'état initial I à un état solution G (n'importe lequel)



# *Solution à un problème de recherche*

- Une **solution** est un chemin reliant l'état initial  $I$  à un état solution  $G$  (n'importe lequel)
- Le **coût** d'un chemin est la somme des coûts des arcs qui le constituent
- Une solution **optimale** est un chemin-solution de coût minimum (la plupart du temps) ou maximum
- Il peut ne pas y avoir de solution !!!



# Quelle est la taille de l'espace de recherche ?

---

Exemple pour un puzzle- $(n^2-1)$

- puzzle-8 → **combien d'états ???**

# Quelle est la taille de l'espace de recherche ?

---

Exemple pour un puzzle

- puzzle-8 →  $9! = 362'880$  états
- puzzle-15 →  $16! \sim 2.09 \times 10^{13}$  états
- puzzle-24 →  $25! \sim 10^{25}$  états

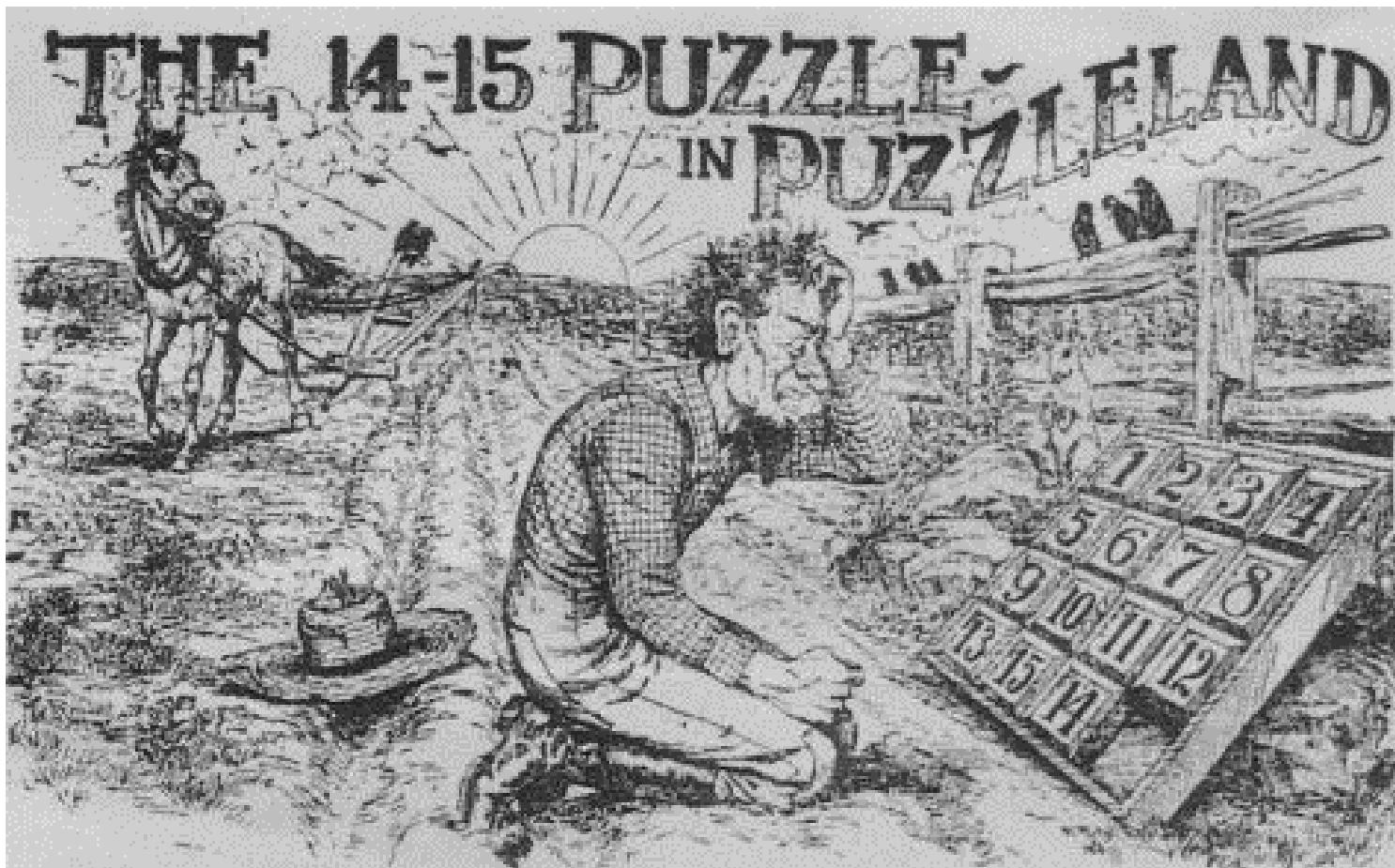
Mais seule la moitié de ces états est accessible à partir de  
n'importe quel état donné !  
(mais on peut ne pas savoir ça d'avance)

# Puzzle

8	2	
3	4	7
5	1	6

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	





Personne n'a jamais remporté le prix !!

## Quel est alors l'espace de recherche ?

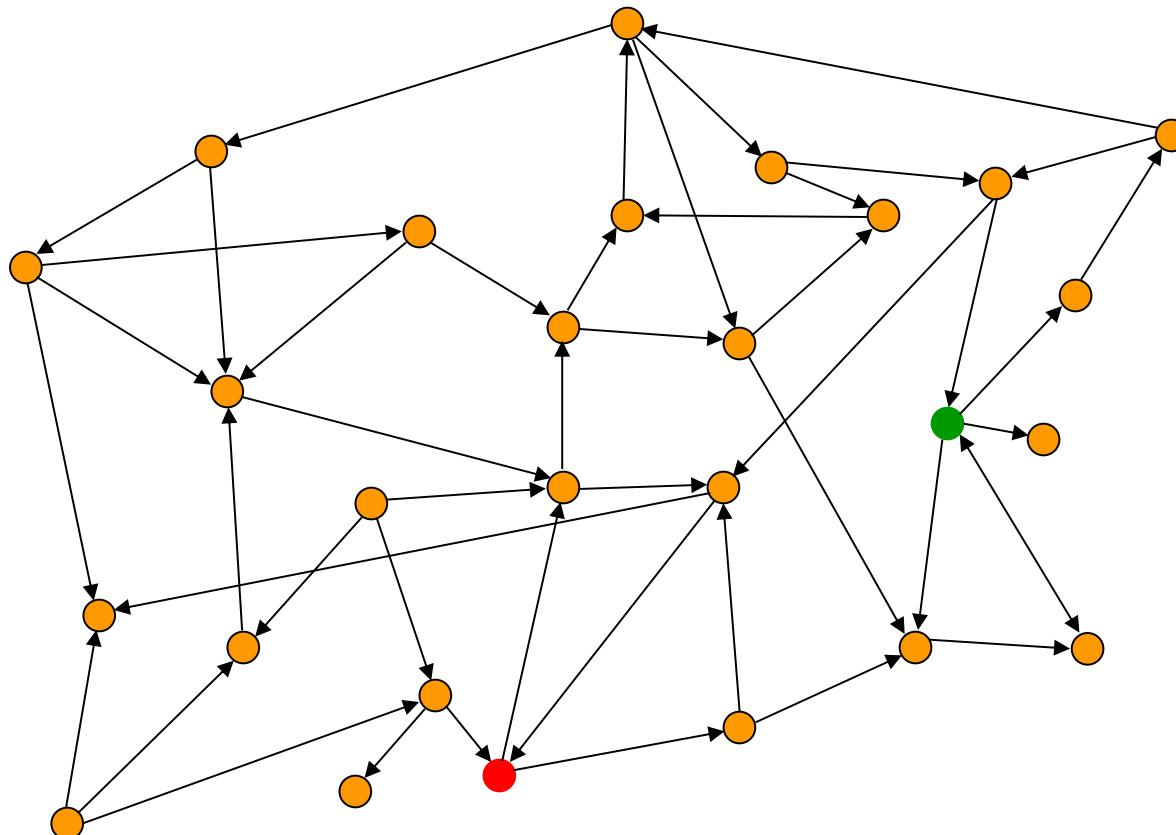
---

- a) L'ensemble de tous les états?  
[càd, un ensemble de  $16!$  états pour le puzzle-15]
- b) L'ensemble de tous les états accessibles à partir d'un état initial donné?  
[càd, un ensemble de  $16!/2$  états pour le puzzle-15]

En général la réponse est a)

[car on ne connaît pas à l'avance quels sont les états accessibles]

# Recherche dans un espace d'états



- En général, il est souvent impossible (ou trop coûteux) de construire une représentation complète du graphe d'états

# Puzzle-8, -15, -24

puzzle-8 → 362,880 états

0.036 sec

puzzle-15 →  $2.09 \times 10^{13}$  états

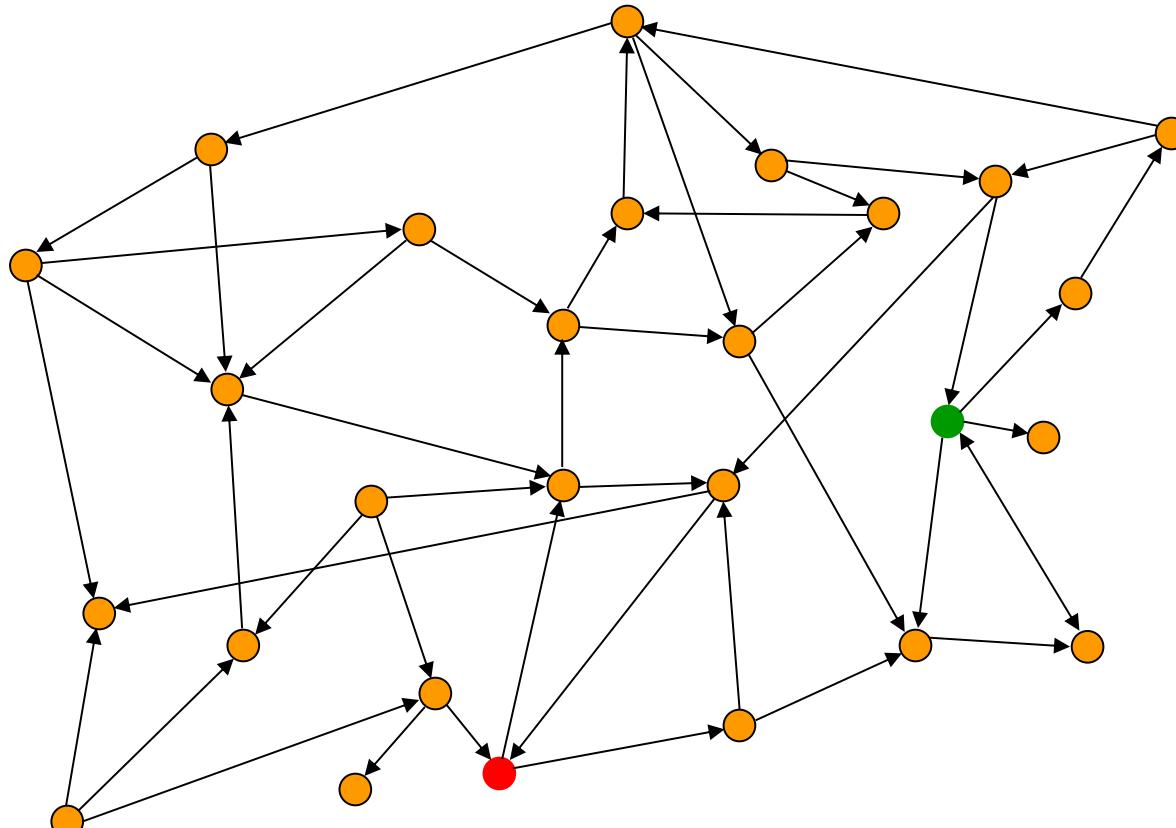
~55 heures

puzzle-24 →  $10^{25}$  états

>  $10^9$  années

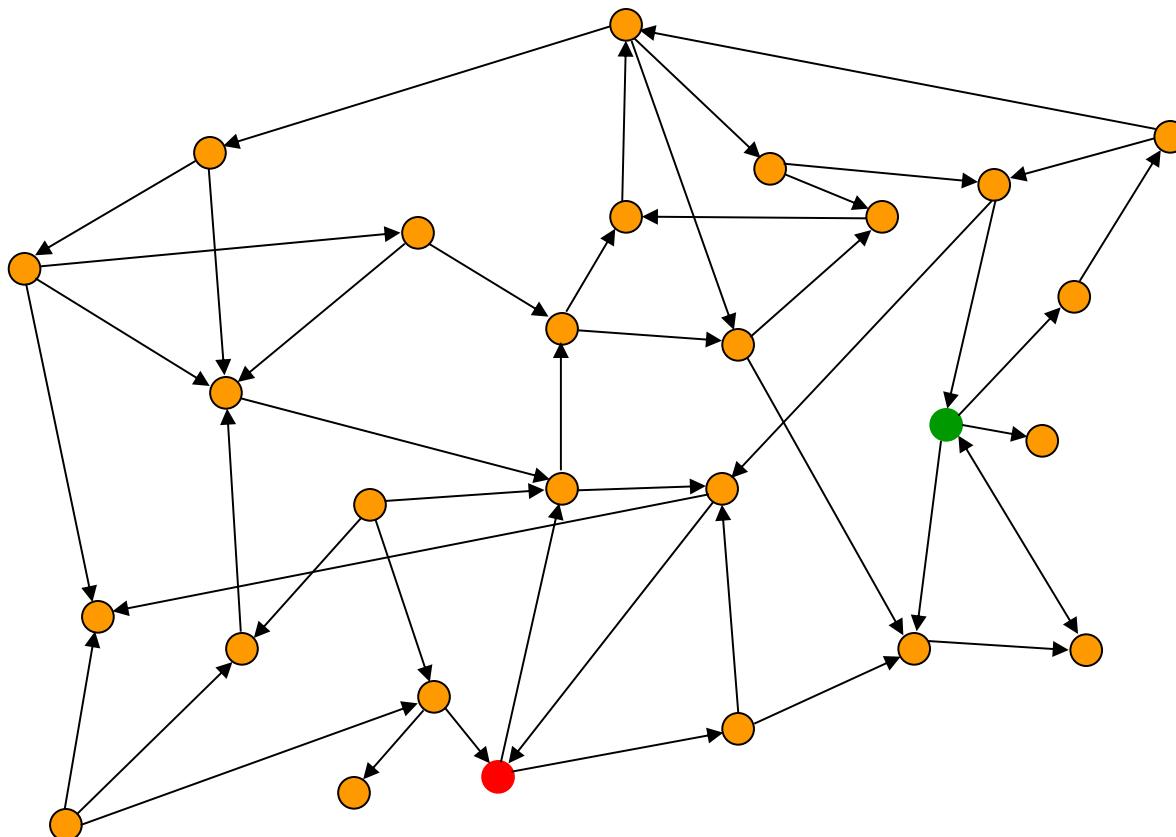
100 millions états/sec

# Recherche dans un espace d'états

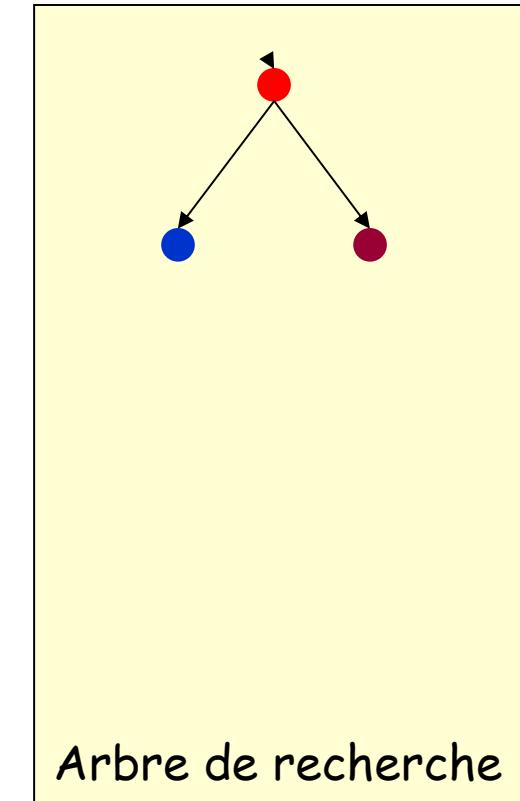
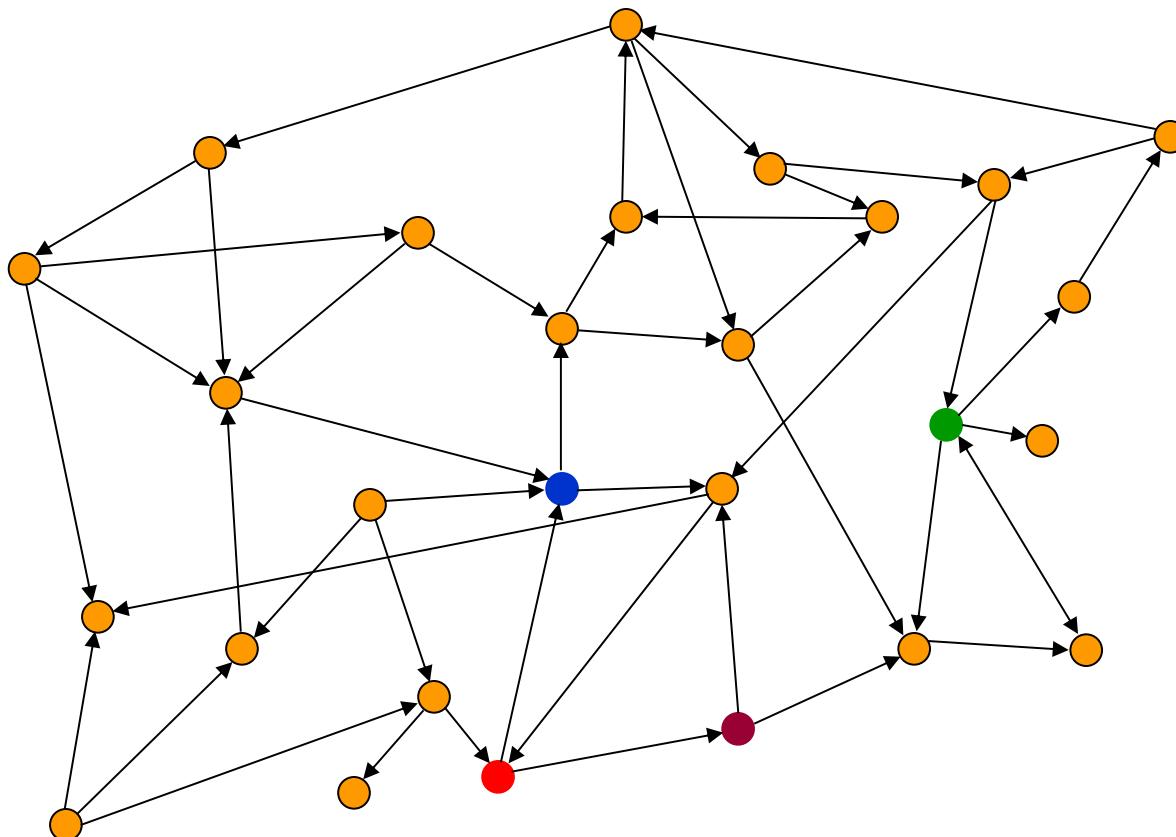


- Souvent il est impossible (ou trop coûteux) de construire une représentation complète du graphe d'états
- On doit donc construire une solution en n'explorant qu'une petite portion du graphe

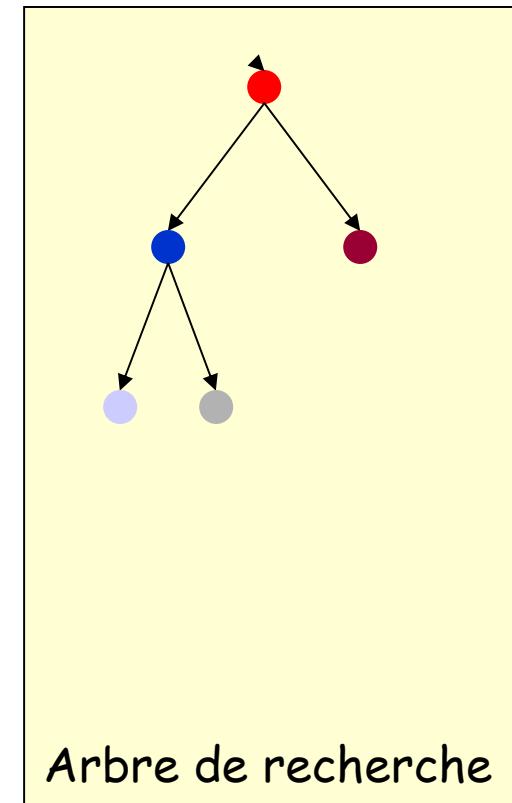
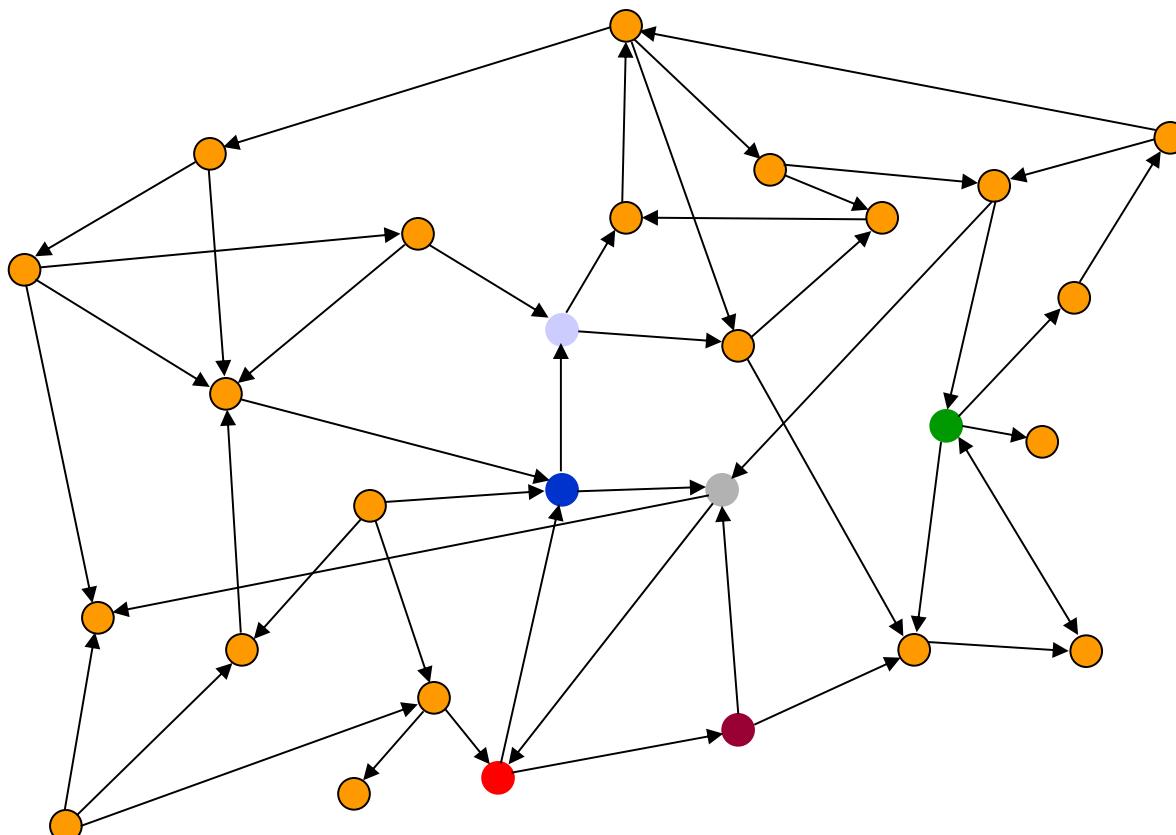
# Recherche dans un espace d'états



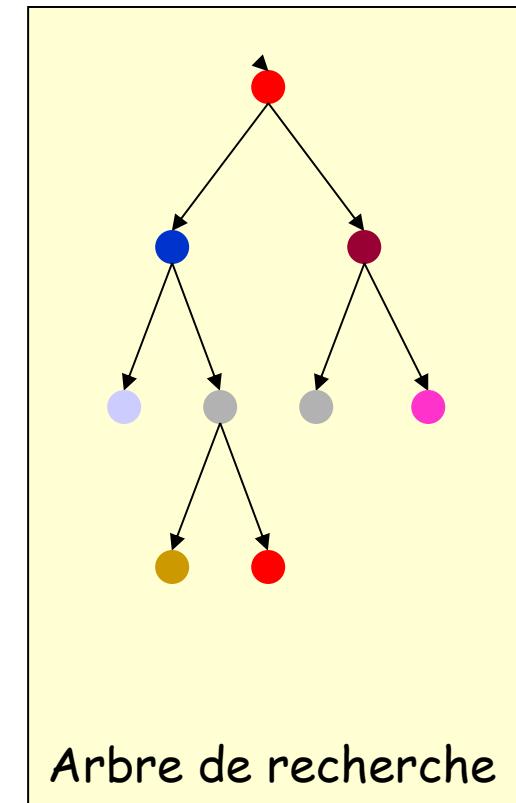
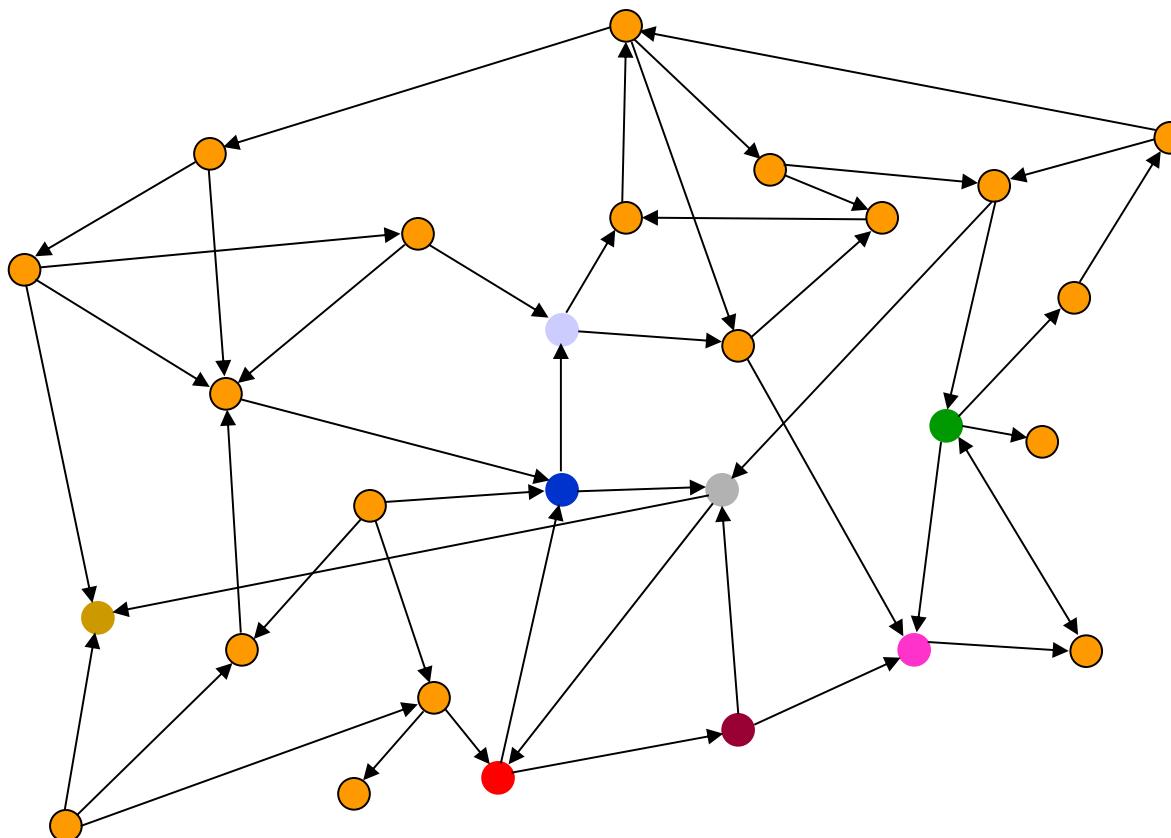
# Recherche dans un espace d'états



# Recherche dans un espace d'états

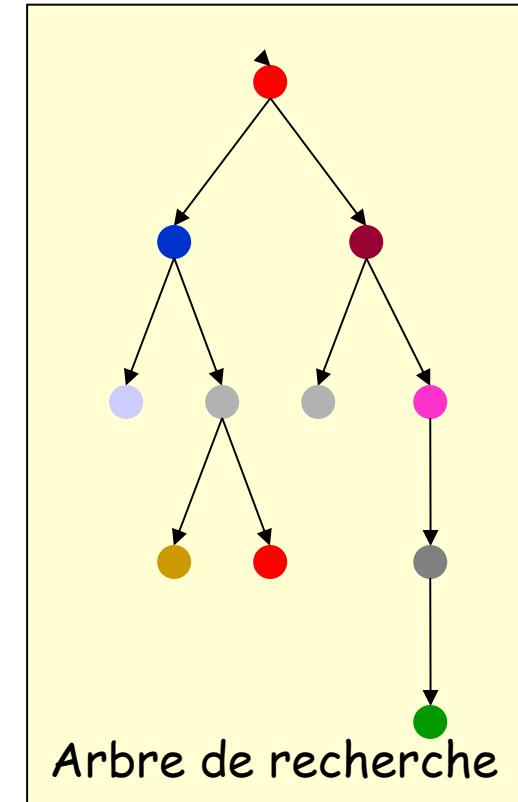
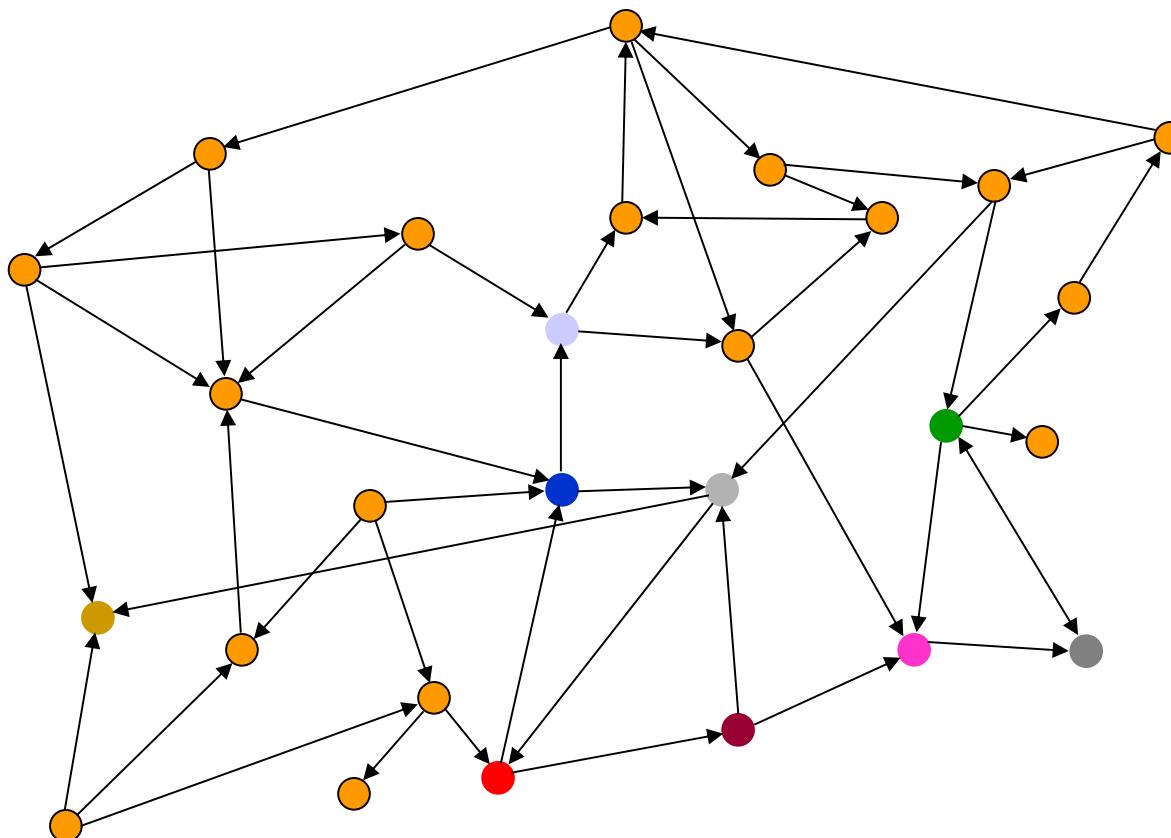


# Recherche dans un espace d'états

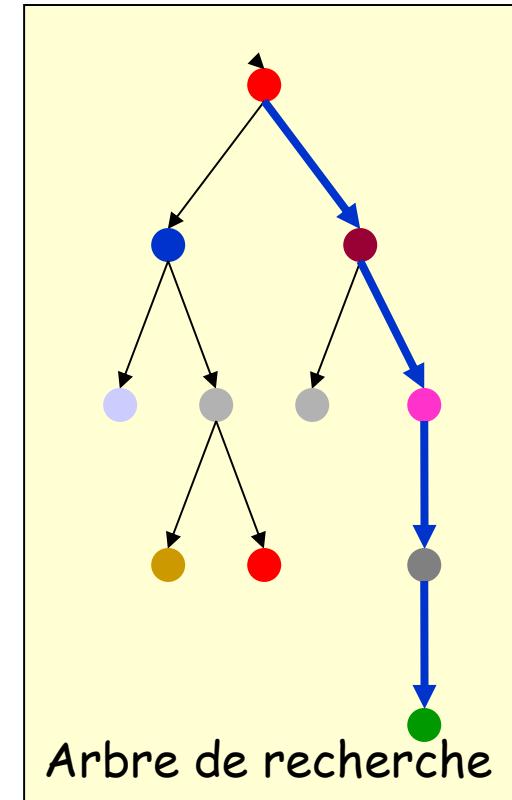
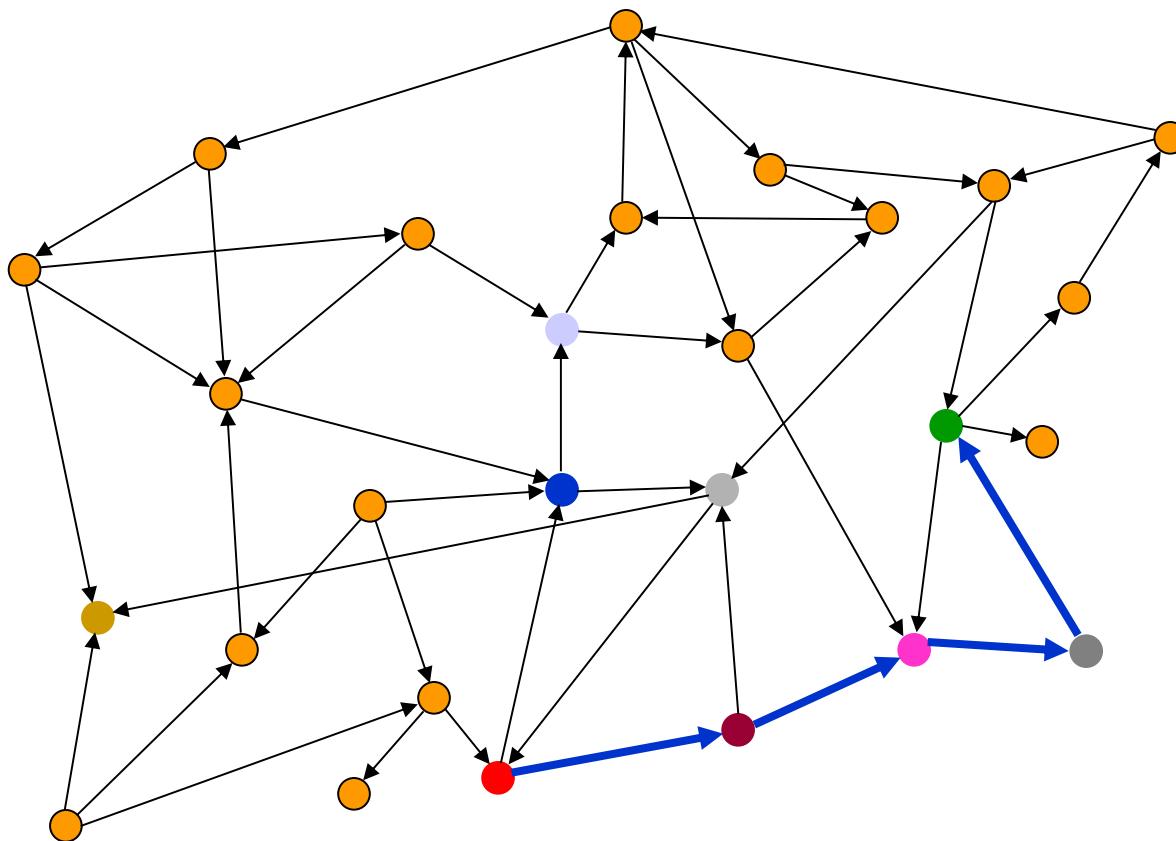


Arbre de recherche

# Recherche dans un espace d'états



# Recherche dans un espace d'états



# Algorithme général de recherche (#1)

1. Si  $\text{SOLUTION?}(\text{état-initial})$  alors retourner état-initial
2.  $\text{INSERER}(\text{noeud-initial}, \text{FILE})$
3. Répéter:
  - a. Si  $\text{vide}(\text{FILE})$  alors retourner **échec**
  - b.  $n \leftarrow \text{RETIRER}(\text{FILE})$  *Prolongement de n*
  - c.  $s \leftarrow \text{ETAT}(n)$
  - d. Pour chaque état  $s'$   $\text{SUCCESSEUR}(s)$ 
    - i. Créer un nouveau noeud  $n'$  comme "enfant" de  $n$
    - ii. Si  $\text{SOLUTION?}(s')$  alors retourner **chemin ou état-solution**
    - iii.  $\text{INSERER}(n', \text{FILE})$

Les méthodes de recherche diffèrent les unes des autres selon l'ordre dans lequel les noeuds sont insérés et extraits de la file d'attente.

# Définition formelle d'un problème de recherche

---

- Un problème de recherche est défini par les éléments suivants:
  - $Q$  un ensemble fini non-vide d'états
  - $S \subseteq Q$  un ensemble non-vide d'états initiaux
  - $G \subseteq Q$  un ensemble non-vide d'états-solutions (pouvant être définis explicitement ou implicitement par un test-solution)
  - $A$  un ensemble d'actions
  - Successeurs:  $Q \times A \rightarrow Q$
  - Fonction d'évaluation:  $Q \times A \times Q \rightarrow \mathbb{R}^+$   
(défini uniquement pour des états qui sont successeurs l'un de l'autre)

# Définir un problème de recherche

---

- Espace d'états
- État initial
- Fonction "successeur"
- Etat-solution
- Coût du chemin

# Définir un problème de recherche

---

- Espace d'états
  - chaque état est une représentation abstraite de l'environnement
  - l'espace d'état est discret, il peut être fini ou infini
- État initial
- Fonction "successeur"
- Etat-solution
- Coût du chemin

# Définir un problème de recherche

---

- Espace d'états
- État initial
  - habituellement l'état courant
  - parfois un ou plusieurs états hypothétiques
- Fonction "successeur"
- Etat-solution
- Coût du chemin

# Définir un problème de recherche

---

- Espace d'états
- État initial
- Fonction "successeur"
  - fonction : [ état → sous-ensemble d'états ]
  - une représentation abstraite des actions possibles
- Etat-solution
- Coût du chemin

# Définir un problème de recherche

---

- Espace d'états
- État initial
- Fonction "successeur"
- Etat-solution
  - habituellement une condition à satisfaire
  - parfois la description explicite d'un état
- Coût du chemin

# Définir un problème de recherche

---

- Espace d'états
- État initial
- Fonction "successeur"
- Etat-solution
- Coût du chemin
  - fonction : [ chemin → nombre positif ]
  - habituellement: coût du chemin = somme des coûts de ses étapes
  - Ex: # déplacements de la plaquette "vide"

## Fonction successeur

---

- Elle représente implicitement toutes les actions possibles dans chaque état
- Seuls les résultats des actions (les états successeurs) et leurs coûts sont retournés par la fonction
- La fonction successeur est une "boîte noire", son contenu est (le plus souvent) inconnu.  
Par ex., en planification d'assemblage, la fonction successeur peut être particulièrement complexe (collision, stabilité, prise, ...)

# Coût d'un chemin

- Le coût d'un arc est un nombre positif mesurant le "coût" de la réalisation de l'action représentée par l'arc, exemples:
  - 1 pour le puzzle-8
  - le temps nécessaire à assembler deux éléments d'un dispositif
- On suppose que pour n'importe quel problème le coût  $c$  d'un arc vérifie toujours la relation

$$c \geq \varepsilon > 0 \quad \text{où } \varepsilon \text{ est une constante}$$

[cette condition garantit que, si le chemin devient arbitrairement long, son coût devient aussi arbitrairement grand]

Pourquoi est-ce nécessaire?

## Etat solution

- Il peut être explicitement décrit

1	2	3
4	5	6
7	8	

- ou partiellement décrit

1	~	~
~	5	~
~	8	~

("~" signifie "quelconque" sauf 1, 5 et 8)

- ou défini par une condition  
ex: la somme de chaque ligne, chaque colonne et chaque diagonale vaut 30

15	1	2	12
4	10	9	7
8	6	5	11
3	13	14	

# Applications

- Les méthodes de recherche jouent un rôle-clé dans beaucoup d'applications ex.:
  - Recherche de chemin : lignes aériennes, réseaux téléphoniques et d'ordinateurs,
  - Distribution courrier, colis
  - Routage dans oléoducs, routage circuits VLSI,
  - Conception de médicaments,
  - Bioinformatique (classification et comparaison de structures de protéines)
  - Planification des déplacements d'un robot autonome,
  - Jeux vidéo.

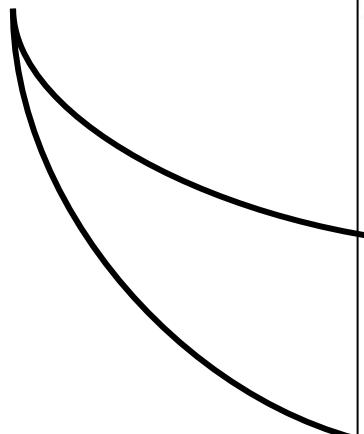
# Recherche heuristique: généralités

---

- Les algorithmes de recherche aveugle n'exploitent aucune information concernant la structure de l'arbre de recherche ou la présence potentielle de nœuds-solution pour optimiser la recherche.
- Recherche "rustique" à travers l'espace jusqu'à trouver une solution.
- La plupart des problèmes réels sont susceptibles de provoquer une explosion combinatoire du nombre d'états possibles.
- Un algorithme de recherche heuristique utilise toute l'information disponible pour rendre le processus de recherche plus efficace.
- Une information heuristique est une règle ou une méthode qui améliore presque toujours le processus de recherche.

# Rappel

L'ordre des éléments dans la file d'attente définit la stratégie de recherche



*Algorithme de recherche modifiée*

```

1. INSERER(noeud-initial,FILE)
2. Répéter:
   a. Si vide(FILE) alors retourner échec
   b.  $n \leftarrow RETIRER(FILE)$ 
   c.  $s \leftarrow ETAT(n)$ 
   ➤ d. Si SOLUTION?( $s$ ) alors retourner chemin ou état-solution
      e. Pour chaque état  $s'$  SUCESSEURS( $s$ )
         i. Créer un noeud  $n'$  comme successeur de  $n$ 
         ii. INSERER( $n'$ ,FILE)
  
```

14.05.2006 - © C. Pellegrin

# Fonction heuristique

- La fonction heuristique  $h(N) \geq 0$  estime la distance séparant ETAT( $N$ ) d'un état-solution
- Sa valeur est indépendante de l'arbre de recherche, elle ne dépend que de ETAT( $N$ ) et du test SOLUTION?
- Exemple

5		8
4	2	1
7	3	6

ETAT( $N$ )

1	2	3
4	5	6
7	8	

état solution

$$h_1(N) = \text{nombre de plaquettes mal placées} = 6$$

## Autres exemples

5		8
4	2	1
7	3	6

ETAT(N)

1	2	3
4	5	6
7	8	

état solution

$h_1(N) = \text{nombre de plaquettes mal placées} = 6$

$h_2(N) = \text{somme des distances (Manhattan) de chaque plaquette numérotée à sa position finale}$   
 $= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13$

## Observations

---

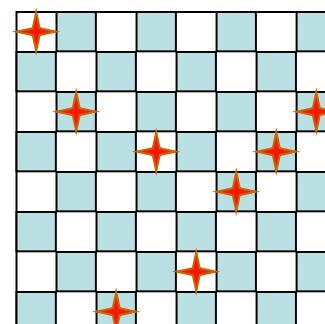
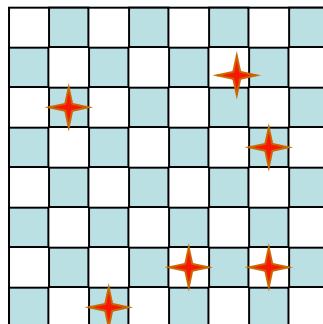
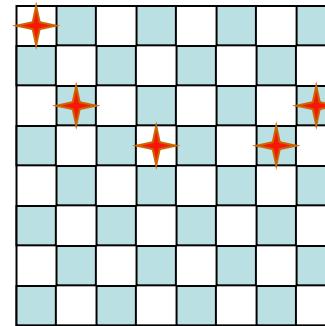
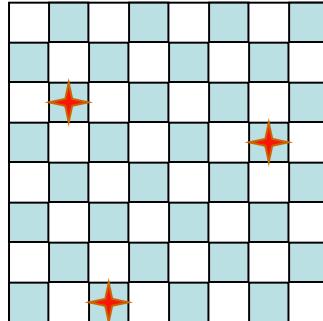
- Si l'espace d'états est infini, alors la recherche n'est en général pas complète (= n'aboutit pas à une solution)
- Si l'espace d'états est fini et si on n'écarte pas les nœuds qui revisitent des états déjà rencontrés, alors la recherche n'est en général pas complète
- Si l'espace d'états est fini et si on écarte les nœuds qui revisitent des états déjà rencontrés, alors la recherche est complète, mais en général elle ne sera pas optimale

# Problèmes à satisfaction de contraintes

- Exemple introductif
- Problèmes à satisfaction de contraintes (PSC)
- PSC comme problème de recherche
- Algorithme de "backtracking"
- Heuristiques générales

Certains transparents présentés dans ce chapitre s'inspirent de ceux du Prof. J-C. Latombe (Université de Stanford) et ont été adaptés à notre contexte, ceci avec l'autorisation de l'auteur.

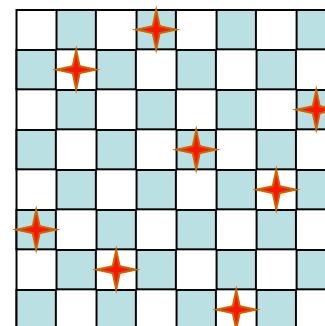
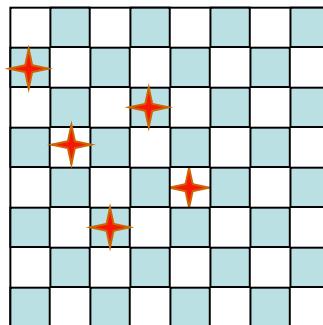
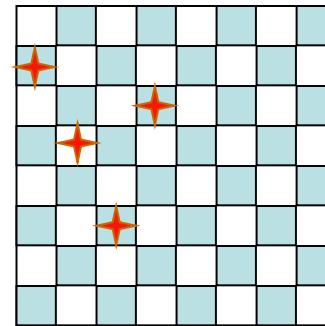
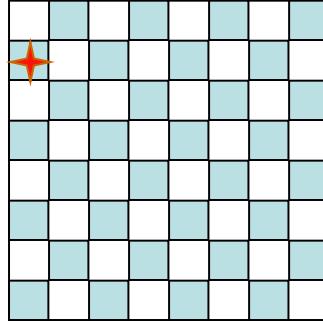
# Exemple introductif: 8-reines - 1<sup>ère</sup> formulation



- **Etats:** tous les arrangements de 0, 1, 2, ..., ou 8 reines sur le damier
- **État initial :** 0 reine sur le damier
- **Fonction successeur :** chacun des successeurs est obtenu en ajoutant une reine sur une case vide
- **Coût d'un arc:** sans importance
- **Test-solution:** 8 reines sur le damier, sans attaque mutuelle

→  $64 \times 63 \times \dots \times 53 \sim 3 \times 10^{14}$  états

# Exemple introductif: 8-reines - 2<sup>ème</sup> formulation



→ 2,057 états

- **Etats:** tous les arrangements de  $k = 0, 1, 2, \dots$ , ou 8 reines dans les  $k$  colonnes de gauche sans attaque mutuelle
- **Etat initial:** 0 reine sur le damier
- **Fonction successeur:** chaque successeur est obtenu en ajoutant une reine dans une case vide de la 1<sup>ère</sup> colonne de gauche disponible sans attaque mutuelle
- **Coût d'un arc:** sans importance
- **Test-solution:** 8 reines placées sur le damier

## *De quoi a-t-on besoin ?*

---

- Fonctions "successeur" et le test-solution ne suffisent plus
- Il faut aussi:
  - le moyen de **propager les contraintes** imposées par la position d'une reine sur les positions possibles des autres
  - un **test d'échec anticipé**

→ Représentation explicite des contraintes  
→ Algorithmes de propagation de contraintes

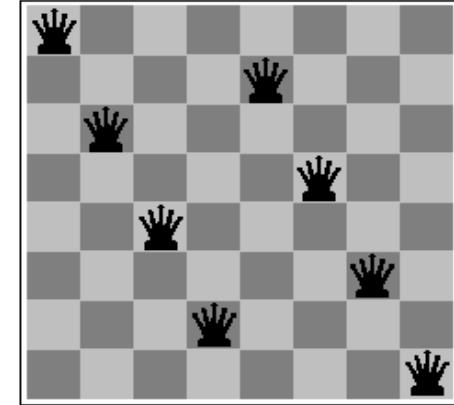
# Problème à satisfaction de contraintes

---

- Un problème à satisfaction de contraintes (PSC) est constitué:
  - d'un ensemble de variables  $\{X_1, X_2, \dots, X_n\}$ 
    - chaque variable  $X_i$  ayant un domaine  $D_i$  de valeurs possibles
      - en général  $D_i$  est discret et fini
  - d'un ensemble de contraintes  $\{C_1, C_2, \dots, C_p\}$ 
    - chaque contrainte  $C_k$  concerne un sous-ensemble de variables et spécifie les combinaisons de valeurs permises pour ces variables
- Objectif: assigner une valeur à chaque variable de sorte que toutes les contraintes soient satisfaites.
- Exemples: 8-reines, arithmétique cryptée, coloration de cartes, disposition des éléments sur un circuit VLSI, ordonnancement, ...

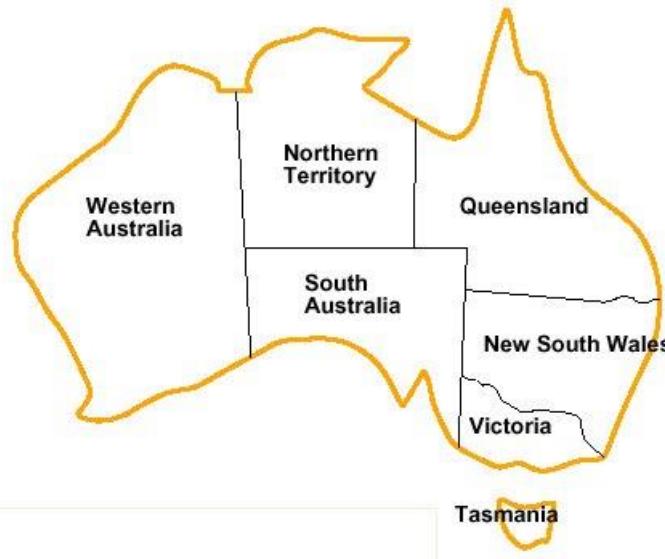
# 8-reines: formulation

- 64 variables  $X_{ij}$ ,  $i = 1 \text{ à } 8$  et  $j = 1 \text{ à } 8$
- Domaine de chaque variable: { 1, 0 }
- Contraintes de la forme:
  - $X_{ij} = 1 \Rightarrow X_{ik} = 0 \quad \forall k = 1 \text{ à } 8, k \neq j$
  - $X_{ij} = 1 \Rightarrow X_{kj} = 0 \quad \forall k = 1 \text{ à } 8, k \neq i$
  - Contraintes semblables pour les diagonales
  - $\sum_{i,j \in [1,8]} X_{ij} = 8$



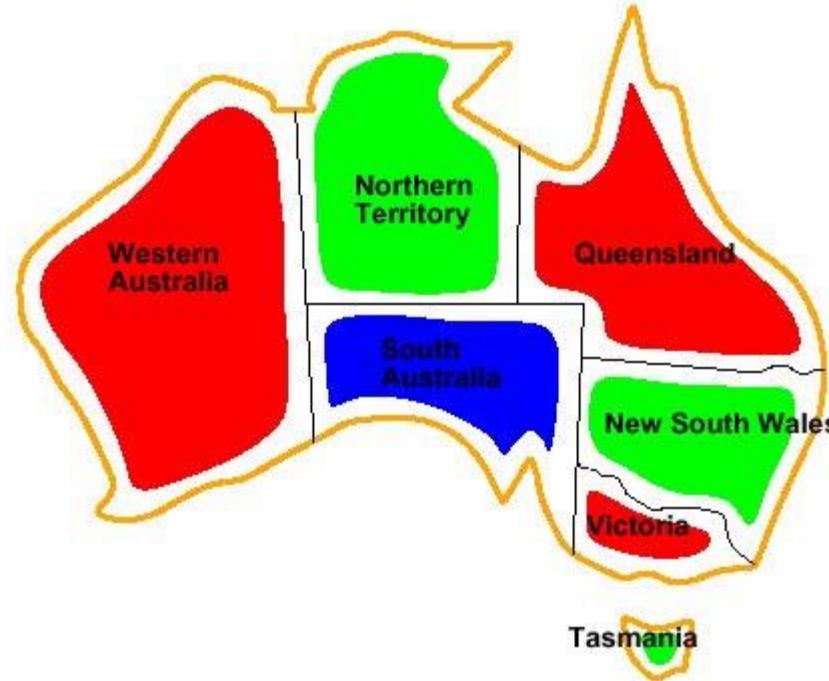
**Contraintes binaires**  
(chaque contrainte lie seulement 2 variables)

# Exemple: coloration de carte



- 7 variables:  $\{WA, NT, SA, Q, NSW, V, T\}$
- Chaque variable a le même domaine  $\{\text{rouge}, \text{vert}, \text{bleu}\}$
- Contraintes: 2 régions adjacentes doivent être de couleurs différentes
  - $WA \neq NT, WA \neq SA, NT \neq SA, NT \neq Q, SA \neq Q, SA \neq NSW, SA \neq V, Q \neq NSW, NSW \neq V$
  - ou:  $(WA, NT) \in \{(\text{rouge}, \text{vert}), (\text{rouge}, \text{bleu}), (\text{vert}, \text{rouge}), (\text{vert}, \text{bleu}) \dots\}$

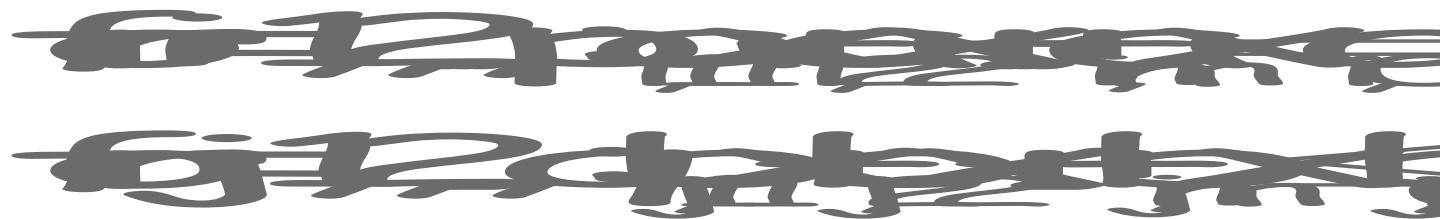
## Exemple: coloration de carte



- Les **solutions** sont des affectations satisfaisant toutes les contraintes, exemple:  
 $\{WA=\text{rouge}, NT=\text{vert}, Q=\text{rouge}, NSW=\text{vert}, V=\text{rouge}, SA=\text{bleu}, T=\text{vert}\}$

# *PSC fini et infini*

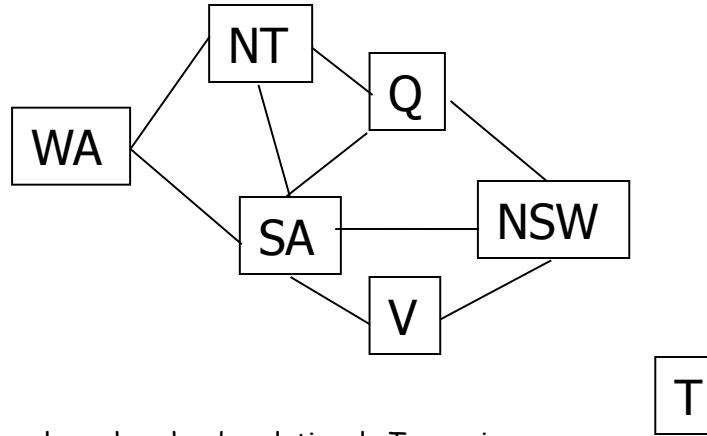
- Chaque variable a un domaine fini de valeurs → PSC fini
- Quelques (toutes) variables ont des domaines infinis → PSC infini  
(cas particulier: programmation linéaire)  
ex., problèmes de programmation linéaire sur des nombres réels:



- On ne traitera que des PSC finis

# Graphe de contraintes

- PSC binaire: chaque contrainte concerne au plus deux variables
- Graphe de contraintes
  - les nœuds représentent les variables
  - les arcs représentent les contraintes



Pour accélérer la recherche de solution, la Tasmanie sera considérée comme un sous-problème indépendant

- 2 variables sont adjacentes (ou voisines) si elles sont reliées par un arc
- les algorithmes généraux de PSC utilisent les structures de graphes

# Types de contraintes

---

- Unaire: ne concerne qu'une seule variable
  - exemple:  $SA \neq \text{vert}$
- Binaire: concerne des paires de variables
  - exemple:  $SA \neq WA$
- Ordre supérieur: concerne 3 variables ou plus
  - exemple: arithmétique cryptée
- Préférence (contrainte "molle"):
  - exemple: rouge est meilleur que vert
  - souvent représentée par un coût associé à chaque affectation possible

→ *problème d'optimisation de contraintes*

# Exemples de PSC réels

---

- Problèmes d'affectation
  - ex: qui enseigne quel cours?
- Problèmes d'horaires
  - ex: où et quand un enseignant donne-t-il ses cours?
- Configuration matérielle
- Organisation de transport (chemins de fer, compagnies aériennes)
- Ordonnancement de production (atelier)
- Occupation d'espace (architecture)
  
- Beaucoup de problèmes du monde réel impliquent des variables à valeurs réelles (continues)

# PSC comme un problème de recherche

- n variables  $X_1, \dots, X_n$
- Affectation valide :  
 $\{X_1 \leftarrow v_1, \dots, X_k \leftarrow v_k\}, \quad 0 \leq k \leq n,$   
telles que les valeurs  $v_1, \dots, v_k$  satisfassent toutes les contraintes liant les variables  $X_1, \dots, X_k$
- Affectation complète: une pour qui  $k = n$   
[Si tous les domaines de variables sont de taille d, il y a  $O(d^n)$  affectations complètes]
- Etats: affectations valides
- Etat initial: affectation vide  $\{ \}$ , càd  $k = 0$
- Successeur d'un état:  
 $\{X_1 \leftarrow v_1, \dots, X_k \leftarrow v_k\} \rightarrow \{X_1 \leftarrow v_1, \dots, X_k \leftarrow v_k, X_{k+1} \leftarrow v_{k+1}\}$
- Test-solution:  $k = n$

- 4 variables  $X_1, \dots, X_4$
- Soit l'assignement valable de N:  
$$A = \{X_1 \leftarrow v_1, X_3 \leftarrow v_3\}$$
- (par exemple) choisir la variable  $X_4$
- Soit  $\{v_{4,1}, v_{4,2}, v_{4,3}\}$  le domaine de  $X_4$
- Les successeurs de A sont tous les assignements valables parmi:

$$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3, X_4 \leftarrow v_{4,1}\}$$

$$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3, X_4 \leftarrow v_{4,2}\}$$

$$\{X_1 \leftarrow v_1, X_3 \leftarrow v_3, X_4 \leftarrow v_{4,3}\}$$

## Propriété des PSC: commutativité

- L'ordre dans lequel les valeurs sont assignées aux variables est sans importance pour la solution finale
    - [WA=rouge suivi de NT=vert] identique à [NT=vert suivi de WA=rouge]
  - Donc:
    1. On peut produire les successeurs d'un nœud N en sélectionnant une variable X absente de l'assignement A associé à N et en assignant chaque valeur v du domaine de X  
[→ importante réduction du facteur de branchement]
    2. Il n'est pas nécessaire de mémoriser le chemin menant à un nœud donné
- Algorithme de recherche par "Backtracking"

# Recherche « Backtracking »

---

- La recherche en profondeur pour des PSC avec affectation d'une seule variable à la fois est appelée recherche "backtracking"
  - c'est l'algorithme non heuristique de base pour les PSC
  - capable de résoudre le problème des n-reines pour  $n \approx 25$
- C'est essentiellement une version simplifiée de l'algorithme de recherche en profondeur utilisant la récursivité

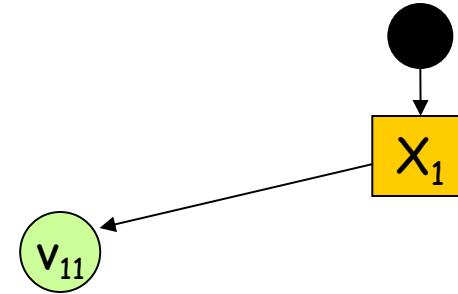
# Recherche « backtracking » (3 variables)

---



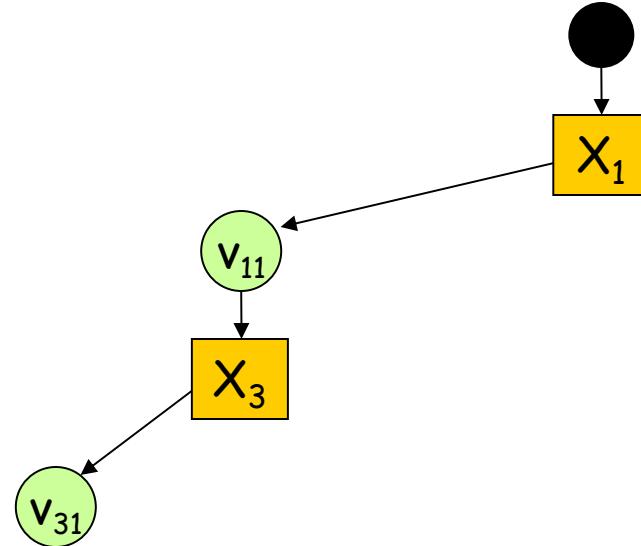
Assignment = {}

# Recherche « backtracking » (3 variables)



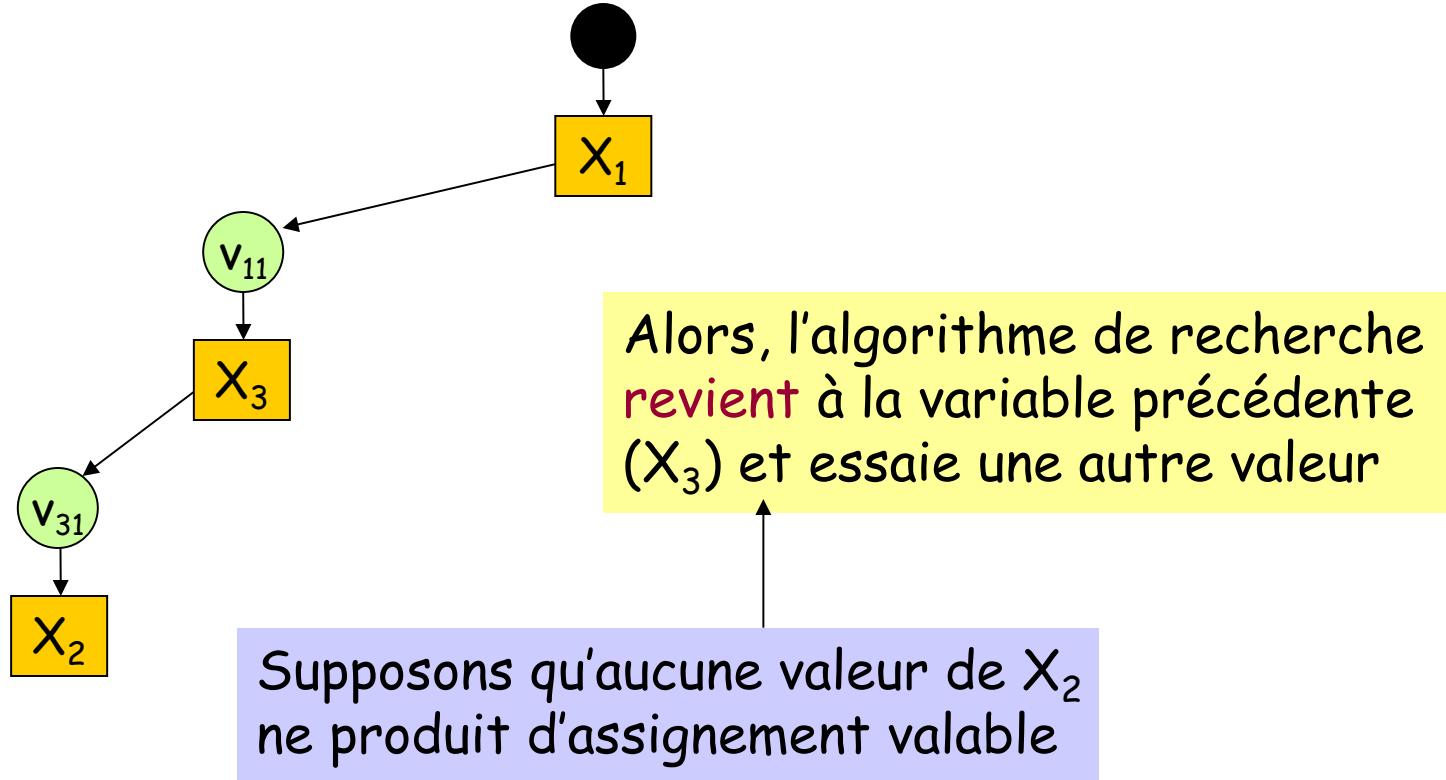
Assignment =  $\{(X_1, v_{11})\}$

# Recherche « backtracking » (3 variables)



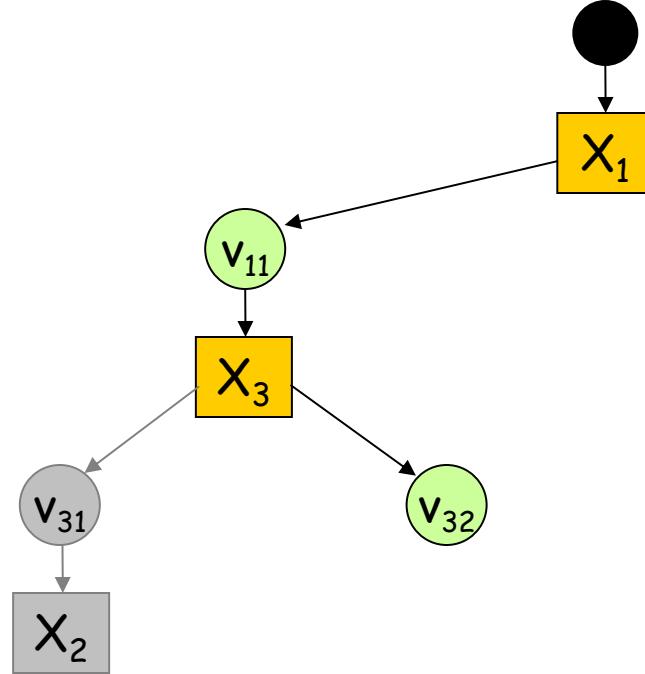
Assignment =  $\{(X_1, v_{11}), (X_3, v_{31})\}$

# Recherche « backtracking » (3 variables)



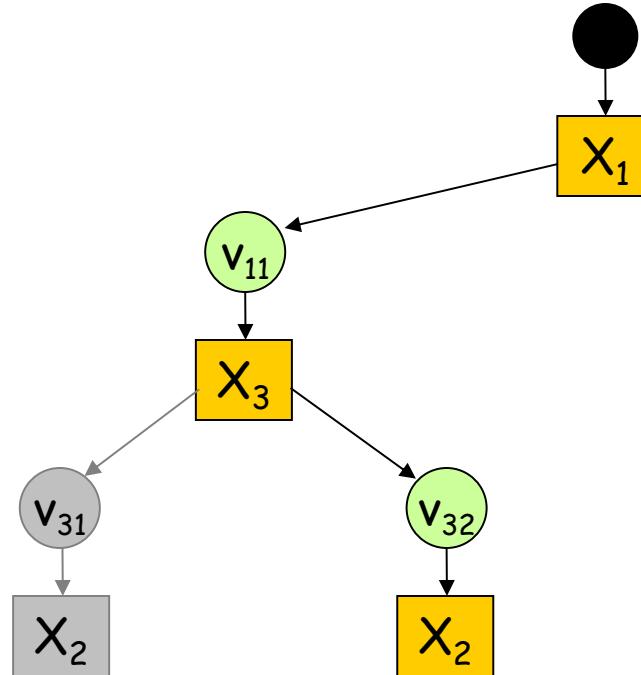
Assignment =  $\{(X_1, v_{11}), (X_3, v_{31})\}$

# Recherche « backtracking » (3 variables)



Assignment =  $\{(X_1, v_{11}), (X_3, v_{32})\}$

# Recherche « backtracking » (3 variables)

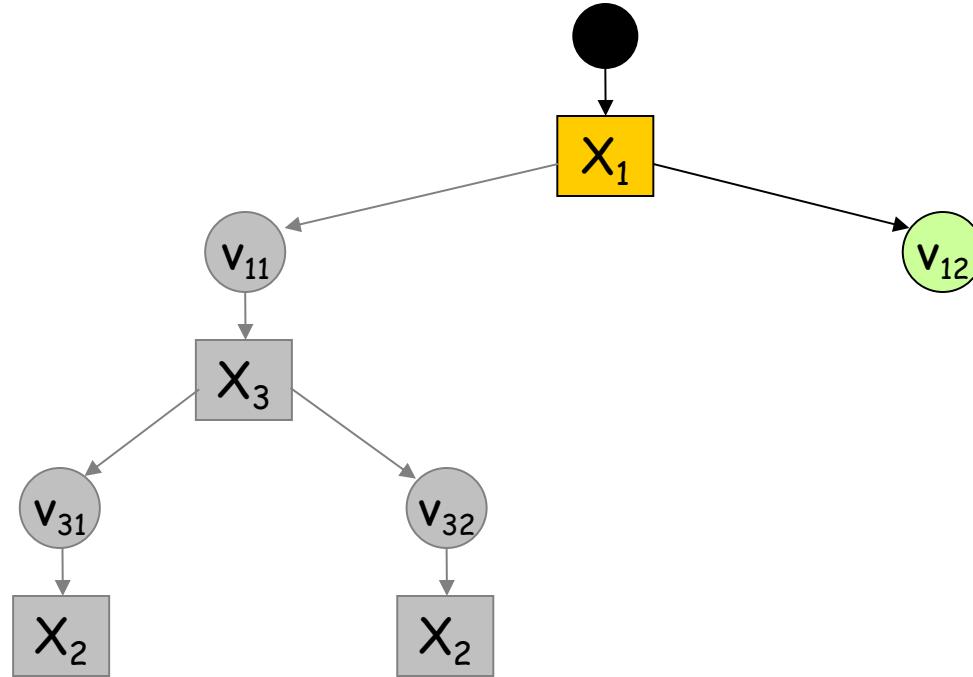


L'algorithme de recherche revient à la variable précédente ( $X_3$ ) et essaie une autre valeur. Mais supposons que  $X_3$  n'a que deux valeurs possibles. Alors l'algorithme revient à  $X_1$

Supposons à nouveau qu'aucune valeur de  $X_2$  ne produit d'assignement valable

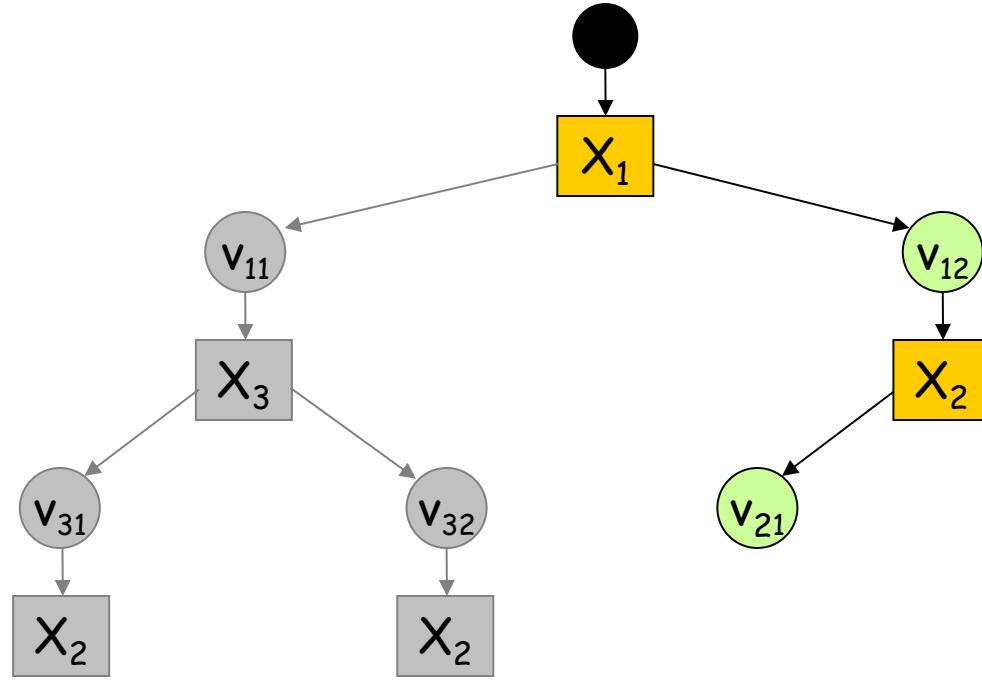
$$\text{Assignment} = \{(X_1, v_{11}), (X_3, v_{32})\}$$

# Recherche « backtracking » (3 variables)



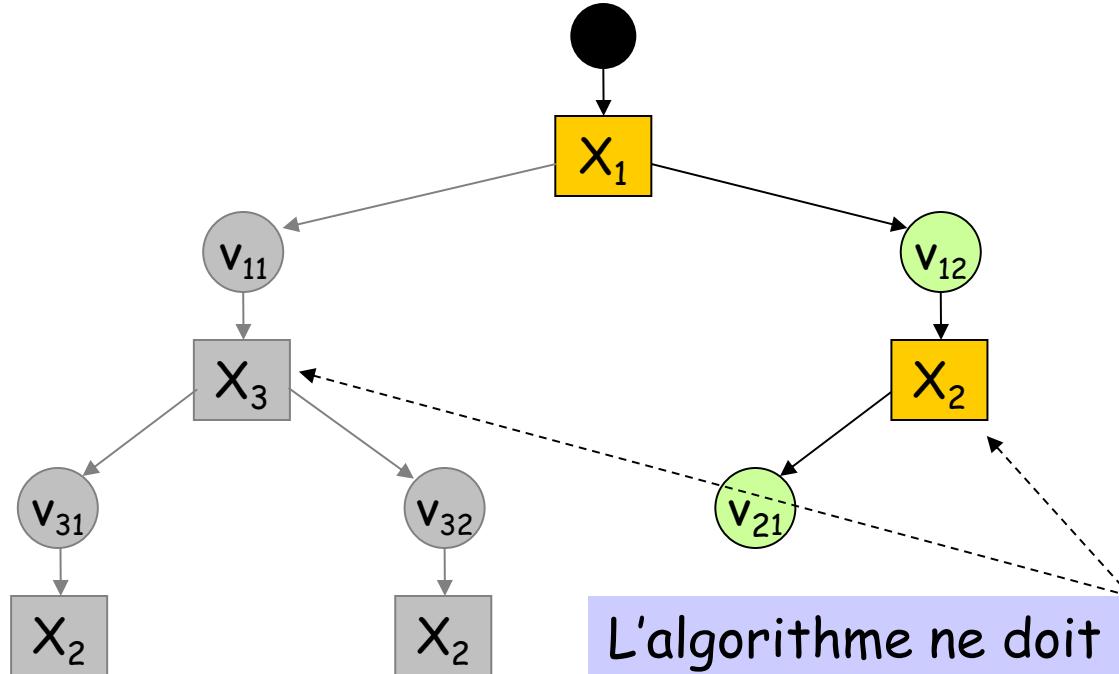
Assignment =  $\{(X_1, v_{12})\}$

# Recherche « backtracking » (3 variables)



Assignment =  $\{(X_1, v_{12}), (X_2, v_{21})\}$

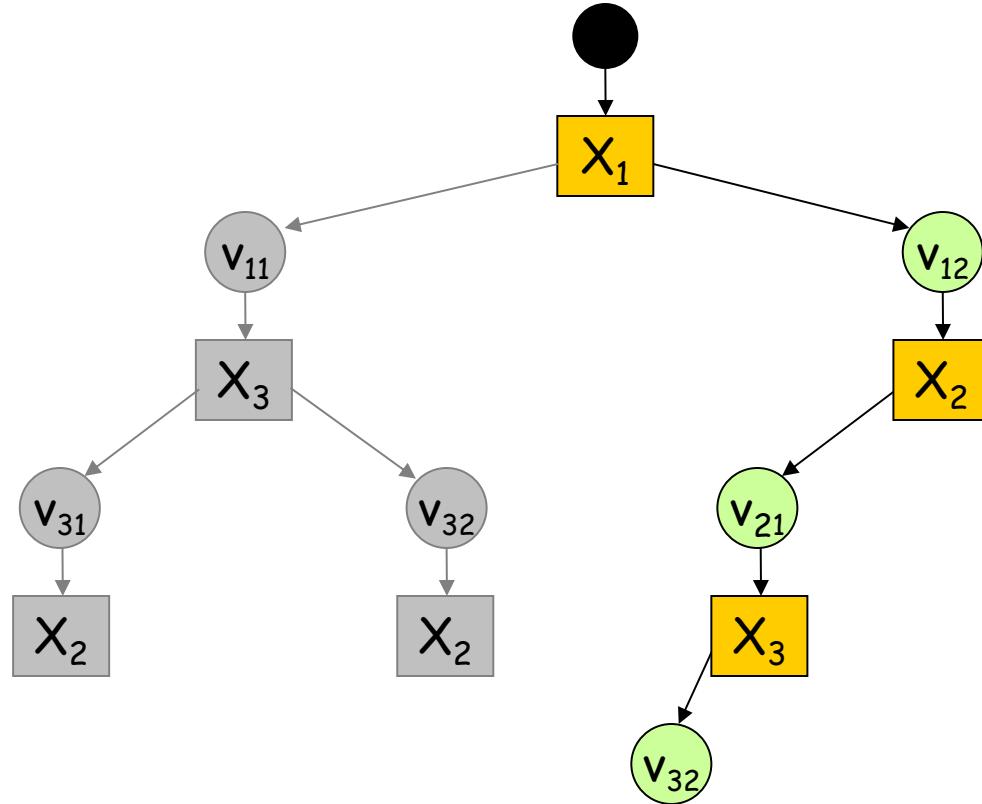
# Recherche « backtracking » (3 variables)



L'algorithme ne doit pas considérer les variables dans le même ordre dans ce sous-arbre et dans l'autre

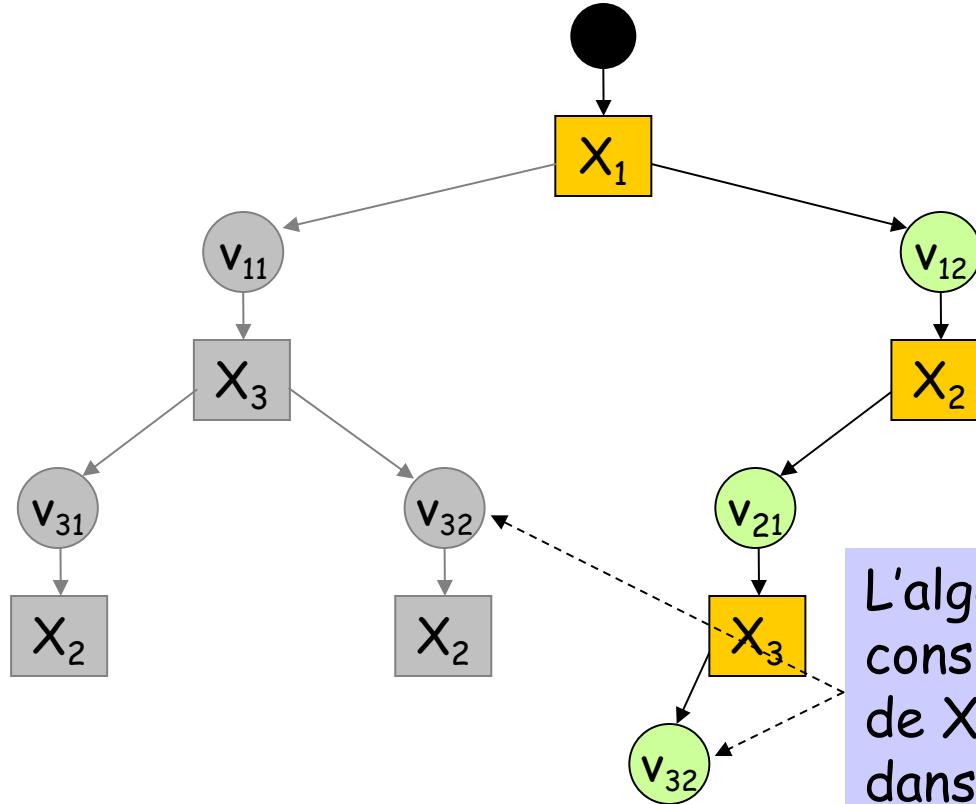
Assignment =  $\{(X_1, v_{12}), (X_2, v_{21})\}$

# Recherche « backtracking » (3 variables)



Assignment =  $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

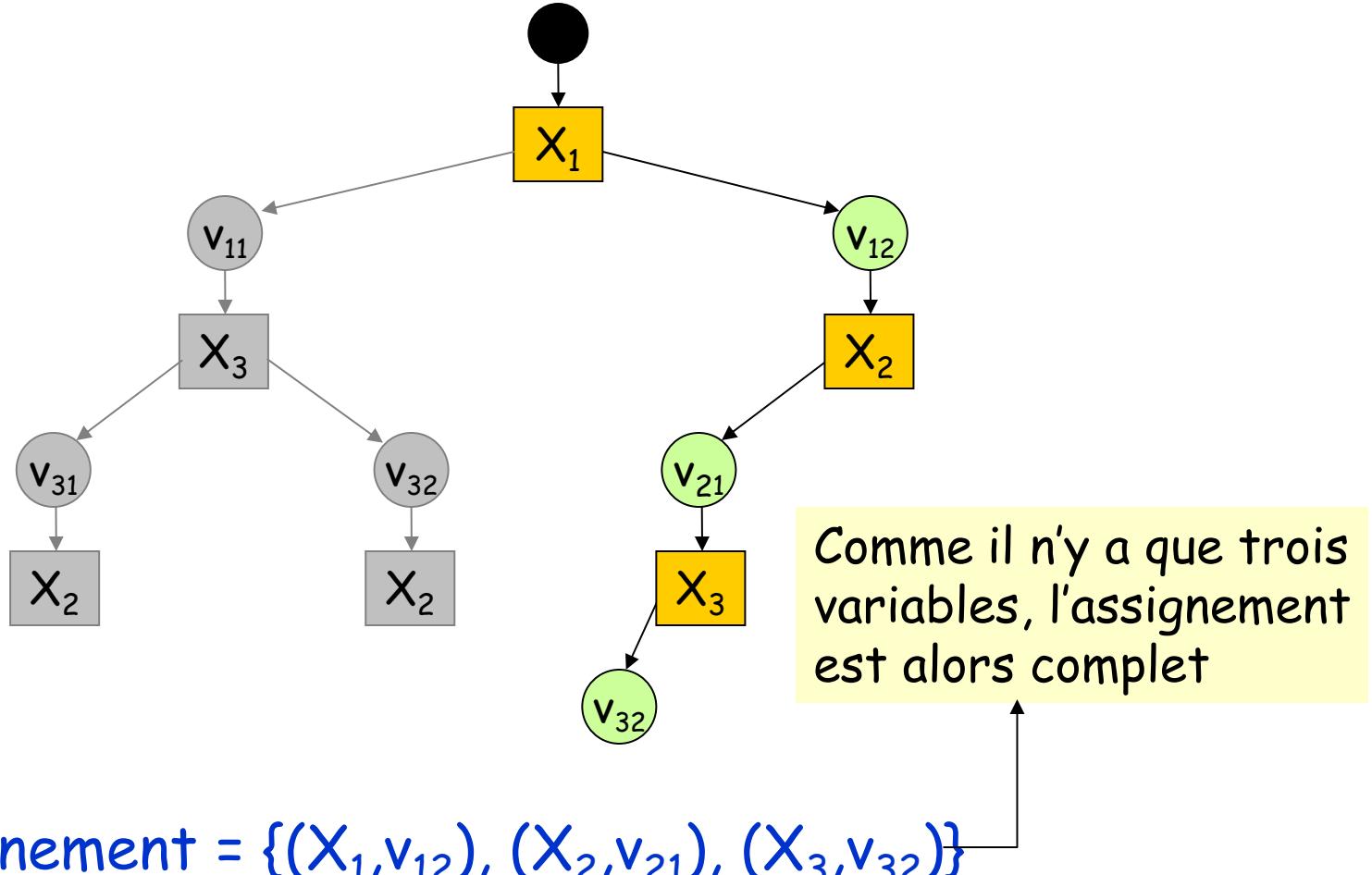
# Recherche « backtracking » (3 variables)



L'algorithme ne doit pas considérer les valeurs de  $X_3$  dans le même ordre dans ce sous-arbre

Assignment =  $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

# Recherche « backtracking » (3 variables)



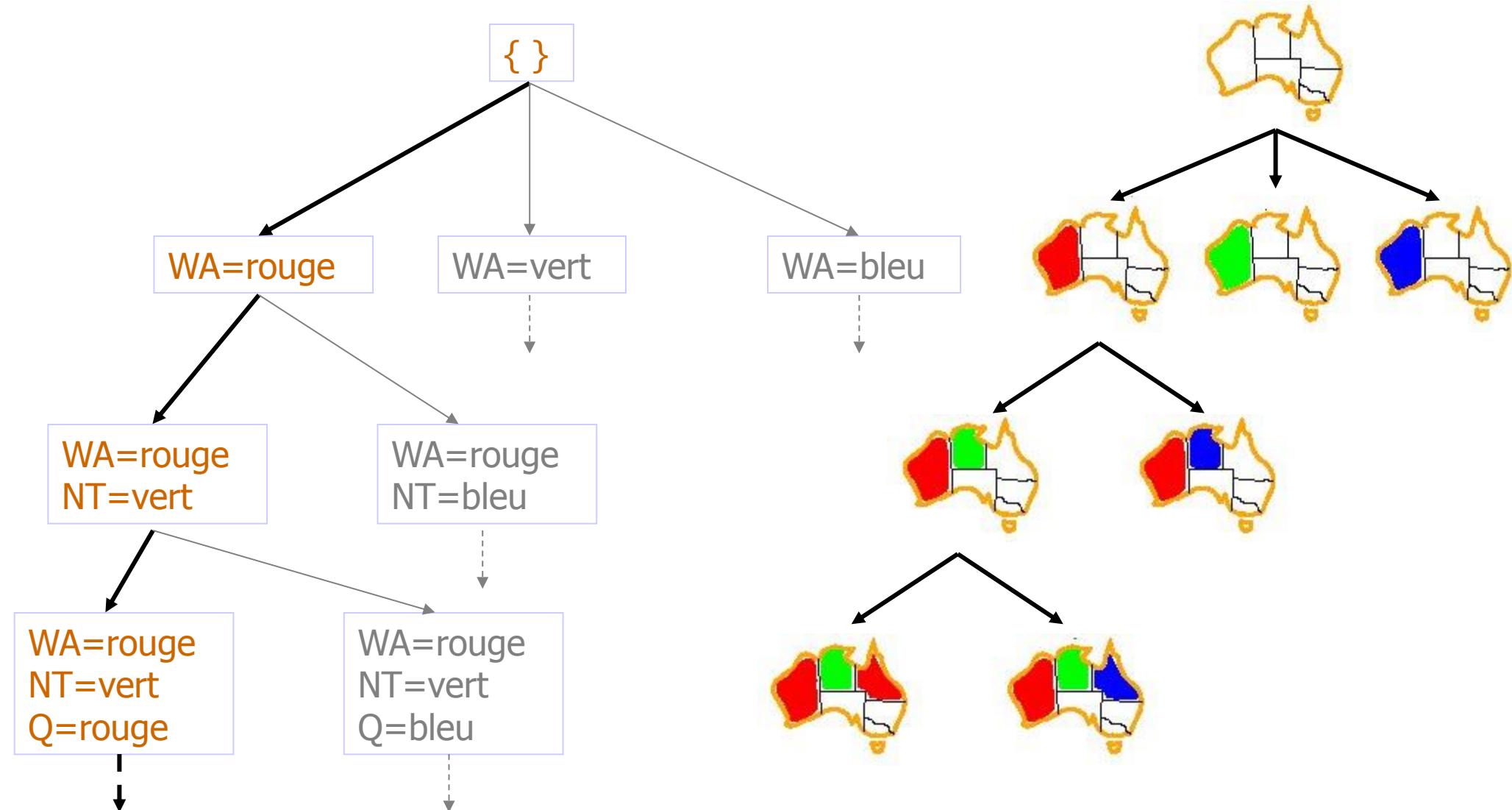
# Algorithme de « backtracking »

## PSC-BACKTRACKING( $A$ )

1. Si assignement  $A$  est complet alors retourner  $A$
2.  $X \leftarrow$  sélectionner une variable absente de  $A$
3.  $D \leftarrow$  sélectionner un ordre sur le domaine de  $X$
4. Pour chaque valeur  $v$  dans  $D$  faire
  - a. Add ( $X \leftarrow v$ ) à  $A$
  - b. Si  $A$  est valide alors
    - i.  $résultat \leftarrow$  PSC-BACKTRACKING( $A$ )
    - ii. Si  $résultat \neq$  échec alors retourner  $résultat$
5. Retourner échec

Appel: PSC-BACKTRACKING({})

## Exemple: coloration de carte



# Questions

## PSC-BACKTRACKING( $A$ )

1. Si assignement  $A$  est complet alors retourner  $A$
2.  $X \leftarrow$  sélectionner une variable absente de  $A$
3.  $D \leftarrow$  sélectionner un ordre sur le domaine de  $X$
4. Pour chaque valeur  $v$  dans  $D$  faire
  - a. Add  $(X \leftarrow v)$  à  $A$
  - b. Si  $A$  est valide alors
    - i. résultat  $\leftarrow$  PSC-BACKTRACKING( $A$ )
    - ii. Si résultat  $\neq$  échec alors retourner résultat
5. Retourner échec

## Questions

---

- 1) Quelle prochaine variable X doit recevoir une valeur ?
  
- 1) Dans quel ordre les valeurs de X doivent-elles être assignées?

## Questions

---

1) Quelle prochaine variable  $X$  doit recevoir une valeur ?

L'assignement courant peut ne pas mener à une quelconque solution, mais l'algorithme ne le sait pas encore. Sélectionner la bonne variable  $X$  peut aider à trouver la contradiction plus rapidement

1) Dans quel ordre les valeurs de  $X$  doivent-elles être assignées?

## Questions

---

1) Quelle prochaine variable X doit recevoir une valeur ?

L'assignement courant peut ne pas mener à une quelconque solution, mais l'algorithme ne le sait pas encore. Sélectionner la bonne variable X peut aider à trouver la contradiction plus rapidement

1) Dans quel ordre les valeurs de X doivent-elles être assignées?

L'assignement peut faire partie de la solution.  
Sélectionner la bonne valeur à assigner à X peut aider à trouver la solution plus rapidement

## Questions

---

1) Quelle prochaine variable X doit recevoir une valeur ?

L'assignement courant peut ne pas mener à une quelconque solution, mais l'algorithme ne le sait pas encore. Sélectionner la bonne variable à assigner peut aider à trouver la contradiction plus rapidement

1) Dans quel ordre les valeurs de X doivent-elles être assignées?

L'assignement peut faire partie de la solution.  
Sélectionner la bonne valeur à assigner à X peut aider à trouver la solution plus rapidement

Plus sur ces questions prochainement ...

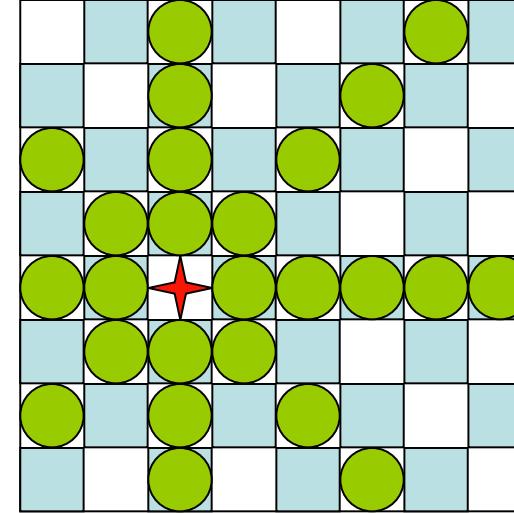
# Forward checking

Une technique simple de propagation de contraintes:

	X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	X <sub>4</sub>	X <sub>5</sub>	X <sub>6</sub>	X <sub>7</sub>	X <sub>8</sub>
1								
2								
3								
4								
5	*							
6								
7								
8								

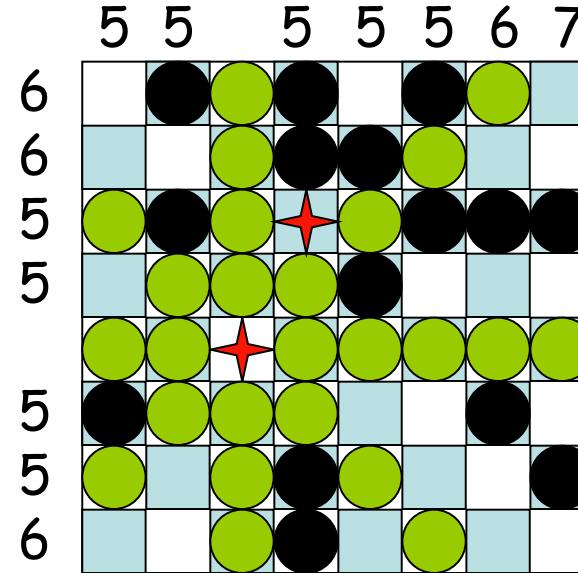
Assigner la valeur 5 à  $X_1$  implique éliminer des valeurs des domaines de  $X_2, X_3, \dots, X_8$

# Forward checking pour les 8 reines



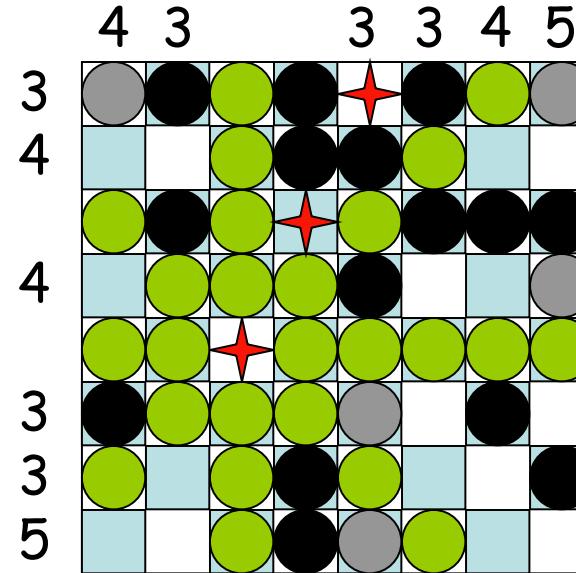
- Placer une reine dans une case
- Éliminer les cases attaquées pour de futures considérations

# Forward checking pour les 8 reines



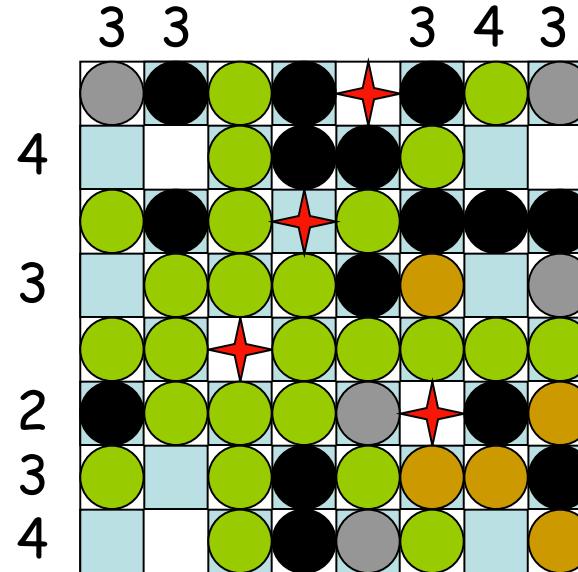
- Compter le nombre de cases libres d'attaques dans chaque ligne et colonne
- Placer une reine dans une ligne ou une colonne ayant le plus petit nombre
- Éliminer les cases attaquées pour de futures considérations

# Forward checking pour les 8 reines

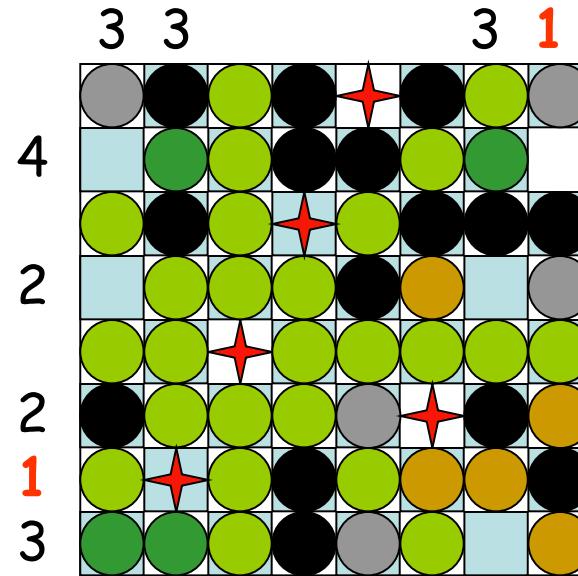


- Répéter

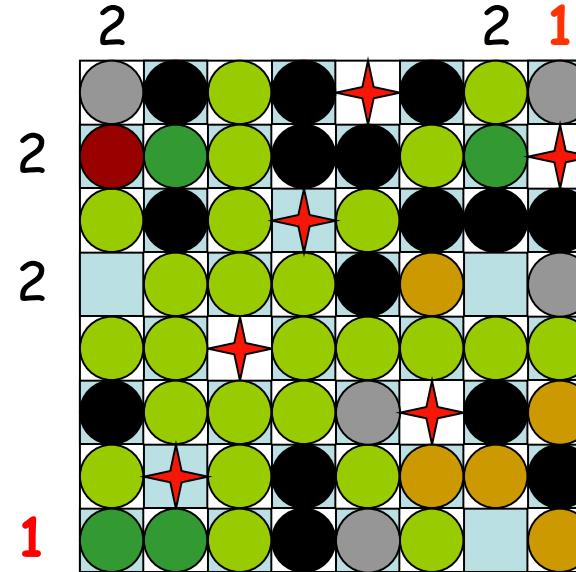
# Forward checking pour les 8 reines



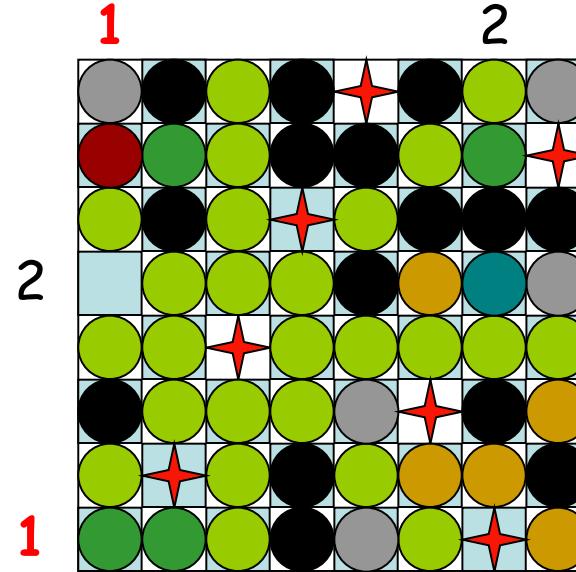
# Forward checking pour les 8 reines



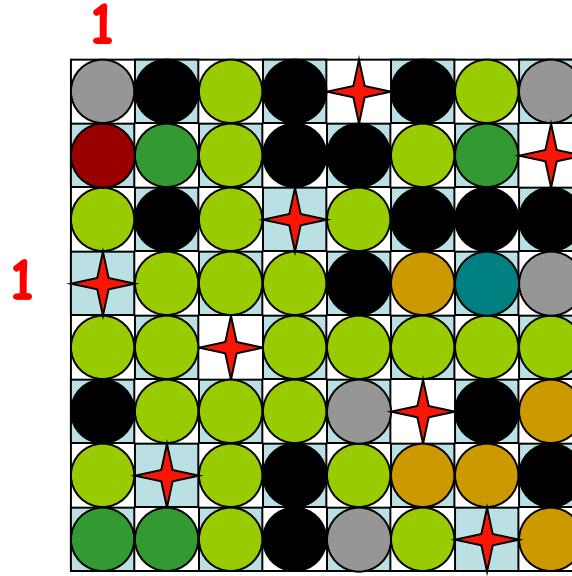
# Forward checking pour les 8 reines



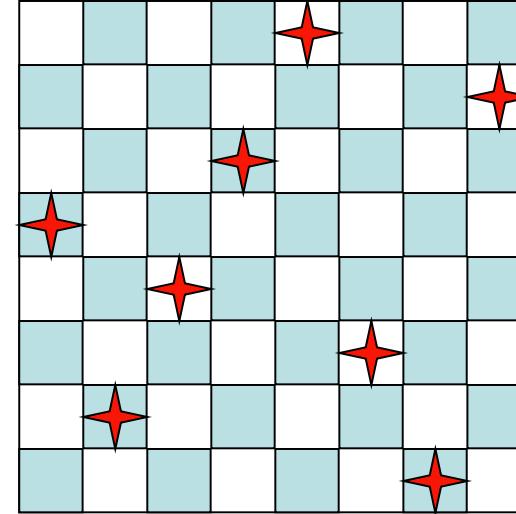
# Forward checking pour les 8 reines



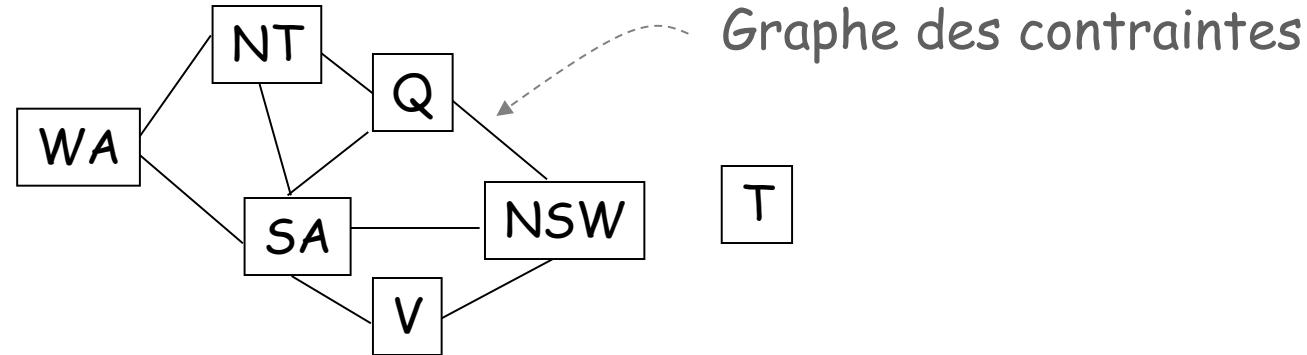
# Forward checking pour les 8 reines



# Forward checking pour les 8 reines



# Forward checking - coloration de cartes



WA

RGB

NT

RGB

Q

RGB

NSW

RGB

V

RGB

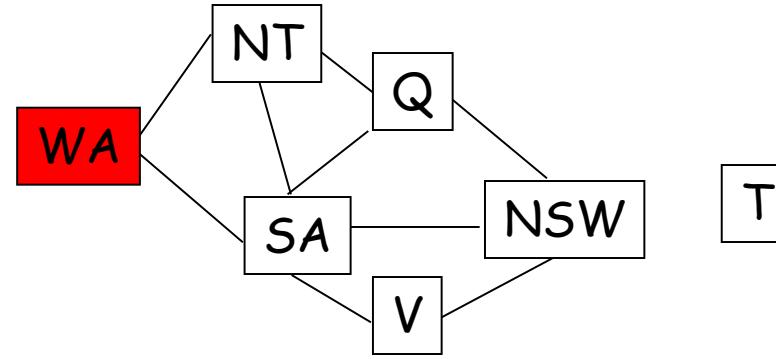
SA

RGB

T

RGB

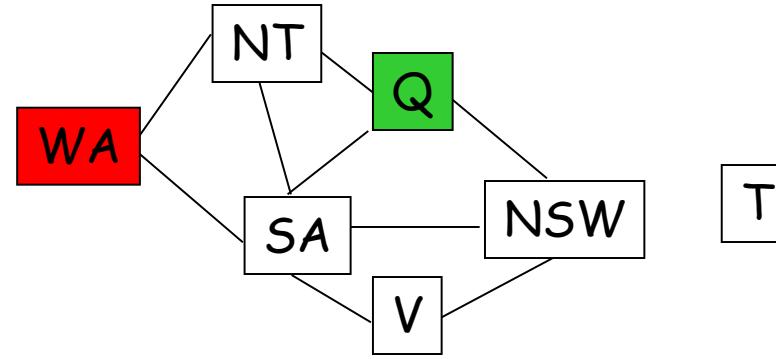
# Forward checking - coloration de cartes



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	<del>RGB</del>	RGB	RGB	RGB	<del>RGB</del>	RGB

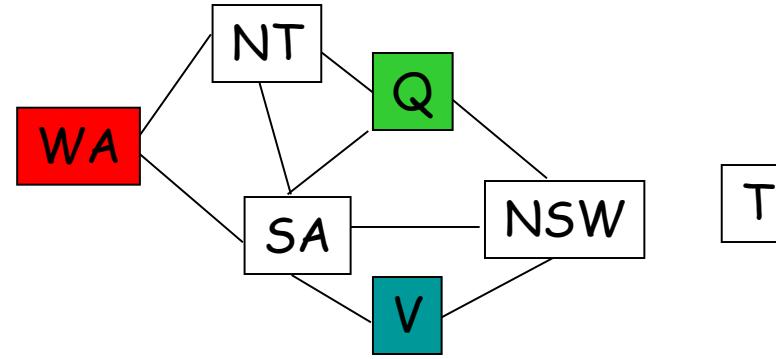
Forward checking élimine la valeur Rouge pour NT et pour SA

# Forward checking - coloration de cartes



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	<del>GB</del>	<del>G</del>	<del>RB</del>	RGB	<del>GB</del>	RGB

# Forward checking - coloration de cartes



WA	NT	Q	NSW	V	SA	T
RGB						
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	RB	B	B	RGB

# Forward checking - coloration de cartes

Ensemble vide: l'assignement courant  
 $\{(WA \leftarrow R), (Q \leftarrow G), (V \leftarrow B)\}$   
ne mène pas à la solution

WA	NT	Q	NSW	V	SA	T
RGB						
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	RB	B	B	RGB

## Forward checking - forme générale

---

Chaque fois qu'une paire  $(X \leftarrow v)$  est ajoutée à un assignement A faire:

Pour chaque variable Y absente de A faire:

Pour chaque contrainte C liant Y aux variables de A faire:

Eliminer toutes les valeurs des domaines de Y qui ne satisfont pas C

# Algorithme de backtracking modifié

## PSC-BACKTRACKING( $A$ , var-domaines)

1. Si assignement  $A$  est complet alors retourner  $A$
2.  $X \leftarrow$  sélectionner une variable absente de  $A$
3.  $D \leftarrow$  sélectionner un ordre sur le domaine de  $X$
4. Pour chaque valeur  $v$  dans  $D$  faire
  - a. Add  $(X \leftarrow v)$  à  $A$
  - b. var-domaines  $\leftarrow$  forward checking(var-domaines,  $X$ ,  $v$ ,  $A$ )
  - c. Si aucune variable a un domaine vide alors
    - i. résultat  $\leftarrow$  PSC-BACKTRACKING( $A$ , var-domaines)
    - ii. Si résultat  $\neq$  échec alors retourner résultat
  - d. Enlever  $(X \leftarrow v)$  de  $A$
5. Retourner échec

# Algorithme de backtracking modifié

## PSC-BACKTRACKING( $A$ , var-domaines)

1. Si assignement  $A$  est complet alors retourner  $A$
2.  $X \leftarrow$  sélectionner une variable absente de  $A$
3.  $D \leftarrow$  sélectionner un ordre sur le domaine de  $X$
4. Pour chaque valeur  $v$  dans  $D$  faire
  - a. Add  $(X \leftarrow v)$  à  $A$  -----  $\rightarrow$  Plus besoin de vérifier que  $A$  est valide
  - b. var-domaines  $\leftarrow$  forward checking(var-domaines,  $X$ ,  $v$ ,  $A$ )
  - c. Si aucune variable a un domaine vide alors
    - i. résultat  $\leftarrow$  PSC-BACKTRACKING( $A$ , var-domaines)
    - ii. Si résultat  $\neq$  échec alors retourner résultat
  - d. Enlever  $(X \leftarrow v)$  de  $A$
5. Retourner échec

# Algorithme de backtracking modifié

PSC-BACKTRACKING( $A$ , var-domaines)

1. Si assignement  $A$  est complet alors retourner  $A$
2.  $X \leftarrow$  sélectionner une variable absente de  $A$
3.  $D \leftarrow$  sélectionner un ordre sur le domaine de  $X$
4. Pour chaque valeur  $v$  dans  $D$  faire
  - a. Add  $(X \leftarrow v)$  à  $A$
  - b. var-domaines  $\leftarrow$  forward checking(var-domaines,  $X$ ,  $v$ ,  $A$ )
  - c. Si aucune variable a un domaine vide alors
    - i. résultat  $\leftarrow$  PSC-BACKTRACKING( $A$ , var-domaines)
    - ii. Si résultat  $\neq$  échec alors retourner résultat
  - d. Enlever  $(X \leftarrow v)$  de  $A$
5. Retourner échec

Besoin de transmettre les domaines de variables modifiés

## Algorithme de backtracking modifié

### PSC-BACKTRACKING( $A$ , var-domaines)

1. Si assignement  $A$  est complet alors retourner  $A$
2.  $X \leftarrow$  sélectionner une variable absente de  $A$
3.  $D \leftarrow$  sélectionner un ordre sur le domaine de  $X$
4. Pour chaque valeur  $v$  dans  $D$  faire
  - a. Add  $(X \leftarrow v)$  à  $A$
  - b. var-domaines  $\leftarrow$  forward checking(var-domaines,  $X$ ,  $v$ ,  $A$ )
  - c. Si aucune variable a un domaine vide alors
    - i. résultat  $\leftarrow$  PSC-BACKTRACKING( $A$ , var-domaines)
    - ii. Si résultat  $\neq$  échec alors retourner résultat
  - d. Enlever  $(X \leftarrow v)$  de  $A$
5. Retourner échec

1) Quelle variable suivant  $X_i$  devrait recevoir une valeur ?

→ heuristique de la variable la plus contrainte

→ heuristique de la variable la plus contraignante

1) Dans quel ordre ses valeurs doivent-elles être assignées?

→ heuristique de la valeur la moins contraignante

Ces heuristiques peuvent être déroutantes

Mais garder à l'esprit que **toutes** les variables finiront par recevoir une valeur, alors que **une** seule valeur d'un domaine doit être assignée à chaque variable

## *Heuristique de la variable la plus contrainte*

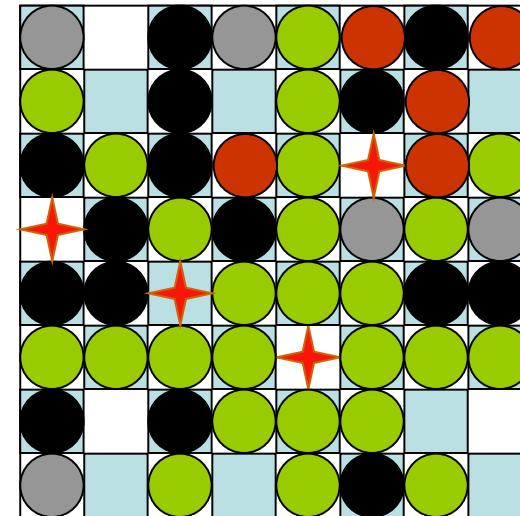
---

1) Quelle variable suivant  $X_i$  devrait recevoir une valeur ?

sélectionner la variable ayant le plus petit domaine restant

[Objectif: minimiser le facteur de branchement]

# 8-reines



Forward checking

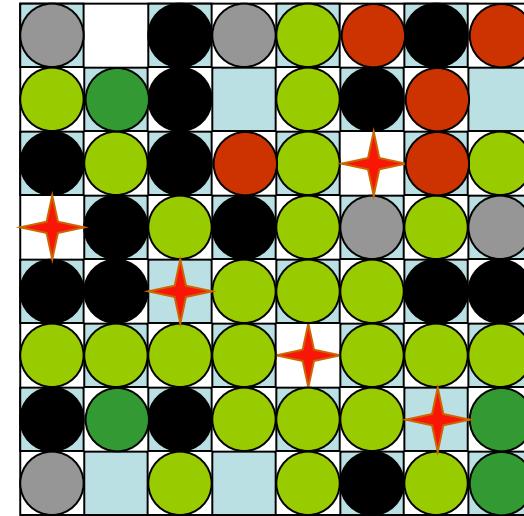
Nouvel assignement

4 3 2 3 4



Nombres de  
valeurs pour  
chaque variable  
non-assignée

## 8-reines



Forward checking

Nouvel assignement

4

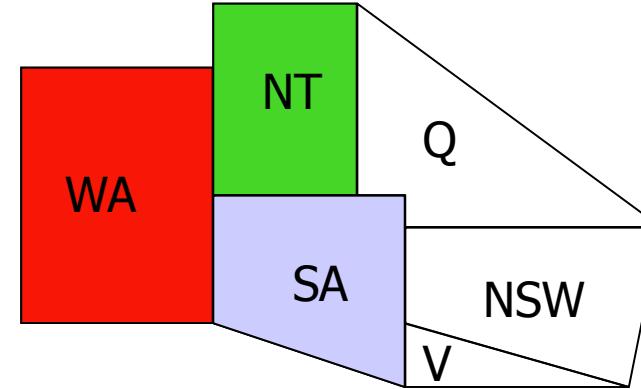
2

1

3

Nombres de  
valeurs pour  
chaque variable  
non-assignée

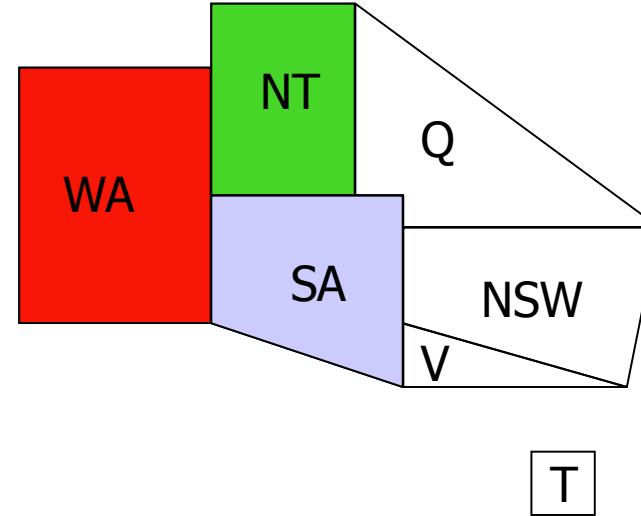
# Coloration de cartes



Problèmes analogues :

- Affecter des fréquences différentes à des cellules voisines dans un réseau de téléphone mobile GSM
- Problème d'incompatibilité. Comment faire cohabiter des personnes ou des animaux en tenant compte de leur incompatibilité ?
- La résolution du Sudoku peut se ramener à un problème de coloration de graphe

# Heuristique de la variable la plus contrainte



- Taille du domaine restant de  $SA = 1$  (valeur Bleu)
  - Taille du domaine restant de  $Q = 2$
  - Tailles des domaines de  $NSW, V$  et  $T = 3$
- Sélectionner  $SA$

## *Heuristique de la variable la plus contraignante*

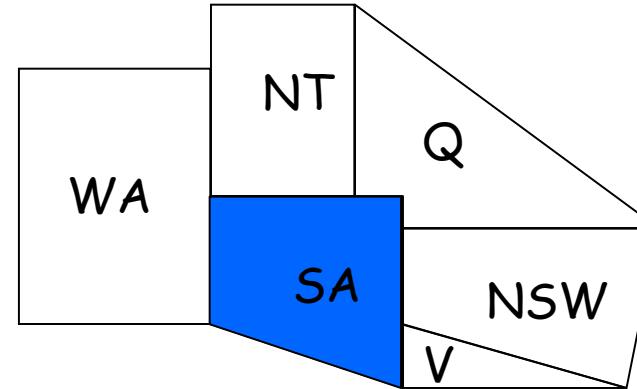
---

1) Quelle variable suivant  $X_i$  devrait recevoir une valeur ?

Parmi les variables ayant les domaines le plus petits, choisir celle qui apparait dans le plus grand nombre de contraintes sur des variables non encore assignées

[Objectif: augmenter le nombre d'éliminations futures de valeurs pour réduire le facteur de branchement]

# Coloration de cartes



- Avant toute assignation de valeurs, toutes les variables ont des domaines de taille 3, mais SA est impliquée dans plus de contraintes (5) que n'importe quelle autre variable  
→ Sélectionner SA et lui assigner une valeur (e.g., Bleu)

## Heuristique de la valeur la moins contraignante

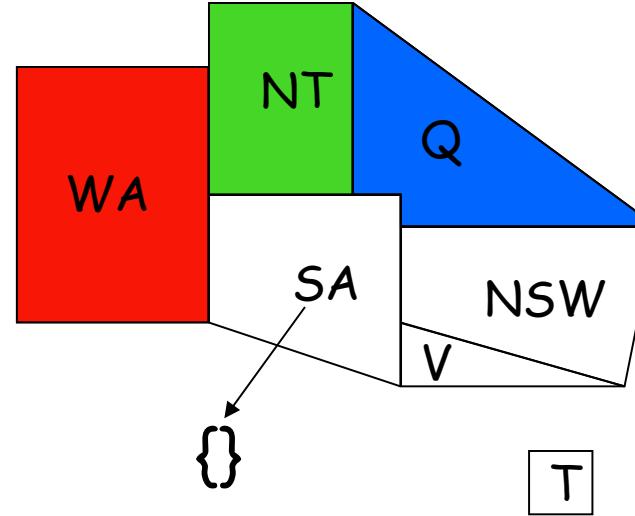
- 1) Dans quel ordre les valeurs de X doivent-elles être assignées?

Sélectionner la valeur de X qui élimine le plus petit nombre de valeurs des domaines des variables non encore assignées

[Argument: comme une seule valeur doit être assignée à X, choisir en premier la moins contraignante, car elle est celle qui peut le plus probablement produire un assignement valide]

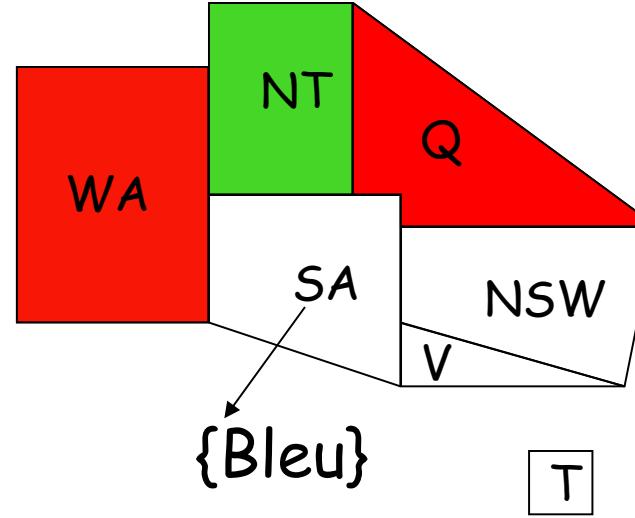
[Note: utiliser cette heuristique demande de faire un "forward-checking" pour chaque valeur, pas seulement pour la valeur sélectionnée]

# Coloration de cartes



- Le domaine de Q a 2 valeurs restantes: Bleu et Rouge
- Assigner Bleu à Q laisserait 0 valeur pour SA, alors qu'assigner Rouge laisserait 1 valeur

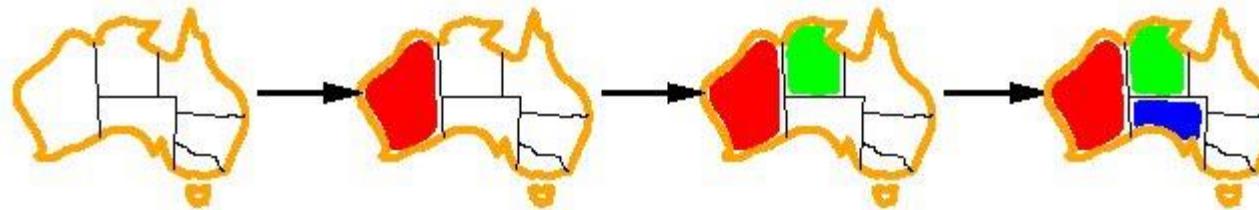
# Coloration de cartes



- Le domaine de Q a 2 valeurs restantes: Bleu et Rouge
- Assigner Bleu à Q laisserait 0 valeur pour SA, alors qu'assigner Rouge laisserait 1 valeur  
→ Donc assigner Rouge à Q

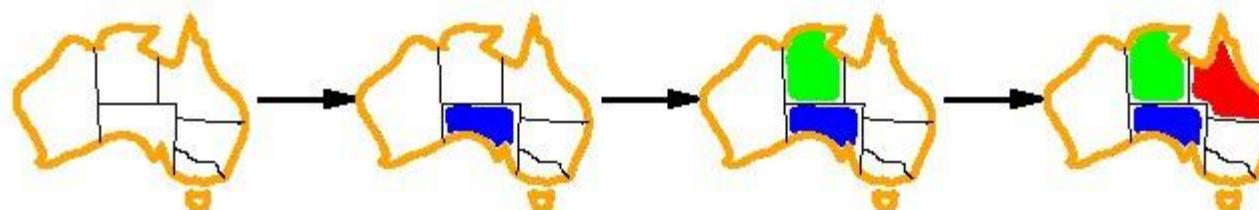
# Résumé

- Heuristique de la variable la plus contrainte
  - sélectionner la variable avec le plus petit nombre de valeurs possibles



but: réduire le facteur de branchement

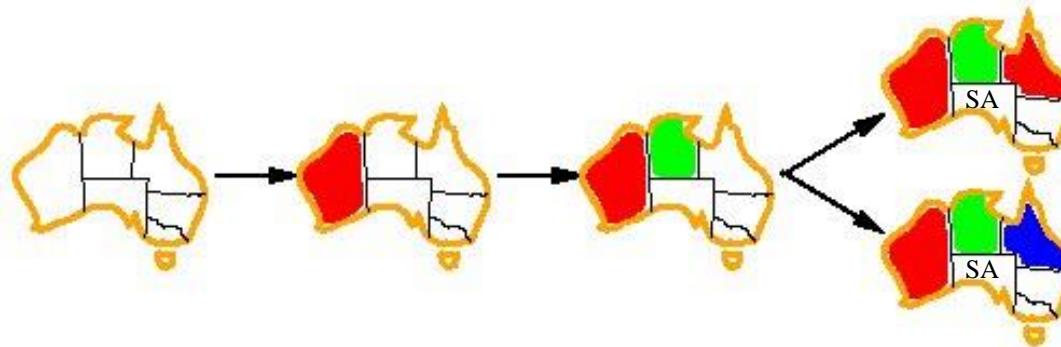
- Heuristique de la variable la plus contraignante
  - sélectionner la variable qui est impliquée dans le plus grand nombre de contraintes sur les variables non encore assignées



but: minimiser le nombre de valeurs restantes possibles

# Résumé

- Heuristique de la valeur la moins contraignante
  - préférer la valeur qui laisse le plus de valeurs possibles pour les autres variables non encore assignées



autorise 1 valeur pour SA (bleu)

autorise 0 valeur pour SA

- Une combinaison de ces différentes heuristiques rend le problème des 1000-reines praticable

# Algorithme Backtracking modifié

PSC-BACKTRACKING( $A$ , var-domaines)

1. Si assignement  $A$  est complet alors retourner  $A$
2.  $X \leftarrow$  sélectionner une variable absente de  $A$
3.  $D \leftarrow$  sélectionner un ordre sur le domaine de  $X$
4. Pour chaque valeur  $v$  dans  $D$  faire
  - a. Add ( $X \leftarrow v$ ) à  $A$
  - b. var-domaines  $\leftarrow$  forward checking(var-domaines,  $X, v, A$ )
  - c. Si aucune variable a une domaine vide alors
    - i. résultat  $\leftarrow$  PSC-BACKTRACKING( $A$ , var-domaines)
    - ii. Si résultat  $\neq$  échec alors retourner résultat
5. Retourner échec

1) Heuristique variable-plus-contrainte  
 2) Heuristique variable-plus-constraignante

1) Heuristique valeur-moins-constraignante

1) Sélectionner la variable ayant le plus petit domaine restant  
 2) Sélectionner la variable apparaissant dans le plus grand nombre de contraintes sur des variables absentes de l'assignement courant

## Applications

---

- Les techniques des PSC permettent de résoudre des problèmes complexes et sont largement utilisées
- De nombreuses applications telles que:
  - attribution d'équipages à des lignes aériennes
  - gestion d'une flotte de transport
  - horaires de trains, d'avions, etc ...
  - ordonnancement et gestion des tâches dans un port marchand
  - conception (en tous genres)
  - procédures/opérations chirurgicales (radiochirurgie)
  - etc ...

# Références

---

- Surveys
  - Kumar, AAAI Mag., 1992
  - Dechter et Frost, AAAI Mag. 1999
- Ouvrages
  - Marriott and Stuckey, 1998
  - AIMA, Russell and Norvig, 2nd ed.
- Applications
  - Freuder and Mackworth, 1994
- Conférence
  - Principles and Practice of Constraint Programming (CP)
- Journal
  - *Constraints* (Kluwer Academic Publishers)
- Internet
  - Constraints Archive  
<http://www.cs.unh.edu/ccc/archive>



# Optimisation Programmation linéaire

hepia HES-SO

Paul Albuquerque

Michel Vinckenbosch

Guido Bologna

# Plan

- Exemples 2D
- Exemples de dimensionnalité supérieure
- Méthode du «tableau»
- Problème auxiliaire
- Problème dual

# Exemple de problème

RESSOURCES			
PRODUIT	Travail (hr/unité)	Argile (Kg/unité)	Revenu (CHF/unité)
Bol	1	4	40
Tasse	2	3	50

On dispose de 40 heures de travail et de 120 kilos d'argile par jour

Variables décisionnelles :

$x_1$  = nombre de bols à produire

$x_2$  = nombre de tasses à produire

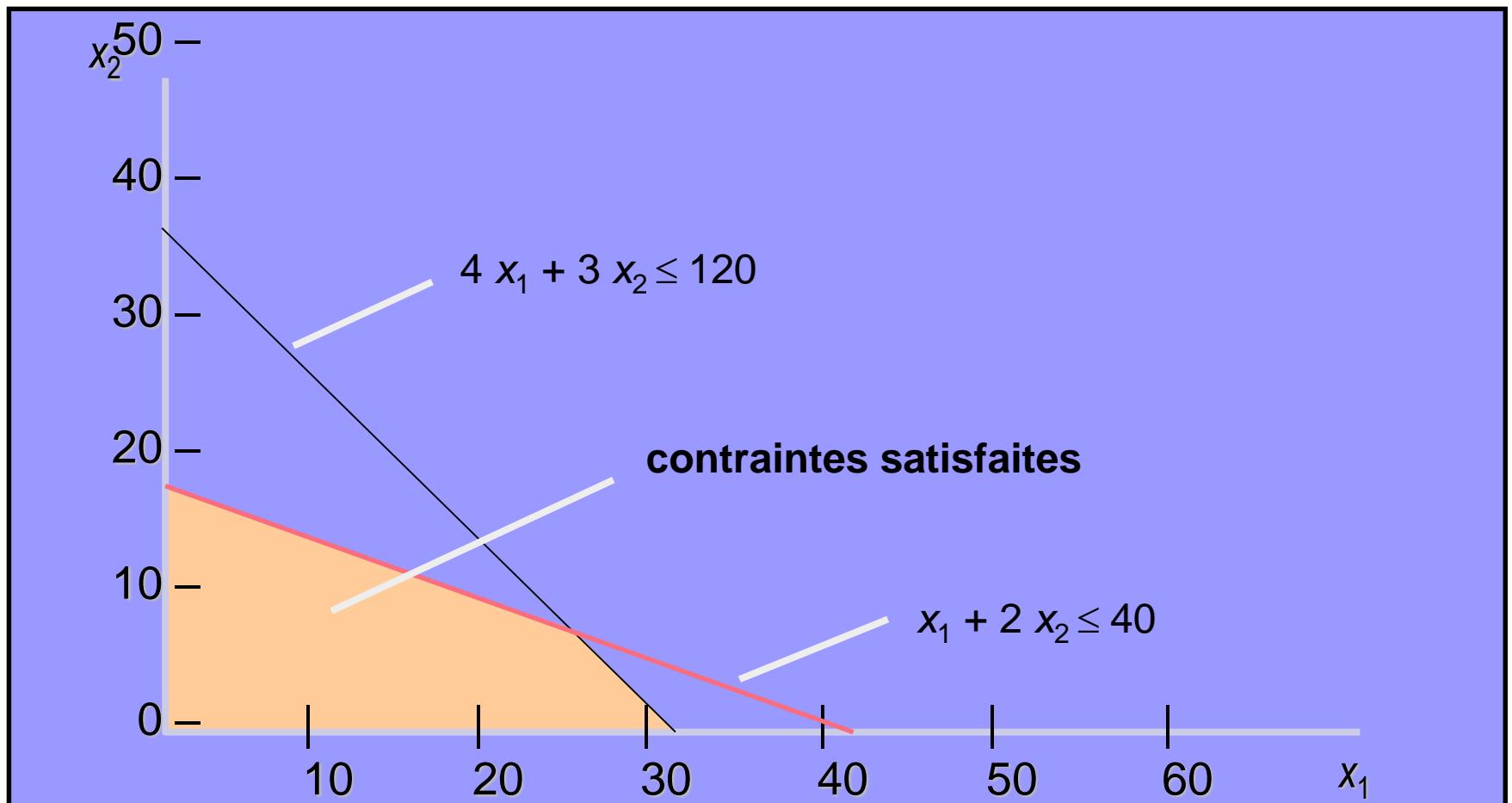
# Exemple de problème

- Maximiser  $Z = 40 x_1 + 50 x_2$
- Sous contraintes :

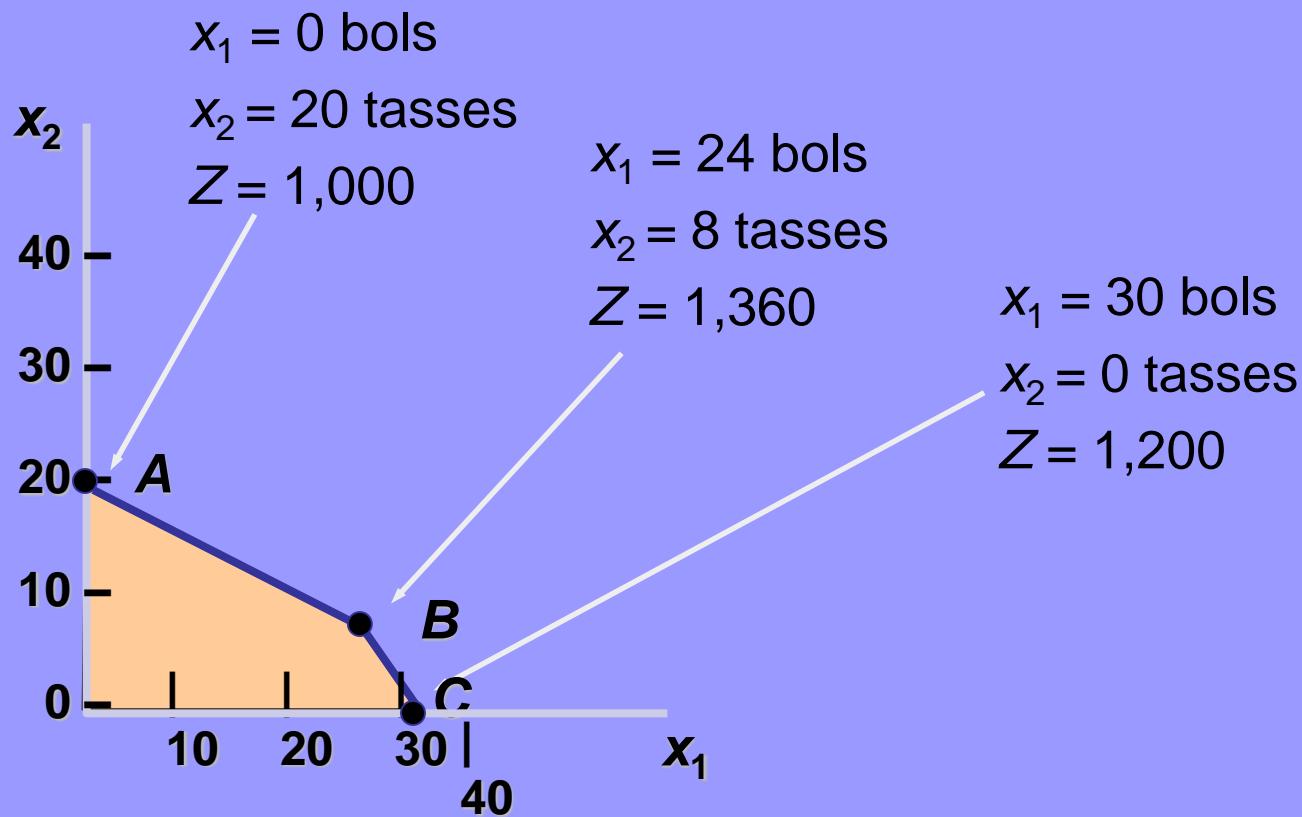
  - $x_1 + 2x_2 \leq 40$  (travail)
  - $4x_1 + 3x_2 \leq 120$  (argile)
  - $x_1, x_2 \geq 0$

- La solution est  $x_1 = 24$  bols et  $x_2 = 8$  tasses
- Revenu = 1360

# Représentation graphique



# Résolution graphique



# Example 2

$$\begin{array}{ll} \text{Max} & 5x_1 + 7x_2 \\ \text{s.c.} & \left. \begin{array}{l} x_1 \leq 6 \\ 2x_1 + 3x_2 \leq 19 \\ x_1 + x_2 \leq 8 \end{array} \right\} \\ & x_1 \geq 0 \text{ and } x_2 \geq 0 \end{array}$$

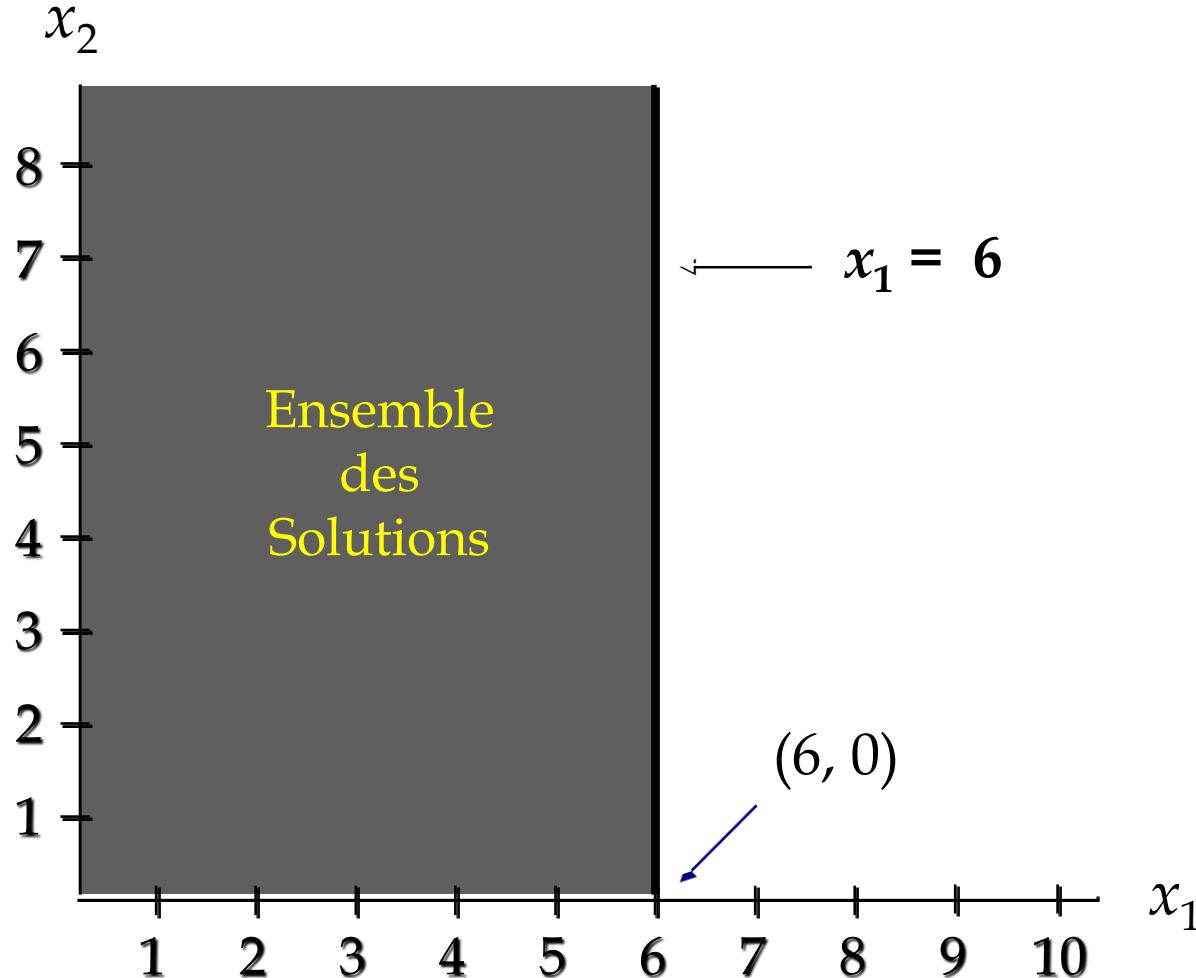
**Fonction objectif**

**Contraintes**

**Contraintes Non-négatives**

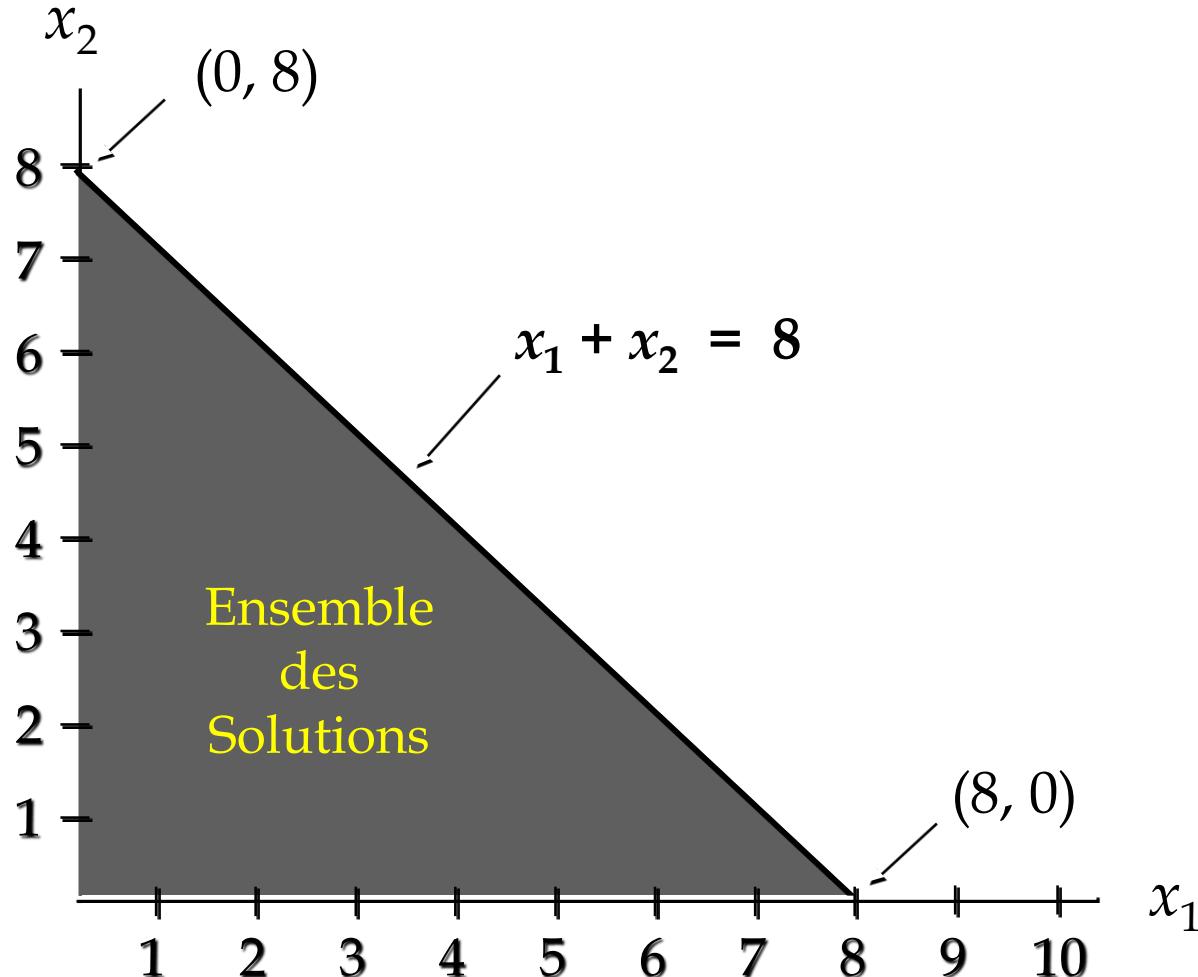
# Example 2

## ■ Première contrainte



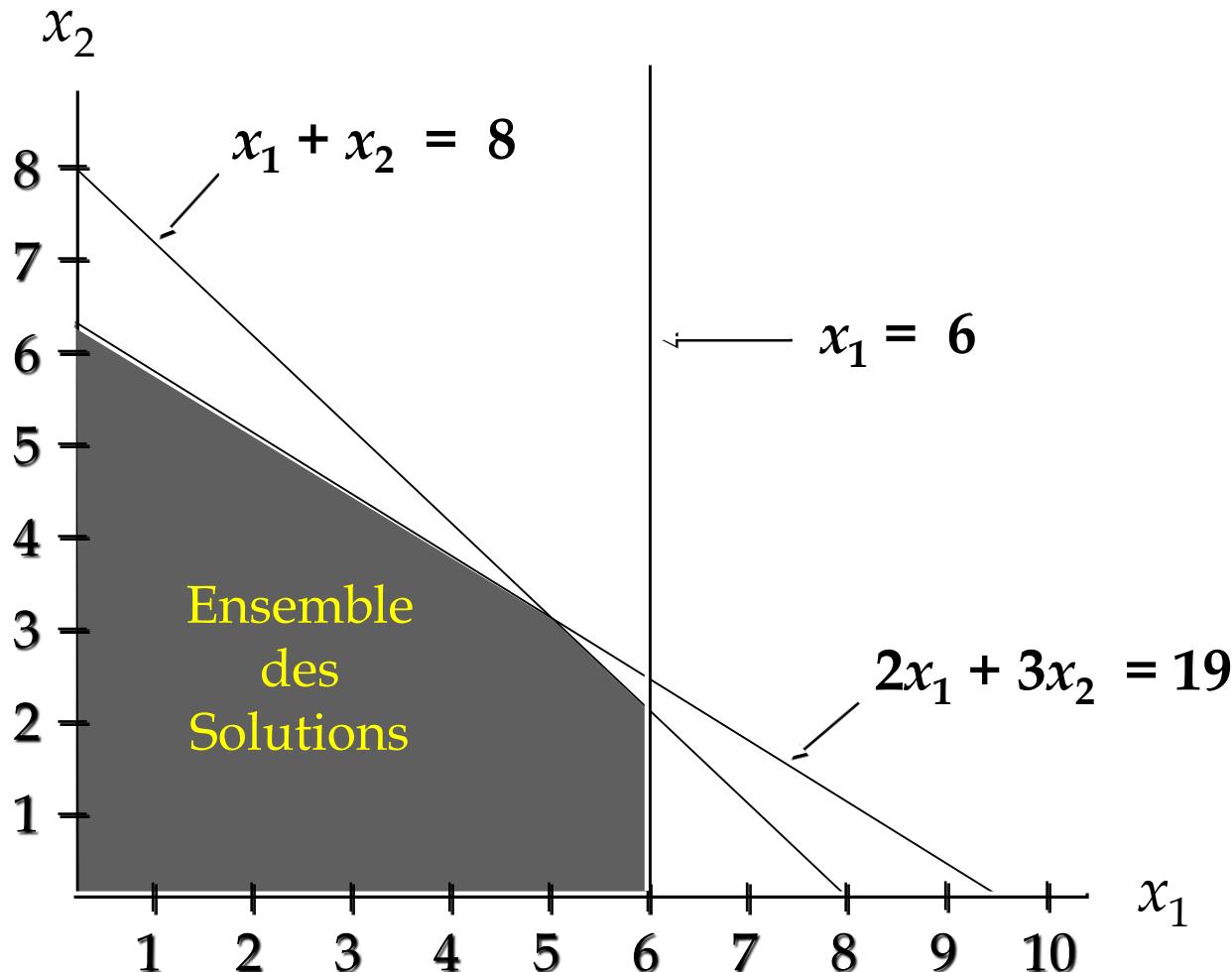
# Example 2

## ■ Deuxième contrainte



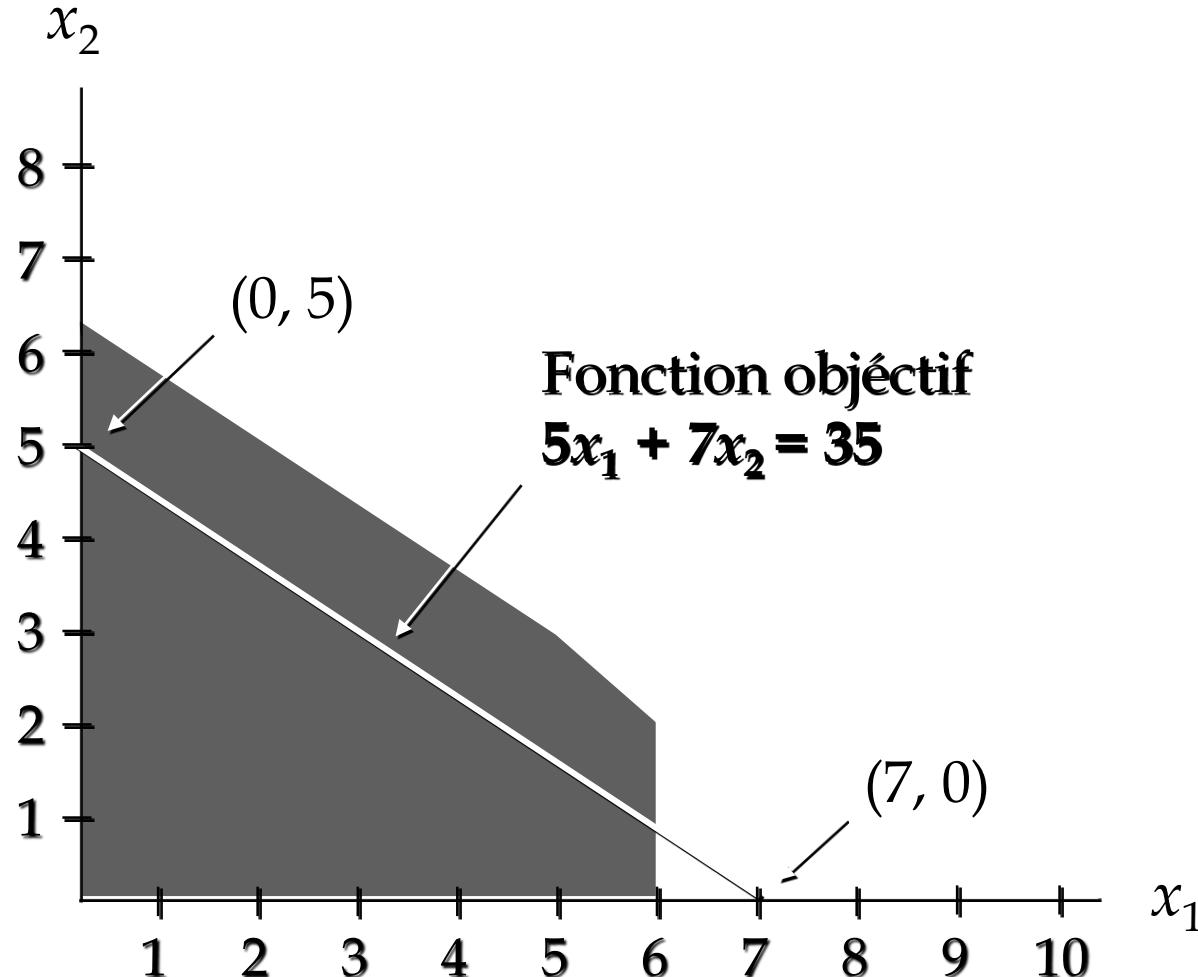
# Example 2

## Combinaison de trois contraintes



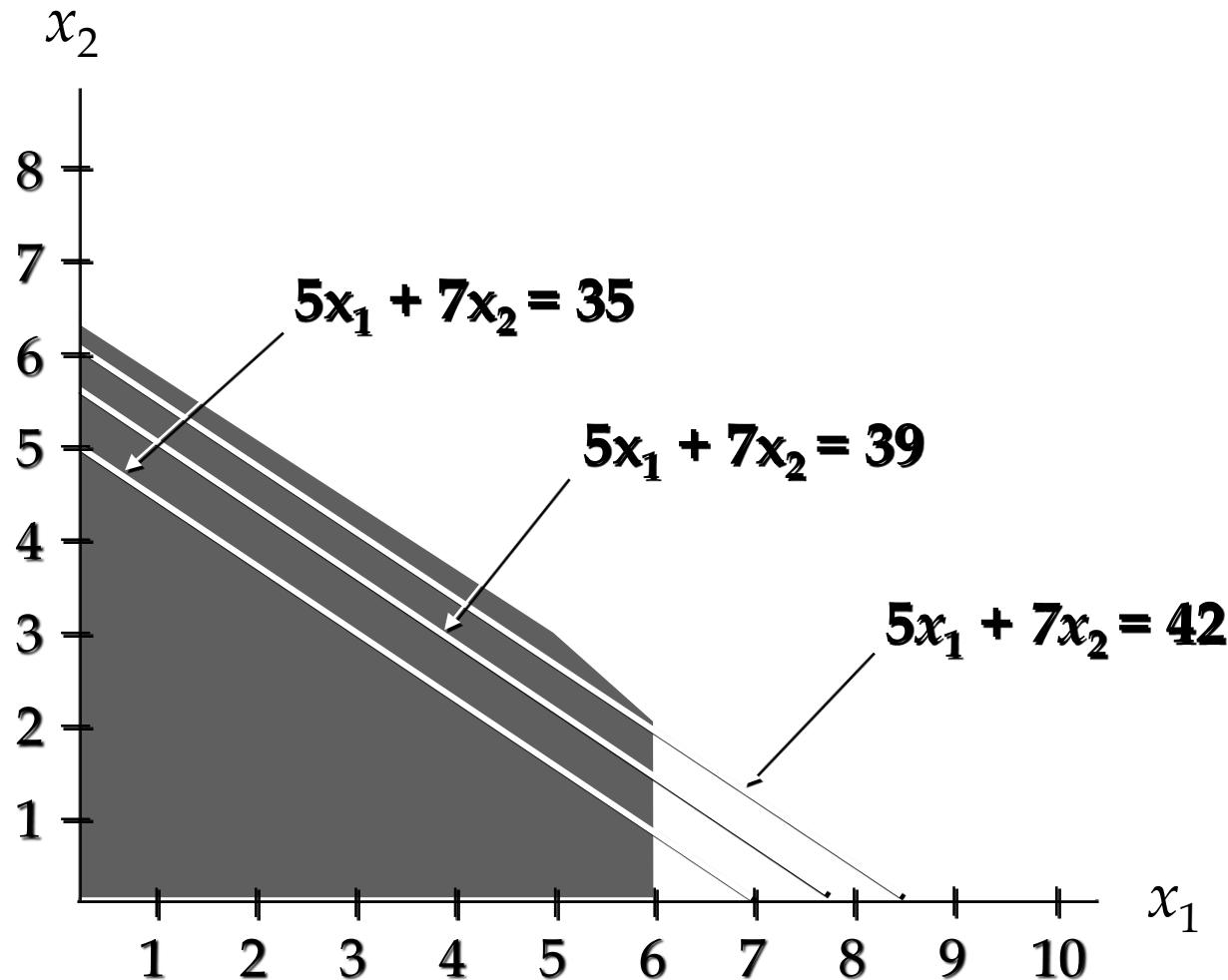
# Example 2

## ■ Fonction objéctif



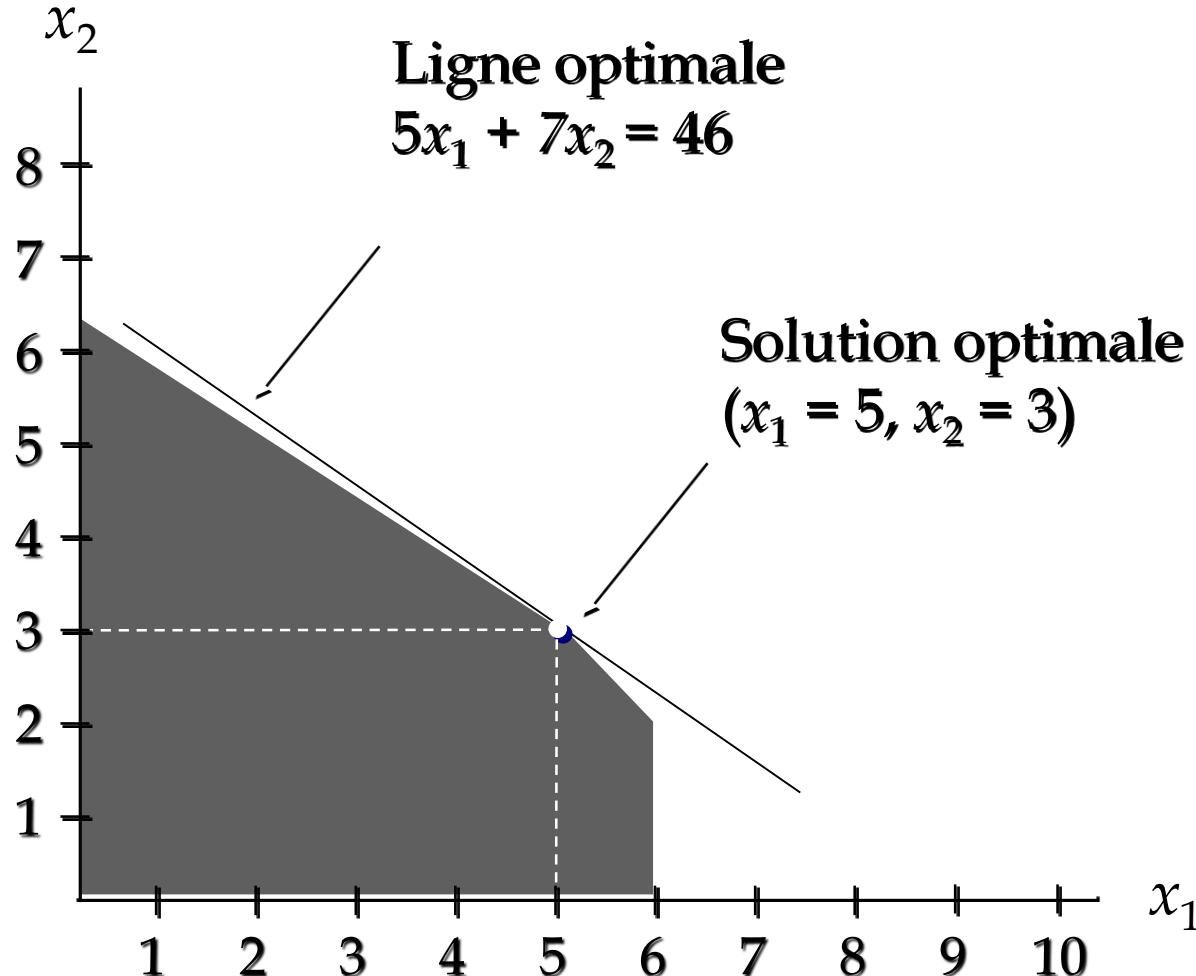
# Example 2

- Fonction objectif avec différentes valeurs de ctes



# Example 2

## ■ Solution Optimale



# Polyèdres

- Hyperplan = ensemble de vecteurs  $(x_1, \dots, x_n)$  tels que  $a_1x_1 + a_2x_2 + \dots + a_nx_n - d = 0$ 
  - Exemples
    - dimension 2 : une droite
    - dimension 3 : un plan
    - dimension 4 : un sous-espace de dimension n
- Polyèdre = espace fini délimité par des hyperplans

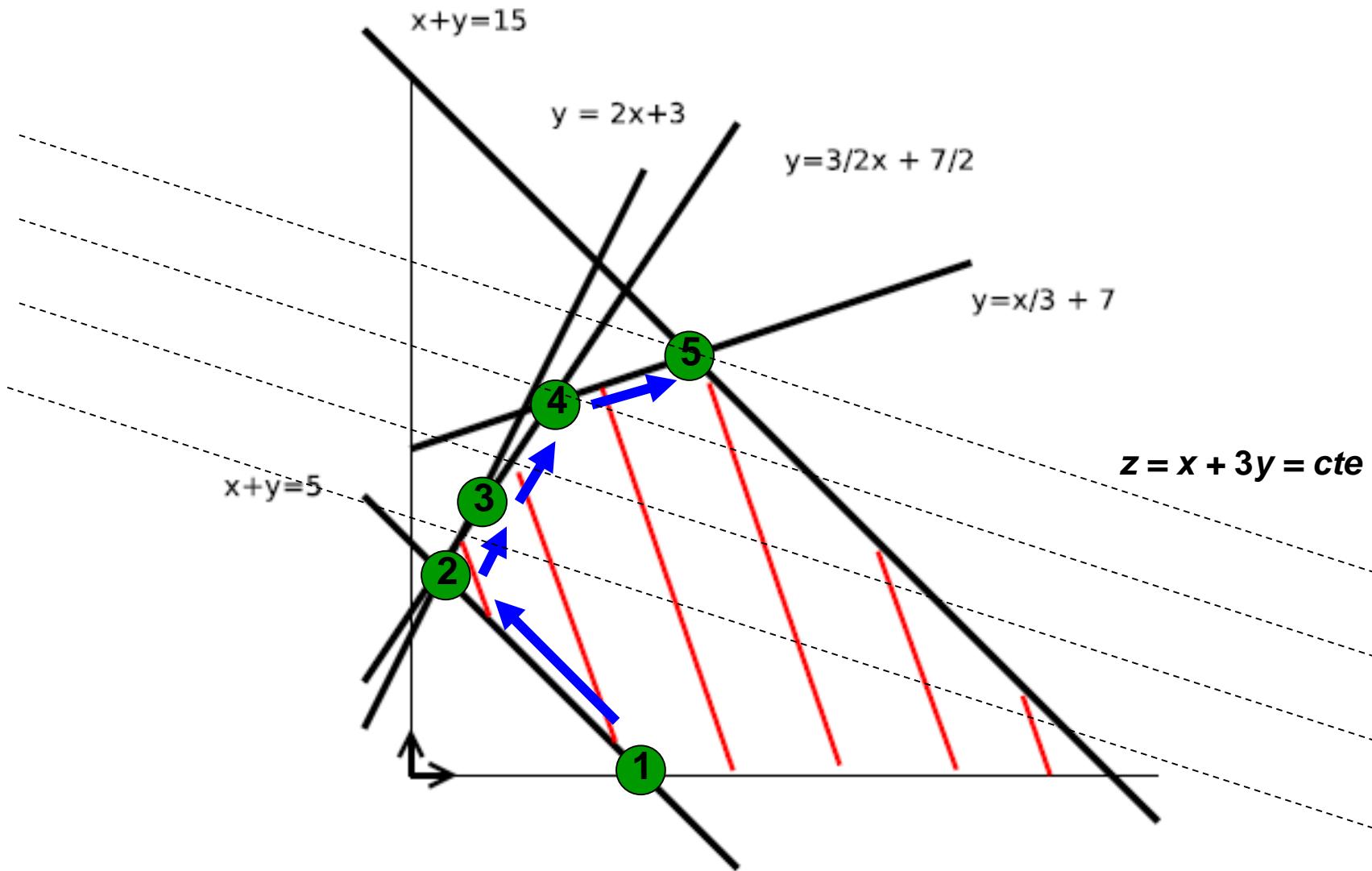
# Polyèdres

- Correspondance algébrique  $\leftrightarrow$  géométrique
  - Nombre de variables  $\leftrightarrow$  dimension de l'espace
  - Contraintes  $\leftrightarrow$  hyperplans
  - Solution  $\leftrightarrow$  tout point à l'intérieur du polyèdre
  - Fonction objectif  $\leftrightarrow$  vecteur (coeffs. de la fonct.)

# Algorithme simple

- Énumérer tous les sommets du polyèdre et choisir le meilleur d'entre eux
- Difficulté
  - $m$  contraintes et  $n$  variables  $\geq 0$
  - $(m+n)!/(m!n!)$  points d'intersections de  $n$  hyperplans
    - $n = 15, m = 10 \Rightarrow (m+n)!/(m!n!) = 3.3 \cdot 10^6$
  - quels points d'intersections sont des sommets du polyèdre (i.e. satisfont les contraintes)?
  - quel sommet maximise l'objectif?

# Algorithme du simplexe (linéaire)



# Algorithme du simplexe

## ■ Principe

- Initialisation sur un sommet  $s$
- Répéter
  - Sélection du sommet voisin de  $s$  qui augmente la fonction objectif

Tant que la fonction objectif croît strictement

## ■ Existence d'une solution optimale

- Si les contraintes délimitent un **polyèdre**  $\neq \emptyset$ , alors il existe une solution optimale sur **un de ses sommets**

# Algorithme du simplexe

- Maximiser  $z = 5x_1 + \frac{13}{2}x_2 + 8x_3 + 9x_4$
- Sous contraintes

$$3x_1 + 6x_2 + 3x_3 + 9x_4 \leq 240$$

$$12x_1 + 15x_2 + 9x_3 + 12x_4 \leq 150$$

$$0.1x_1 + 0.1x_2 + 0.1x_3 + 0.1x_4 \leq 2$$

$$x_1, x_2, x_3, x_4 \geq 0$$

# Forme augmentée

## ■ Introduction de variables d'écart $e_i$

$$3x_1 + 6x_2 + 3x_3 + 9x_4 + e_1 = 240$$

$$12x_1 + 15x_2 + 9x_3 + 12x_4 + e_2 = 150$$

$$0.1x_1 + 0.1x_2 + 0.1x_3 + 0.1x_4 + e_3 = 2$$

$$e_1, e_2, e_3, x_1, x_2, x_3, x_4 \geq 0$$

# Résolution

- Maximiser  $z = 5x_1 + \frac{13}{2}x_2 + 8x_3 + 9x_4$
- Sous contraintes

$$e_1 = 240 - 3x_1 - 6x_2 - 3x_3 - 9x_4$$

$$e_2 = 150 - 12x_1 - 15x_2 - 9x_3 - 12x_4$$

$$e_3 = 2 - 0.1x_1 - 0.1x_2 - 0.1x_3 - 0.1x_4$$

$$e_1, e_2, e_3, x_1, x_2, x_3, x_4 \geq 0$$

- Terminologie :  $e_i$  variables **de base** (ou liées)  
 $x_i$  variables **hors base** (ou libres)

# Résolution: solution initiale

- Maximiser  $z = 5x_1 + \frac{13}{2}x_2 + 8x_3 + 9x_4$
- Sous contraintes

$$e_1 = 240 - 3x_1 - 6x_2 - 3x_3 - 9x_4$$

$$e_2 = 150 - 12x_1 - 15x_2 - 9x_3 - 12x_4$$

$$e_3 = 2 - 0.1x_1 - 0.1x_2 - 0.1x_3 - 0.1x_4$$

$$e_1, e_2, e_3, x_1, x_2, x_3, x_4 \geq 0$$

- Solution initiale :  $x_1 = x_2 = x_3 = x_4 = 0$
- Valeur initiale :  $z = 0$  (à améliorer)

# Résolution: première itération

- Choix de la variable faisant augmenter

$$z = 5x_1 + \frac{13}{2}x_2 + 8x_3 + 9x_4$$

- $\max(5, 13/2, 8, 9) = 9$
- $x_4$  donne le plus fort accroissement de  $z$
- $x_4$  est appelée variable **entrante**

# Résolution: première itération

- Augmenter  $x_4$  tant qu'aucune contrainte n'est violée, i.e. jusqu'à  $e_i = 0$  pour un  $i$

$$e_1 = 240 - 3x_1 - 6x_2 - 3x_3 - 9x_4$$

$$e_2 = 150 - 12x_1 - 15x_2 - 9x_3 - 12x_4$$

$$e_3 = 2 - 0.1x_1 - 0.1x_2 - 0.1x_3 - 0.1x_4$$

$$e_1, e_2, e_3, x_1, x_2, x_3, x_4 \geq 0$$

# Résolution: première itération

- En partant de la solution  $x_1 = x_2 = x_3 = x_4 = 0$  augmenter  $x_4$  jusqu'à  $e_i = 0$  pour un  $i$

$$e_1 = 240 - 3 \cdot 0 - 6 \cdot 0 - 3 \cdot 0 - 9x_4$$

$$e_2 = 150 - 12 \cdot 0 - 15 \cdot 0 - 9 \cdot 0 - 12x_4$$

$$e_3 = 2 - 0.1 \cdot 0 - 0.1 \cdot 0 - 0.1 \cdot 0 - 0.1x_4$$

- $e_1 \geq 0 \Rightarrow x_4 \leq 240/9$
  - $e_2 \geq 0 \Rightarrow x_4 \leq 150/12$
  - $e_3 \geq 0 \Rightarrow x_4 \leq 2/0.1$
- $\left. \begin{array}{l} \\ \\ \end{array} \right\} \Rightarrow \min(240/9, 150/12, 2/0.1) = 150/12$

# Résolution: première itération

- En prenant  $x_4 = 150/12$ , on a :  $e_2 = 0$
- $e_2$  est appelée variable **sortante**
- À cette étape, on dit que  $x_4$  entre dans la base et  $e_2$  sort de la base
- $x_4$  devient une variable de base

$e_2$  - - - hors base

# Résolution: première itération

- On exprime  $x_4$  en fonction de  $e_2$

$$e_1 = 240 - 3x_1 - 6x_2 - 3x_3 - 9x_4$$

$$x_4 = 25/2 - x_1 - 5/4x_2 - 3/4x_3 - 1/12e_2$$

$$e_3 = 2 - 0.1x_1 - 0.1x_2 - 0.1x_3 - 0.1x_4$$

$$e_1, e_2, e_3, x_1, x_2, x_3, x_4 \geq 0$$

- Puis on remplace  $x_4$  dans les autres équations et dans la fonction objectif

# Résolution: seconde itération

- On obtient

$$e_1 = 255/2 + 6x_1 + 21/4x_2 + 15/4x_3 + 3/4e_2$$

$$x_4 = 25/2 - x_1 - 5/4x_2 - 3/4x_3 - 1/12e_2$$

$$e_3 = 3/4 + 1/40x_2 - 1/40x_3 + 1/120e_2$$

---

$$z = 225/2 - 4x_1 - 19/4x_2 + 5/4x_3 - 3/4e_2$$

- Variables de base :  $e_1, x_4, e_3$

- hors base :  $x_1, x_2, x_3, e_2$

# Résolution: seconde itération

## ■ Choix de la variable entrante

- $z = 225/2 - 4x_1 - 19/4x_2 + 5/4x_3 - 3/4e_2$
- $\max(-4, -19/4, 5/4, -3/4) = 5/4$
- $x_3$  donne le plus fort accroissement de  $z$
- $x_3$  est la variable entrante

# Résolution: seconde itération

- En partant de la solution  $x_1 = x_2 = x_3 = e_2 = 0$  augmenter  $x_3$  jusqu'à  $e_1, x_4$  ou  $e_3 = 0$

$$e_1 = 255/2 + 6 \cdot 0 + 21/4 \cdot 0 + 15/4x_3 + 3/4 \cdot 0$$

$$x_4 = 25/2 - 0 - 5/4 \cdot 0 - 3/4x_3 - 1/12 \cdot 0$$

$$e_3 = 3/4 + 1/40 \cdot 0 - 1/40x_3 + 1/120 \cdot 0$$

- $e_1 \geq 0 \Rightarrow$  indép. de  $x_3$
  - $x_4 \geq 0 \Rightarrow x_3 \leq 50/3$
  - $e_3 \geq 0 \Rightarrow x_3 \leq 30$
- $\left. \begin{array}{l} \\ \\ \end{array} \right\} \Rightarrow \min(50/3, 30) = 50/3$

# Résolution: seconde itération

- En prenant  $x_3 = 50/3$ , on a :  $x_4 = 0$
- $x_4$  est la variable sortante
- $x_3$  devient une variable de base

$x_4$  - - - - hors base

- On remplace ensuite

$$x_3 = 50/3 - 4/3 x_1 - 5/3 x_2 - x_4 - 1/9 e_2$$

dans les équations pour  $e_1$ ,  $e_3$ ,  $z$

# Résolution: seconde itération

- On obtient

$$e_1 = 190 + x_1 - x_2 + 5x_4 + 1/3e_2$$

$$x_3 = 50/3 - 4/3x_1 - 5/3x_2 - 4/3x_4 - 1/9e_2$$

$$e_3 = 1/3 - 1/30x_4 + 1/15x_2 - 1/30x_4 + 1/90e_2$$

---

$$z = 400/3 - 17/3x_1 - 41/6x_2 - 5/3x_4 - 8/9e_2$$

- Variables de base :  $e_1, x_3, e_3$ 
  - hors base :  $x_1, x_2, x_4, e_2$

# Terminaison de l'algorithme

## ■ Choix de la variable entrante

- $z = 400/3 - 17/3x_1 - 41/6x_2 - 5/3x_4 - 8/9e_2$
- $\max(-17/3, -41/6, -5/3, -8/9) \leq 0$
- Impossible d'augmenter  $z$
- L'optimum est atteint en  $x_1 = x_2 = x_4 = e_2 = 0$   
et vaut  $400/3$

Les équations pour  $e_1$ ,  $x_3$ ,  $e_3$  donnent

$$e_1 = 190, x_3 = 50/3, e_3 = 1/3$$

# Algorithme du simplexe

- Remarques
  - Initialisation
    - Quelle solution de départ? L'origine?
  - Itération
    - Une stagnation est-elle possible?
  - Terminaison
    - L'algorithme se termine-t-il toujours?

# Résolution avec tableaux

## ■ Maximiser

$$5x_1 + 6.5x_2 + 8x_3 + 9x_4 = z$$

## ■ Sous contraintes

$$3x_1 + 6x_2 + 3x_3 + 9x_4 + e_1 = 240$$

$$12x_1 + 15x_2 + 9x_3 + 12x_4 + e_2 = 150$$

$$0.1x_1 + 0.1x_2 + 0.1x_3 + 0.1x_4 + e_3 = 2$$

$$e_1, e_2, e_3, x_1, x_2, x_3, x_4 \geq 0$$

# Résolution avec tableaux

## ■ Nouvelle représentation

variables hors base				variables de base			
$x_1$	$x_2$	$x_3$	$x_4$	$e_1$	$e_2$	$e_3$	
3	6	3	9	1	0	0	240
12	15	9	12	0	1	0	150
0, 1	0, 1	0, 1	0, 1	0	0	1	2
5	6, 5	8	9	0	0	0	$Z$

# Résolution avec tableaux

- Itération 1: choix de la variable entrante

$x_1$	$x_2$	$x_3$	$x_4$	$e_1$	$e_2$	$e_3$	
3	6	3	9	1	0	0	240
12	15	9	12	0	1	0	150
0, 1	0, 1	0, 1	0, 1	0	0	1	2
5	6, 5	8	9	0	0	0	Z

- $x_4$  entre dans la base

# Résolution avec tableaux

- Itération 1: choix de la variable sortante

$x_1$	$x_2$	$x_3$	$x_4$	$e_1$	$e_2$	$e_3$	
3	6	3	9	1	0	0	240
12	15	9	12	0	1	0	150
0, 1	0, 1	0, 1	0, 1	0	0	1	2
5	6, 5	8	9	0	0	0	$Z$

- $\min(240/9, 150/12, 2/0.1) \Rightarrow e_2$  sort de la base

# Résolution avec tableaux

- Itération 1:  $x_4$  rentre et  $e_2$  sort de la base  
⇒ on échange les colonnes de  $x_4$  et  $e_2$

variables hors base				variables de base			
$x_1$	$x_2$	$x_3$	$e_2$	$e_1$	$x_4$	$e_3$	
3	6	3	0	1	9	0	240
12	15	9	1	0	12	0	150
0, 1	0, 1	0, 1	0	0	0, 1	1	2
5	6, 5	8	0	0	9	0	$Z$

# Résolution avec tableaux

- Itération 1: reconstruction de la matrice identité
  - on normalise la seconde ligne par 12

$x_1$	$x_2$	$x_3$	$e_2$	$e_1$	$x_4$	$e_3$	
3	6	3	0	1	9	0	240
1	$\frac{15}{12}$	$\frac{9}{12}$	$\frac{1}{12}$	0	1	0	$\frac{150}{12}$
1	$\frac{5}{4}$	$\frac{3}{4}$	$\frac{1}{12}$	0	1	0	$\frac{25}{2}$
0, 1	0, 1	0, 1	0	0	0, 1	1	2
5	6, 5	8	0	0	9	0	Z

- puis on l'utilise comme **pivot** pour éliminer  $x_4$  des autres lignes

# Résolution avec tableaux

- À la fin de la première itération

$x_1$	$x_2$	$x_3$	$e_2$	$e_1$	$x_4$	$e_3$	
-6	$-\frac{21}{4}$	$-\frac{15}{4}$	$-\frac{3}{4}$	1	0	0	$\frac{255}{2}$
1	$\frac{5}{4}$	$\frac{3}{4}$	$\frac{1}{12}$	0	1	0	$\frac{25}{2}$
0	$-\frac{1}{40}$	$\frac{1}{40}$	$-\frac{1}{120}$	0	0	1	$\frac{3}{4}$
5	6,5	8	0	0	9	0	Z

# Résolution avec tableaux

- Itération 2 : choix de la variable entrante

$x_1$	$x_2$	$x_3$	$e_2$	$e_1$	$x_4$	$e_3$	
-6	$-\frac{21}{4}$	$-\frac{15}{4}$	$-\frac{3}{4}$	1	0	0	$\frac{255}{2}$
1	$\frac{5}{4}$	$\frac{3}{4}$	$\frac{1}{12}$	0	1	0	$\frac{25}{2}$
0	$-\frac{1}{40}$	$\frac{1}{40}$	$-\frac{1}{120}$	0	0	1	$\frac{3}{4}$
-4	$-\frac{19}{4}$	$\frac{5}{4}$	$-\frac{3}{4}$	0	0	0	$Z - \frac{225}{2}$

- $x_3$  entre dans la base

# Résolution avec tableaux

## ■ Itération 2 : choix de la variable sortante

$x_1$	$x_2$	$x_3$	$e_2$	$e_1$	$x_4$	$e_3$	
-6	$-\frac{21}{4}$	$-\frac{15}{4}$	$-\frac{3}{4}$	1	0	0	$\frac{255}{2}$
1	$\frac{5}{4}$	$\frac{3}{4}$	$\frac{1}{12}$	0	1	0	$\frac{25}{2}$
0	$-\frac{1}{40}$	$\frac{1}{40}$	$-\frac{1}{120}$	0	0	1	$\frac{3}{4}$
-4	$-\frac{19}{4}$	$\frac{5}{4}$	$-\frac{3}{4}$	0	0	0	$Z - \frac{225}{2}$

■  $\min(50/3, 30) \Rightarrow x_4$  sort de la base

# Résolution avec tableaux

- Itération 2 :  $x_3$  rentre et  $x_4$  sort de la base  
⇒ on échange les colonnes de  $x_3$  et  $x_4$

variables hors base				variables de base			
$x_1$	$x_2$	$x_4$	$e_2$	$e_1$	$x_3$	$e_3$	
-6	$-\frac{21}{4}$	0	$-\frac{3}{4}$	1	$-\frac{15}{4}$	0	$\frac{255}{2}$
1	$\frac{5}{4}$	1	$\frac{1}{12}$	0	$\frac{3}{4}$	0	$\frac{25}{2}$
0	$-\frac{1}{40}$	0	$-\frac{1}{120}$	0	$\frac{1}{40}$	1	$\frac{3}{4}$
-4	$-\frac{19}{4}$	0	$-\frac{3}{4}$	0	$\frac{5}{4}$	0	$Z - \frac{225}{2}$

# Résolution avec tableaux

- Itération 2 : reconstruction de la matrice identité
  - on normalise la seconde ligne par 3/4
  - puis on supprime  $x_3$  des autres lignes

$x_1$	$x_2$	$x_4$	$e_2$	$e_1$	$x_3$	$e_3$	
-1	1	5	$-\frac{1}{3}$	1	0	0	190
$\frac{4}{3}$	$\frac{5}{3}$	$\frac{4}{3}$	$\frac{1}{9}$	0	1	0	$\frac{50}{3}$
$-\frac{1}{30}$	$-\frac{1}{15}$	$-\frac{1}{30}$	$-\frac{1}{90}$	0	0	1	$\frac{1}{3}$
$-\frac{17}{3}$	$-\frac{41}{6}$	$-\frac{5}{3}$	$-\frac{8}{9}$	0	0	0	$Z - \frac{400}{3}$

- Coefficients dans z tous  $\leq 0 \Rightarrow$  on a terminé

# Principes généraux

- Mise sous forme normale
- Itération
  - Choix d'un pivot qui accroît la fonction objectif
  - Détection d'un optimum ou de l'infaisabilité
- Problèmes possibles
  - Solution non-bornée
  - Infaisabilité
  - Cycles
  - Solution initiale

# Difficultés du simplexe

- Itération
  - Peut-on toujours itérer vers l'optimum?
- Terminaison
  - Les itérations se terminent-elles toujours?
- Initialisation
  - Peut-on toujours trouver une solution initiale?

# Solution non-bornée

$$x_2 = 5 + 2x_3 - x_4 - 3x_1$$

$$x_5 = 7 - 3x_4 - 4x_1$$

$$z = 5 + x_3 - x_4 - x_1$$

$x_3$  entre dans la base

Pas de borne supérieure sur  $x_3$ :  $x_3 \geq -5/2$

Valeur de z arbitrairement grande !

Pas de solution optimale

# Forme matricielle

- Maximiser  $z = c^t x$   
sous contraintes

$$Ax \leq b$$

$$x \geq 0$$

- Si  $b \geq 0$ , alors l'origine est une solution admissible,  
sinon elle n'appartient pas au simplexe

# Initialisation

Maximiser  $z = x_1 - x_2 + x_3$   
sous contraintes

$$2x_1 - x_2 + 2x_3 \leq 4$$

$$2x_1 - 3x_2 + x_3 \leq -5$$

$$-x_1 + x_2 - 2x_3 \leq -1$$

$$x_1, x_2, x_3 \geq 0$$

- L'origine n'est pas une solution admissible
- But: construire une solution admissible

# Initialisation: problème auxiliaire

Maximiser  $w = -x_0$   
sous contraintes

$$2x_1 - x_2 + 2x_3 - x_0 \leq 4$$

$$2x_1 - 3x_2 + x_3 - x_0 \leq -5$$

$$-x_1 + x_2 - 2x_3 - x_0 \leq -1$$

$$x_0, x_1, x_2, x_3 \geq 0$$

# Initialisation

$$x_4 = \begin{array}{rcccccc} 4 & -2x_1 & + x_2 & -2x_3 & + x_0 \\ x_5 = & -5 & -2x_1 & + 3x_2 & -x_3 & + x_0 \\ x_6 = & -1 & + x_1 & - x_2 & + 2x_3 & + x_0 \end{array}$$

$$w = -x_0$$

- Démarrage normale du simplexe impossible!
- Pivot:  $x_0$  entre et  $x_5$  sort car  $\min(4, -5, -1) = -5$

# Initialisation

$$x_0 = 5 + 2x_1 - 3x_2 + x_3 + x_5$$

$$x_4 = 9 - 2x_2 - x_3 + x_5$$

$$x_6 = 4 + 3x_1 - 4x_2 + 3x_3 + x_5$$

$$w = -5 - 2x_1 + 3x_2 - x_3 - x_5$$

- Itérations normales du simplexe
- Pivot:  $x_2$  entre et  $x_6$  sort

# Initialisation

$$x_2 = 1 + 0.75x_1 + 0.75x_3 + 0.25x_5 - 0.25x_6$$

$$x_0 = 2 - 0.25x_1 - 1.25x_3 + 0.25x_5 + 0.75x_6$$

$$x_4 = 7 - 1.5x_1 - 2.5x_3 + 0.5x_5 + 0.5x_6$$

$$w = -2 + 0.25x_1 + 1.25x_3 - 0.25x_5 - 0.75x_6$$

- Pivot:  $x_3$  entre et  $x_0$  sort

# Initialisation

$$x_3 = 1.6 - 0.2x_1 + 0.2x_5 + 0.6x_6 - 0.8x_0$$

$$x_2 = 2.2 + 0.6x_1 + 0.4x_5 + 0.2x_6 - 0.6x_0$$

$$x_4 = 3 - x_1 - x_6 + 2x_0$$

$$w = -x_0$$

- Optimum:  $x_0 = 0, x_2 = 2.2, x_3 = 1.6, x_4 = 3$
- Il faut retraduire dans le **lexique** original

# Initialisation

$$x_3 = \quad 1.6 \quad -0.2x_1 \quad +0.2x_5 \quad +0.6x_6$$

$$x_2 = \quad 2.2 \quad +0.6x_1 \quad +0.4x_5 \quad +0.2x_6$$

$$x_4 = \quad 3 \quad \quad \quad -x_1 \quad \quad \quad \quad \quad -x_6$$

---

$$z = \quad -0.6 \quad +0.2x_1 \quad -0.2x_5 \quad +0.4x_6$$

- On a posé:  $x_0 = 0$ , puis remplacé  $x_2$  et  $x_3$  dans z
- On obtient ainsi un dictionnaire admissible pour le problème original

# Initialisation: principe général

- 1<sup>ère</sup> étape:  $x_0$  entre et une autre variable sort
- Étape générale: itération normale du simplexe
- Terminaison
  - $x_0$  n'est pas dans la base et  $w = 0 \Rightarrow$  problème résoluble
  - $x_0$  est dans la base et  $w \neq 0 \Rightarrow$  problème insoluble

# Simplexe à deux phases

- Phase 1
  - Résolution du problème auxiliaire
- Phase 2
  - Résolution du problème originel à partir du tableau / dictionnaire obtenu au terme de la phase 1
- Existence d'une solution: cas possibles
  - Chaque problème de programmation linéaire est soit
    - infaisable (polyèdre vide)
    - non borné (polyèdre ouvert)
    - résoluble (polyèdre non-vide)

# Dualité: motivations

Maximiser  $z = 4x_1 + x_2 + 5x_3 + 3x_4$   
sous contraintes

$$x_1 - x_2 - x_3 + 3x_4 \leq 1$$

$$5x_1 + x_2 + 3x_3 + 8x_4 \leq 55$$

$$-x_1 + x_2 + 5x_3 + 3x_4 \leq 3$$

$$x_1, x_2, x_3, x_4 \geq 0$$

- Borne inférieure sur  $z$  : solutions admissibles
  - $z > 5$  avec  $x = (0,0,1,0)$        $z > 22$  avec  $x = (3,0,2,0)$
- Borne supérieure sur  $z$  ?

# Dualité: motivations

Maximiser  $z = 4x_1 + x_2 + 5x_3 + 3x_4$   
sous contraintes

$$x_1 - x_2 - x_3 + 3x_4 \leq 1$$

$$5x_1 + x_2 + 3x_3 + 8x_4 \leq 55$$

$$-x_1 + x_2 + 5x_3 + 3x_4 \leq 3$$

$$x_1, x_2, x_3, x_4 \geq 0$$

- Borne supérieure sur  $z$ 
  - Mult. la 2<sup>ème</sup> contrainte par 5/3, comparaison avec  $z$ 
$$\begin{aligned} z &= 4x_1 + x_2 + 5x_3 + 3x_4 \\ &\leq 25/3 x_1 + 5/3 x_2 + 5x_3 + 40/3 x_4 \leq 275/3 \end{aligned}$$

# Dualité: motivations

Maximiser  $z = 4x_1 + x_2 + 5x_3 + 3x_4$   
sous contraintes

$$x_1 - x_2 - x_3 + 3x_4 \leq 1$$

$$5x_1 + x_2 + 3x_3 + 8x_4 \leq 55$$

$$-x_1 + x_2 + 5x_3 + 3x_4 \leq 3$$

$$x_1, x_2, x_3, x_4 \geq 0$$

## ■ Borne supérieure sur $z$

- Addition des 2<sup>ème</sup> et 3<sup>ème</sup> contraintes, comparaison avec  $z$

$$z = 4x_1 + x_2 + 5x_3 + 3x_4 \leq 4x_1 + 2x_2 + 8x_3 + 11x_4 \leq 58$$

- Idée: combinaison linéaire des contraintes

# Dualité: motivations

Maximiser  $z = 4x_1 + x_2 + 5x_3 + 3x_4$   
sous contraintes

$$(x_1 - x_2 - x_3 + 3x_4 \leq 1) \bullet y_1$$

$$(5x_1 + x_2 + 3x_3 + 8x_4 \leq 55) \bullet y_2^+$$

$$(-x_1 + x_2 + 3x_3 - 5x_4 \leq 3) \bullet y_3^+$$

$$x_1, x_2, x_3, x_4 \geq 0$$

## ■ On obtient

$$(y_1 + 5y_2 - y_3)x_1 + (-y_1 + y_2 + 2y_3)x_2$$

$$+ (-y_1 + 3y_2 + 3y_3)x_3 + (3y_1 + 8y_2 - 5y_3)x_4 \leq y_1 + 55y_2 + 3y_3$$

## ■ Condition pour que le terme de gauche majore $z$ ?

# Dualité: motivations

- $$\begin{aligned} z &= 4x_1 + x_2 + 5x_3 + 3x_4 \\ &\leq (y_1 + 5y_2 - y_3) x_1 + (-y_1 + y_2 + 2y_3) x_2 + (-y_1 + 3y_2 + 3y_3) x_3 \\ &\quad + (3y_1 + 8y_2 - 5y_3) x_4 \leq y_1 + 55y_2 + 3y_3 \end{aligned}$$

## ■ Contraintes

$$\begin{aligned} y_1 + 5y_2 - y_3 &\geq 4 \\ -y_1 + y_2 + 2y_3 &\geq 1 \\ -y_1 + 3y_2 + 3y_3 &\geq 5 \\ 3y_1 + 8y_2 - 5y_3 &\geq 3 \\ y_1, y_2, y_3 &\geq 0 \end{aligned}$$

- Pour obtenir la meilleure borne supérieure, il faut minimiser  $y_1 + 55y_2 + 3y_3$

# Problèmes primal et dual

■ Maximiser	$z = 4x_1 + x_2 + 5x_3 + 3x_4$	Problème primal
sous contraintes	$x_1 - x_2 - x_3 + 3x_4 \leq 1$	
	$5x_1 + x_2 + 3x_3 + 8x_4 \leq 55$	
	$-x_1 + x_2 + 3x_3 - 5x_4 \leq 3$	
	$x_1, x_2, x_3, x_4 \geq 0$	

■ Minimiser	$w = y_1 + 55y_2 + 3y_3$	Problème dual
sous contraintes	$y_1 + 5y_2 - y_3 \geq 4$	
	$-y_1 + y_2 + 2y_3 \geq 1$	
	$-y_1 + 3y_2 + 3y_3 \geq 5$	
	$3y_1 + 8y_2 - 5y_3 \geq 3$	
	$y_1, y_2, y_3 \geq 0$	

# Problèmes dual et primal

- Maximiser sous contraintes

$$z = \sum_{j=1}^n c_j x_j \quad \text{ou } z = cx$$

$$\sum_{j=1}^n a_{ij} x_j \leq b_i \quad \text{ou } Ax \leq b$$

$$x_j \geq 0 \quad (i = 1, \dots, m, \quad j = 1, \dots, n)$$

Problème primal

- Minimiser sous contraintes

$$w = \sum_{i=1}^m b_i y_i \quad \text{ou } w = b^t y$$

$$\sum_{i=1}^m a_{ij} y_i \geq c_j \quad \text{ou } A^t y \geq c$$

$$y_i \geq 0 \quad (i = 1, \dots, m, \quad j = 1, \dots, n)$$

Problème dual

# Dualité: remarques

- Problème primal de maximisation
  - ↔ Problème dual de minimisation
- nb.  $m$  de contraintes > nb.  $n$  de variables libres
  - ⇒ problème dual plus avantageux à résoudre  
(nb. d'itérations typique % nb. de variables)

$$\begin{aligned} z = \sum_{j=1}^n c_j x_j &\leq \sum_{j=1}^n \left( \sum_{i=1}^m a_{ij} y_i \right) x_j \\ &\leq \sum_{i=1}^m \left( \sum_{j=1}^n a_{ij} x_j \right) y_i \leq \sum_{i=1}^m b_i y_i = w \end{aligned}$$

# Complexité: mauvais cas

- Maximiser sous contraintes

$$z = \sum_{j=1}^n 10^{n-j} x_j$$

$$\left( 2 \sum_{j=1}^{i-1} 10^{i-j} x_j \right) + x_i \leq 100^{i-1}$$

$$x_i \geq 0 \quad (i = 1, \dots, n)$$

- Nombre d'étapes de l'algorithme du simplexe:  $2^n$
- L'algorithme du simplexe est non-polynomial
- « De tels mauvais cas sont rares »

# Historique

- Algorithme du simplexe (Dantzig, 1947)
  - Planification de l'US Air Force
- Applications à la productique
  - Problème d'allocation de ressources
  - Formalisation de problèmes de décision
- Applications en économie
  - Prix Nobel 1975 (Kantorovich / Koopmans)
- Théorie math. plus ancienne (Fourier, 19<sup>ème</sup> s.)
  - Mise en pratique possible avec l'informatique
- Programmation linéaire  $\supsetneq$  algo. du simplexe
  - variables entières  $\Rightarrow$  difficile / résolution différente

# Bibliographie

- V. Chvátal, *Linear Programming*, Freeman, 1983
- D. de Werra, *Éléments de programmation linéaire avec applications aux graphes*, Presses Polytechniques Romandes, 1990
- R. Faure, B. Lemaire et C. Picouleau. *Précis de recherche opérationnelle: méthodes et exercices d'application*, 5<sup>ème</sup> édition, Dunod, 2000
- D. de Werra, T. M. Liebling et J.-F. Hêche, *Recherche opérationnelle pour ingénieurs*, Presses polytechniques et universitaires romandes, 2003



# Optimisation Reclut simulé

hepia HES-SO

Paul Albuquerque

Michel Vinckenbosch

Guido Bologna

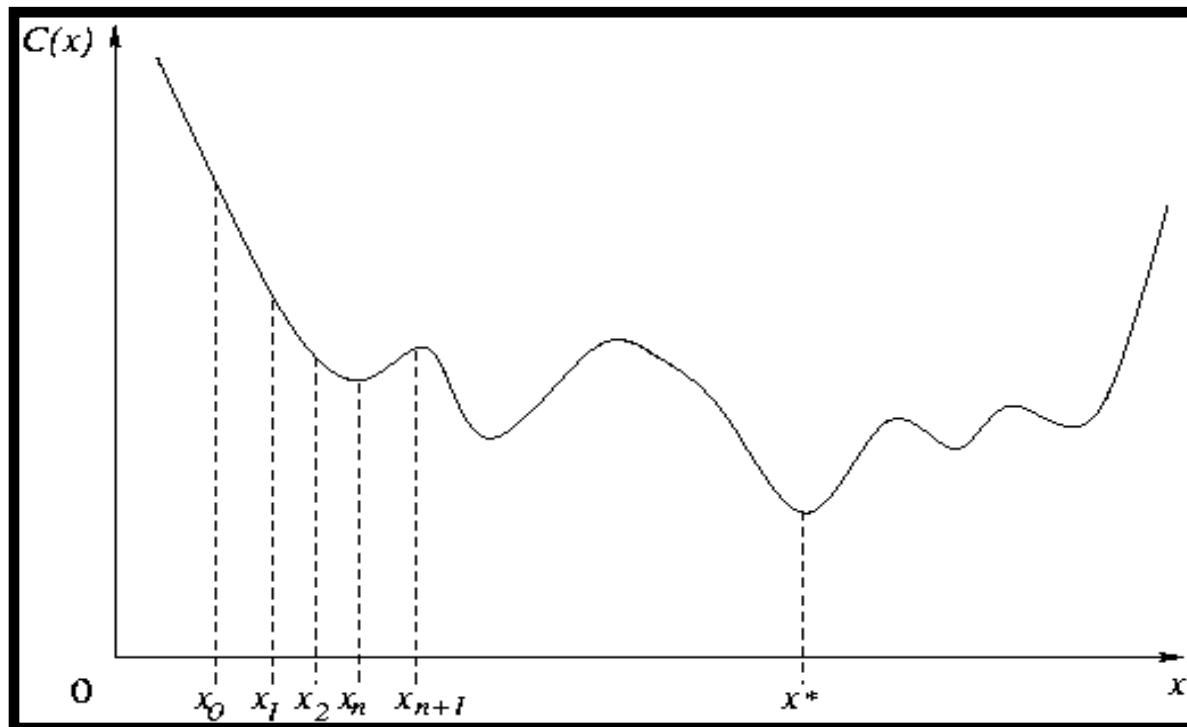
# Problème d'optimisation

- $\Omega$  : espace des configurations  
ou espace de recherche
- $C$  : fonction de coût (ou objectif ou fitness)
- Trouver une configuration  $x^* \in \Omega$  de coût minimal

$$C(x^*) = \min \left\{ C(x) \mid x \in \Omega \right\}$$

# Piège des minima locaux

- But: trouver une stratégie pour pouvoir sortir **des minima locaux**



# Principe du recuit

- Initialement le métal est porté à haute température
- Puis refroidissement progressif
  - à haute température
    - atomes très agités (configurations atomiques équiprobables)
  - à basse température
    - atomes organisés en une structure atomique parfaite (configuration proche de l'état d'énergie minimale)
- Contrainte
  - Refroidissement lent
    - Si le refroidissement est trop rapide, il y a un risque de rester bloqué dans un minimum local (configuration sous-optimale)

# Le recuit simulé

- Analogie problème d'optimisation / système physique

Problème d'optimisation	Système physique
fonction de coût / objectif $C(x)$	énergie libre $E(X)$
variables du problème	" coordonnées " des atomes
trouver une "bonne" configuration	trouver un état de basse énergie

- Algorithme du recuit simulé (Kirkpatrick & al. - 1983)

# Le recuit simulé

## ■ Algorithme

- Le paramètre **température** (agitation thermique) autorise avec une certaine probabilité le choix de configurations d'énergie plus élevée  
    ⇒ capacité d'éviter les minima locaux
- Suite de configurations dont le choix et l'acceptation dépendent de la fonction objectif et de la température
- Procédure de refroidissement qui donne la décroissance de la température au cours du temps

# Le recuit simulé

## ■ Algorithme du recuit simulé (version Metropolis)

$T \leftarrow T_0$  -- Température initiale

$X \leftarrow X_0$  -- Configuration initiale

**répéter**

**répéter**

Tirer aléatoirement  $Y \in \text{Voisinage}(X)$

**si**  $\Delta E = E(Y) - E(X) < 0$  ou  $\exp(-\Delta E / T) > \mu$ ,  $\mu \in [0;1]$  aléatoire

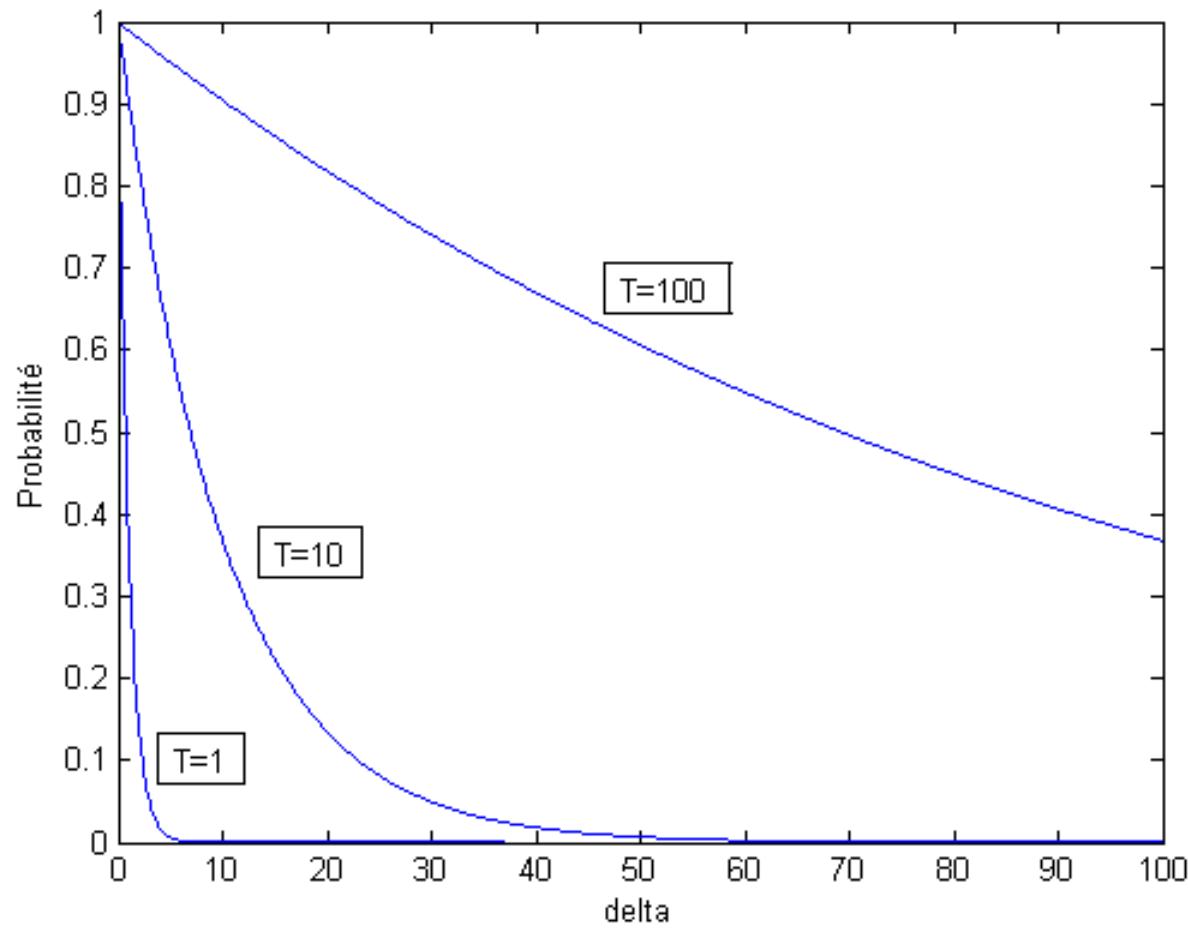
**alors**  $X \leftarrow Y$

**jusqu'à** fin palier

$T \leftarrow g(T)$  -- refroidissement :  $g$  strictement décroissante

**jusqu'à** critère d'arrêt vérifié

# Probabilité d'acceptation des sauts positifs



# Le recuit simulé

- Choix des paramètres (en pratique)
  - $T_0$  est choisie à l'issue d'expérimentations ou tests
  - La longueur d'un palier est également fixée empiriquement
  - Schéma de température exponentiel
$$T_n = T_0 \cdot \alpha^n, \text{ avec } n \in \{0,1,\dots\} \text{ et } 0 < \alpha < 1$$
  - Critères d'arrêt possibles
    - pourcentage de configurations acceptées en dessous d'un seuil fixé
    - variation de l'énergie trop faible
    - choix d'une température minimale

# Le recuit simulé

- Résumé du choix des paramètres
  - Loi de décroissance de la température
    - Baisse de température entre deux paliers pas être trop importante  
En théorie :  $T_n = T_0 / \ln n$   
En pratique :  $T_n = T_0 \cdot \alpha^n$ , avec  $n \in \{0,1,\dots\}$  et  $0.9 < \alpha < 1$
  - Critères d'arrêt possibles
    - En pratique
      - Le pourcentage de configurations acceptées descend en dessous d'un seuil fixé
      - Variation de l'énergie trop faible
      - Choix d'une température minimale
- Remarque
  - Voyageur de commerce
    - Recherche de la solution exacte  $\Rightarrow$  temps de calcul exponentiel
    - Recherche d'une solution approchée à 2%  $\Rightarrow$  temps de calcul en  $O(N^3)$

Nb. de variables  
du problème



# Le recuit simulé

## ■ Avantages

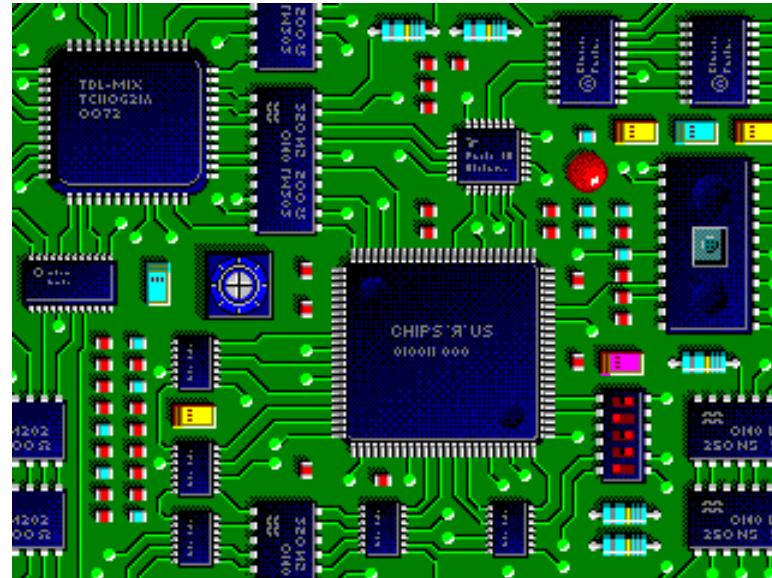
- Solutions de bonnes qualités
- Méthode générale applicable à tout problème d'optimisation
  - Fonction à optimiser évaluable (mieux si variations évaluables)
  - Notion de voisins garantissant la connexité
- Facile à programmer
- Nouvelles contraintes incorporables à tout moment  
(e.g. problème des horaires)

## ■ Inconvénients

- Temps de calcul

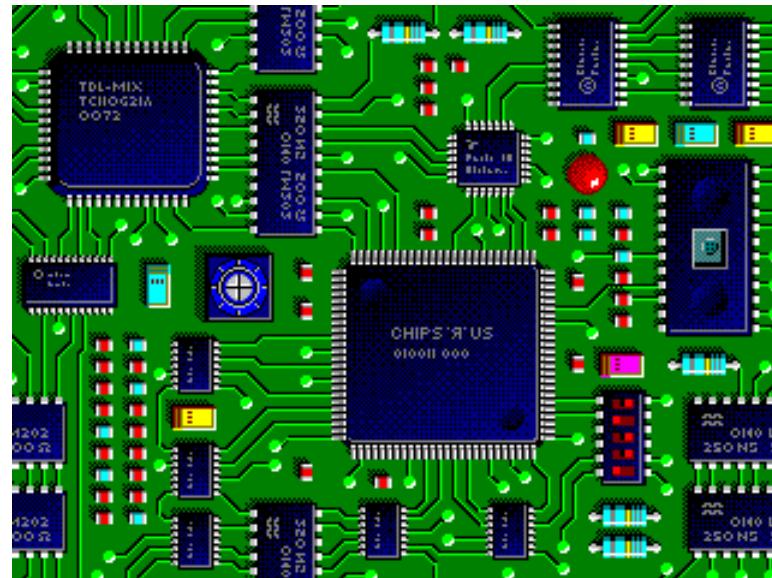
# Le recuit simulé: application

- Placement de composants électroniques
  - Fonction à optimiser?
  - Comment formuler le problème?
  - Quel algorithme?



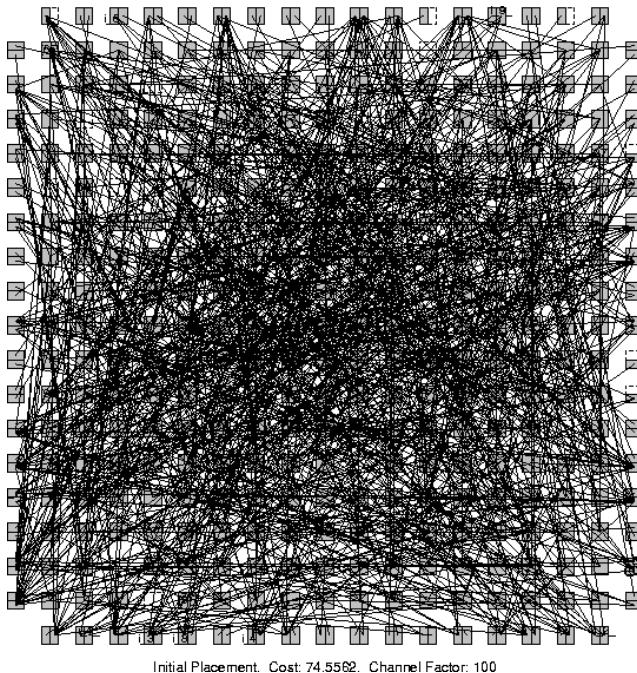
# Le recuit simulé: application

- Placement de composants électroniques
  - Modules → sommets d'un graphe
  - Interconnexions → arêtes d'un graphe
  - Coût de l'arrangement
    - longueur totale des fils + aire totale occupée

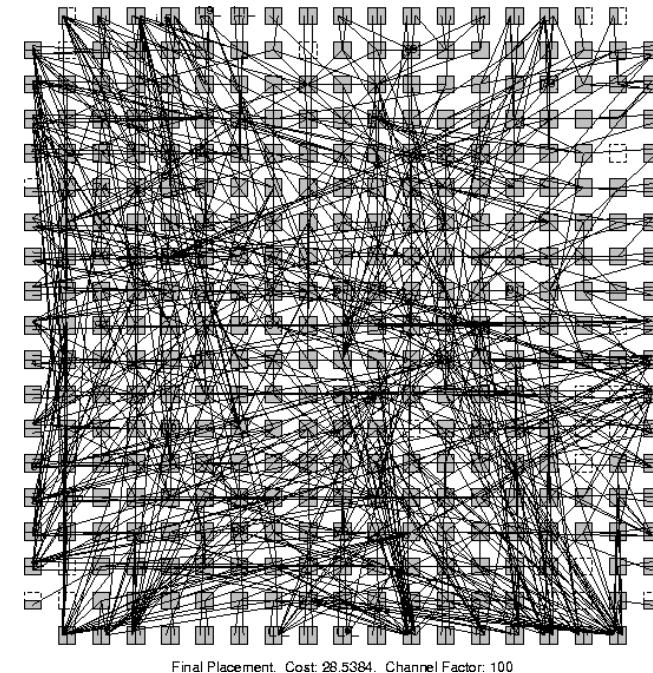


# Placement sur une FPGA

Placement initial  
aléatoire



Placement final



- Les emplacements possibles sont fixes ici
- Donc pas d'optimisation de l'aire totale



# Optimisation Algorithmes évolutionnaires

hepia HES-SO

Guido Bologna

Michel Vinckenbosch

# Algorithmes évolutionnaires

-  Informatique Bio-inspirée  $\neq$  Bioinformatique
  - Fondements biologiques
    - C. Darwin (1809-1882) A. R. Wallace (1823-1913): sélection naturelle
    - G. Mendel (1822-1884) : hérédité des caractères « génétiques »
    - T. H. Morgan (1866–1945) : théorie chromosomique
  - Algorithmes génétiques *Genetic Algorithms*
    - GA J. Holland (1962, 1975, Ann Arbor)
  - Programmation génétique *Genetic Programming*
    - GP J. Koza (1989, Palo Alto)
  - Stratégies d'évolution *Evolution Strategies*
    - ES I. Rechenberg et H.-P. Schwefel (1965, Berlin)
    - codage réel, mutation gaussienne
  - Programmation évolutionnaire *Evolutionnary Programming*
    - EP L. Fogel (1962, San Diego)
    - pas de croisement, sélection par tournoi

# Bioinformatique

## ■ Analyse de séquence

- Séquences de génome, de transcriptome ou de protéome
- Alignement de séquences (BLAST)
- Séquençage et utilisation de puces à ADN

## ■ Modélisation moléculaire

- Structure 3D des protéines: conformation, sites actifs d'une enzyme, mécanismes, cibles moléculaires possibles pour cette enzyme
- Structure 3D d'acides nucléiques (ARN et ADN)
- Dynamique moléculaire

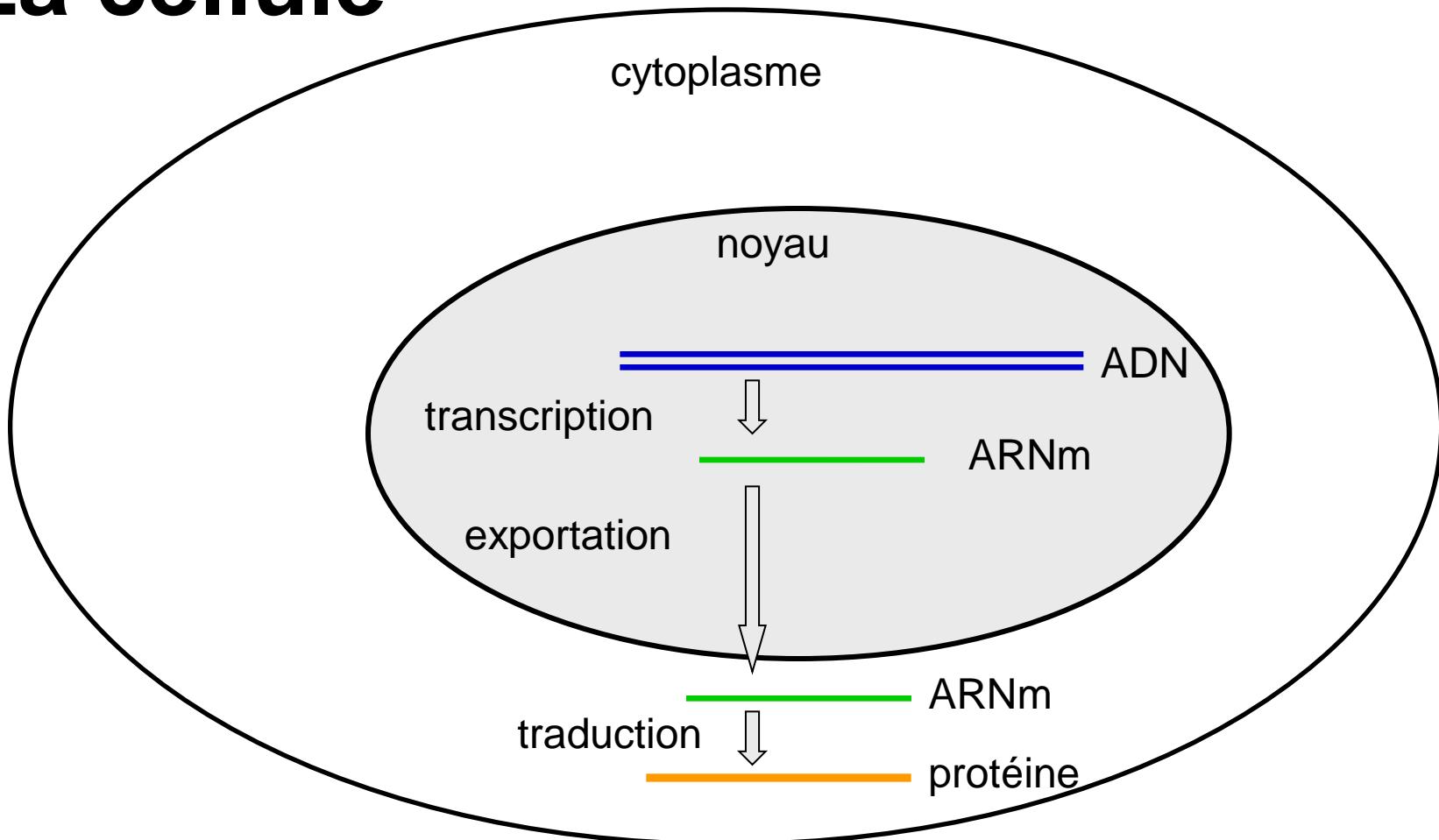
## ■ Arbres phylogénétiques

- Distance génétique (nombre de mutations séparant les gènes de 2 espèces)
- Arbres: hiérarchie des espèces selon leur proximité

## ■ Modélisation de population

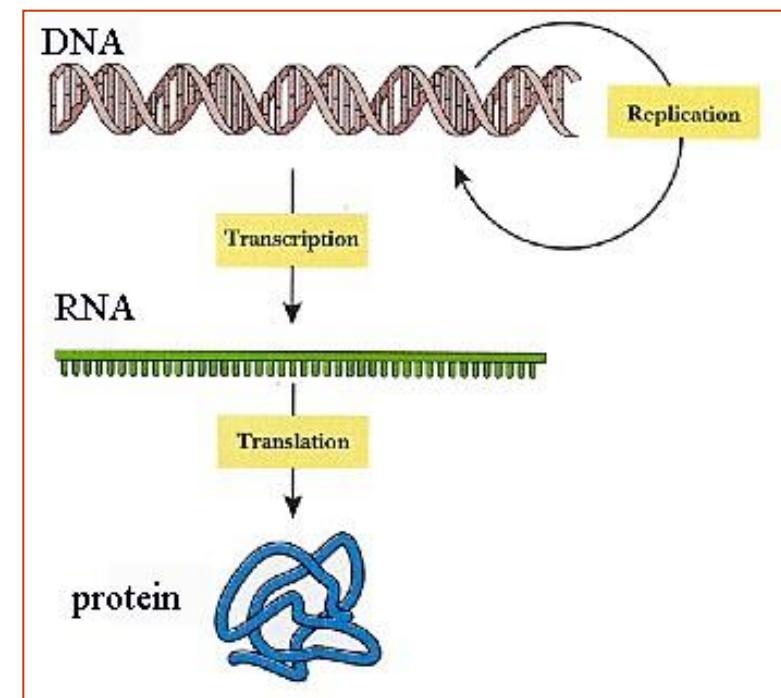
# Notions de biologie

## La cellule

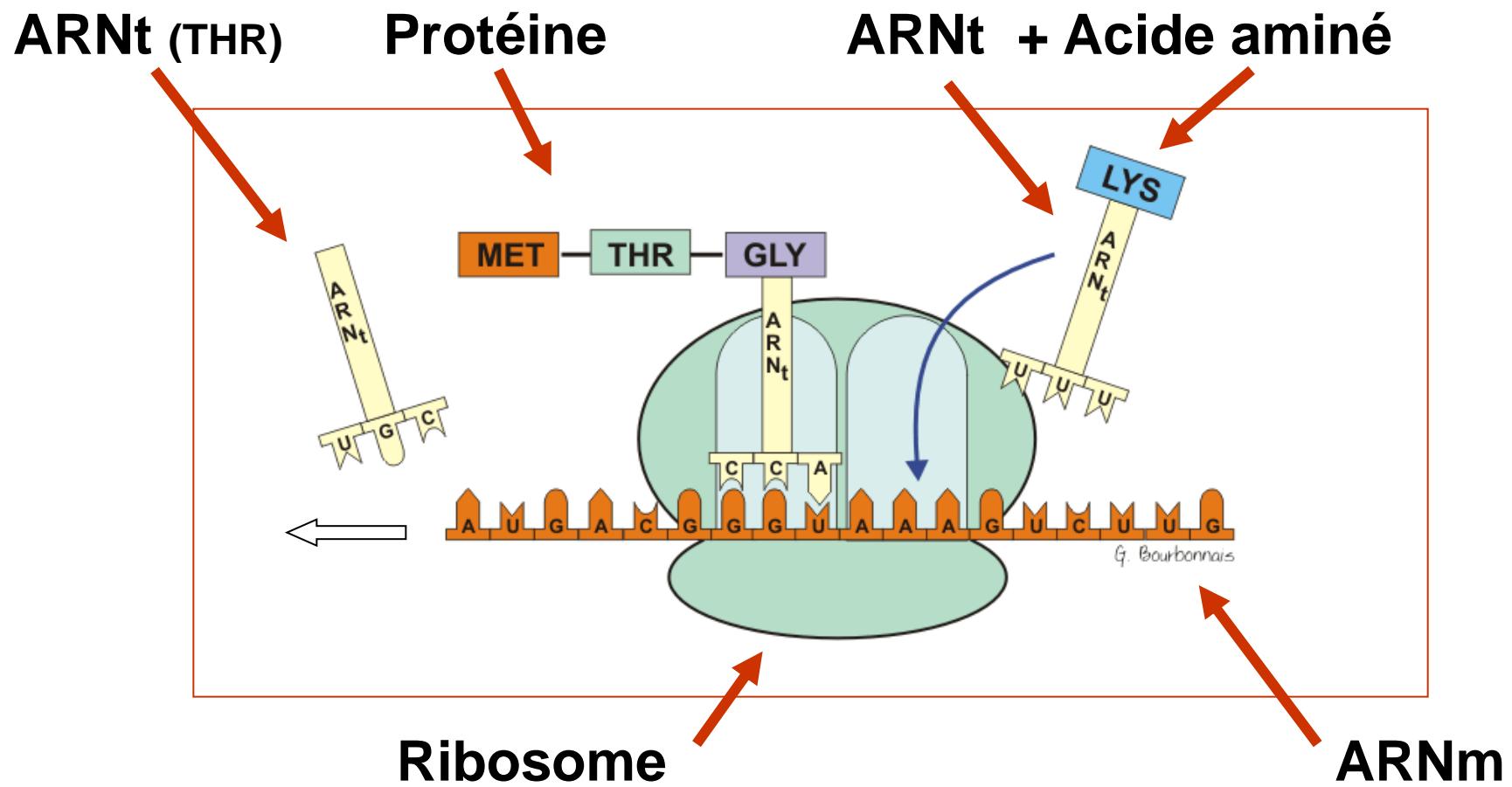


# Synthèse des protéines

- Information génétique: **ADN** (Acide DésoxiriboNucléique)
- L'ADN ne sort pas du noyau. L'information passe au cytoplasme sous forme de copie **ARN** (Acide RiboNucléique)
- Synthèse des protéines dans le cytoplasme au niveau des ribosomes (traducteurs)



# Mécanisme de la traduction



# Codage

## 3 nucléotides => 1 acide aminé

- Un ARNt ne transporte pas n'importe quel acide aminé: ça dépend de l'anticodon
  - ARNt **A-A-A** transporte toujours l'acide aminé **PHE**
  - ARNt **G-A-U** transporte toujours l'acide aminé **LEU**
- Note
  - un gène peut coder la synthèse d'une protéine (ARNm)
  - un gène peut coder la synthèse d'un ARNr (ARN Ribosomal) ou d'un ARNt (ces gènes existent en des milliers de copies dans le génome)
  - gène = brin d'ADN qui est copié en ARN

# Code génétique (partiel)

UUU	phénylalanine PHE	UCU	sérine SER
UUC		UCC	
UUA	leucine LEU	UCA	
UUG		UCG	
CUU	leucine LEU	CCU	proline PRO
CUC		CCC	
CUA		CCA	
CUG		CCG	

- code **dégénéré**: un acide aminé => plusieurs codons
- code **univoque**: un codon => un seul acide aminé
- code **universel** (grande majorité du vivant)
- UAA et UAG: codons **STOP**

# Principe général de l'évolution darwinienne

- Le contexte de l'évolution est une **population** (d'organismes, d'objets, d'agents ...) qui survivent un temps limité puis meurent. Certains produisent une **descendance** pour les générations suivantes. Les '**mieux adaptés**' tendent à se reproduire plus
- Sur plusieurs **générations**, la composition de la population change sans qu'aucun individu ne change au cours de sa vie. Lors des générations successives, « l'espèce » évolue et s'adapte en quelque sorte aux conditions.



# Principe général de l'évolution

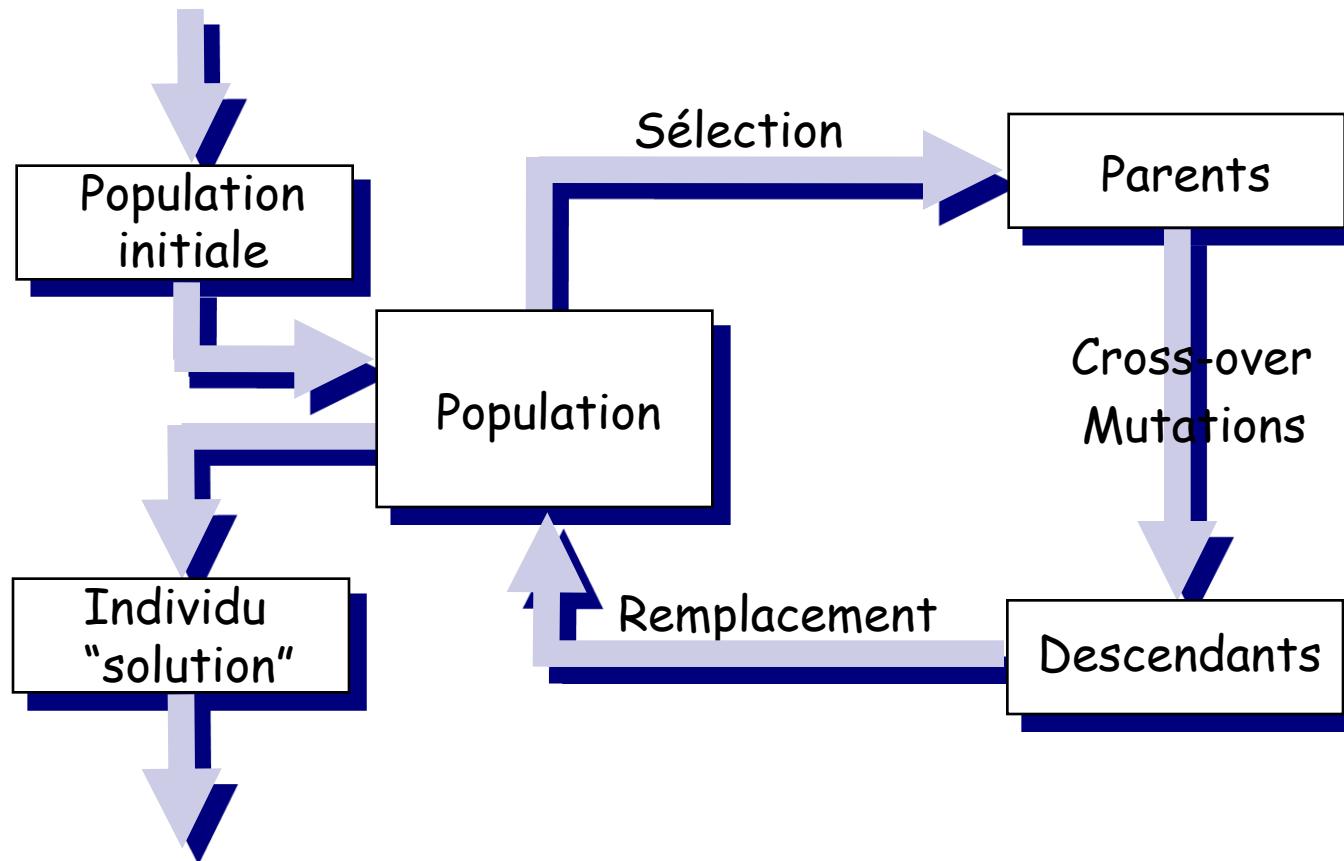
- Hérédité
  - Les descendants sont **similaires** aux parents
- Variabilité
  - Des changements aléatoires « perturbent » l'hérédité
  - Des recombinaisons aléatoires augmentent la diversité
- Sélection
  - Les organismes les plus adaptés à leur environnement sont susceptibles d'avoir plus de descendants

*Ces trois processus sont réunis au sein d'une boucle générationnelle ...*

# Algorithmes Génétiques: méthode d'exploration aléatoire bio-inspirée

- Sur le plan algorithmique, les AGs sont caractérisés par
  - Codage des solutions potentielles (génotypes des individus)
  - Exploration multiple (population)
  - La fonction à optimiser permet l'évaluation des individus (phénotype, performance ou *fitness*)
  - Règles de transition probabilistes (mécanismes de sélection / reproduction / mutation).
- On cherche une bonne solution à partir d'une population de solutions candidates
  - Tant qu'une bonne solution n'est pas trouvée, la population est soumise à une évolution dirigée

# Principe du cycle d'évolution



# Algorithme génétique

créer population initiale

**faire**

évaluer tous les individus (avec fct fitness)

**faire**

sélectionner parents

créer descendants par recombinaison

faire muter descendants

**jusqu'à** taille population

remplacer tous les individus par les descendants

**jusqu'à** critère d'arrêt

# Représentation et initialisation

## Exemple

- Un individu est codé par une chaîne de bits (génotype)
- Chaque chaîne code une solution candidate
- La population initiale est aléatoire

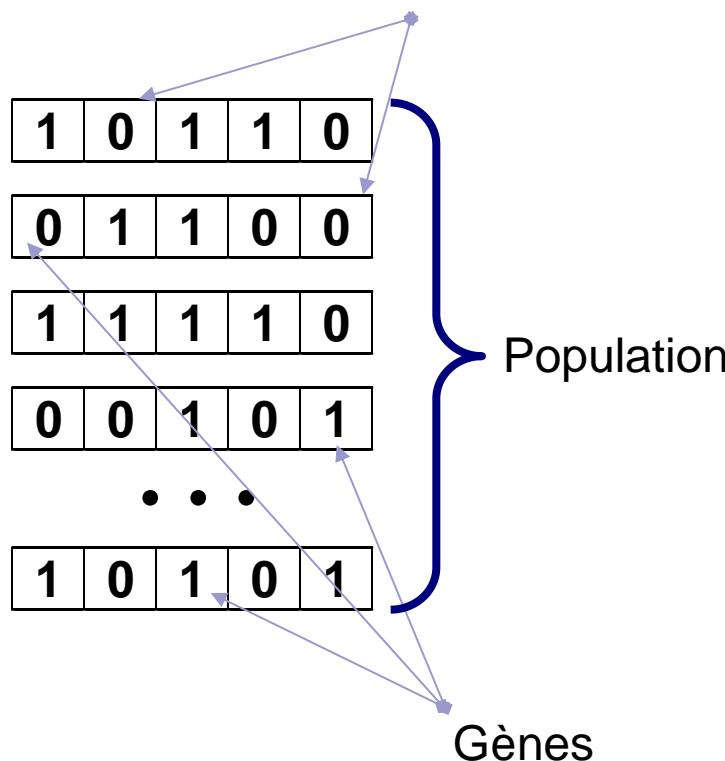
A diagram illustrating a population of individuals. On the left, five binary strings are shown, each consisting of five bits. The first four strings are grouped by a blue curly brace. Three dots below the fourth string indicate that there are more individuals in the population. The fifth string is shown separately below the dots.

1	0	1	1	0
0	1	1	0	0
1	1	1	1	0
0	0	1	0	1
• • •				
1	0	1	0	1

N individus générés aléatoirement  
(population initiale)

# Terminologie

Chromosomes, Individus ...

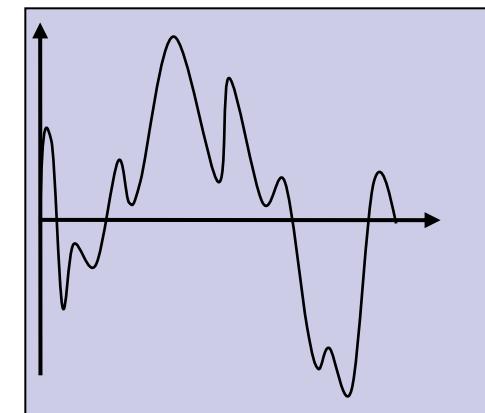


Génotype : configuration de la chaîne binaire

Phénotype : ce que le génotype représente « réellement »

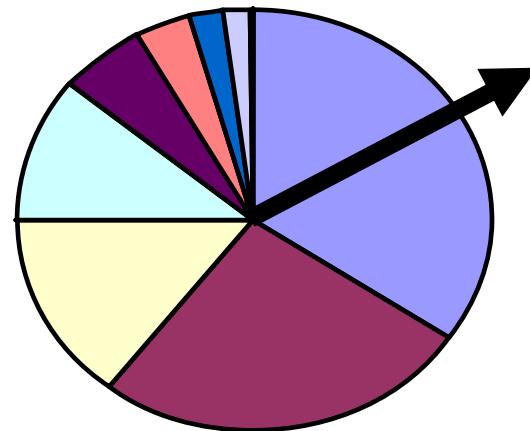
$$\begin{array}{ccccc} 0 & 1 & 1 & 0 & 0 \end{array} = 2$$

*Fitness Landscape* (paysage évolutif) : fonction performance sur l'espace de recherche



# Sélection proportionnelle

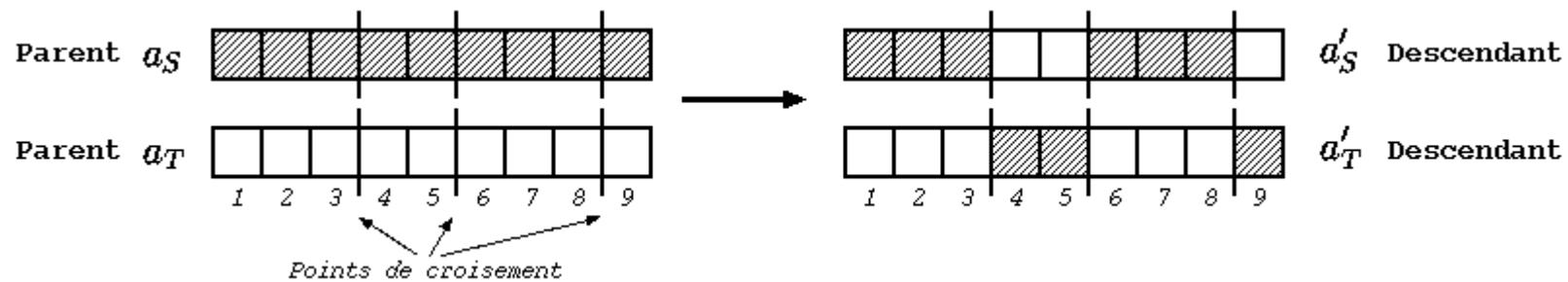
- Un individu  $i$  de fitness  $f_i$  a une probabilité  $f_i / \sum_j f_j$  d'être sélectionné comme parent
- Principe de la roulette biaisée
  - Chaque individu reçoit une portion de roulette proportionnelle à sa fitness



■ Individu 1
■ Individu 2
■ Individu 3
■ Individu 4
■ Individu 5
■ Individu 6
■ Individu 7
■ Individu 8
■ Individu 9
■ Individu 10

# AG : génération

- Sélection
- Croisement / recombinaison (*crossover*)
  - choix des individus formant la nouvelle population

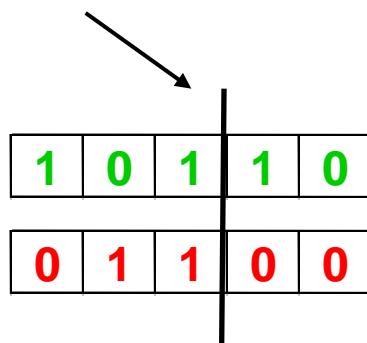


- Mutation
- modification aléatoire d'un individu



# Croisement / mutation

point de croisement



croisement

1	0	1	0	0
0	1	1	1	0

1	0	1	0	0
---	---	---	---	---

mutation

0	0	1	0	0
---	---	---	---	---

# Étapes de mise en œuvre d'un AG

- Choix « algorithmiques »
  - Représentation des individus (codage du génotype),
  - Génération de la population initiale
  - Modalités de reproduction (opérateurs d'évolution)
  - Choix du mode de sélection des individus (opérateurs de sélection)
- Choix « paramétriques »
  - Taille de la population, taux de reproduction, de survie, de croisement, de mutation

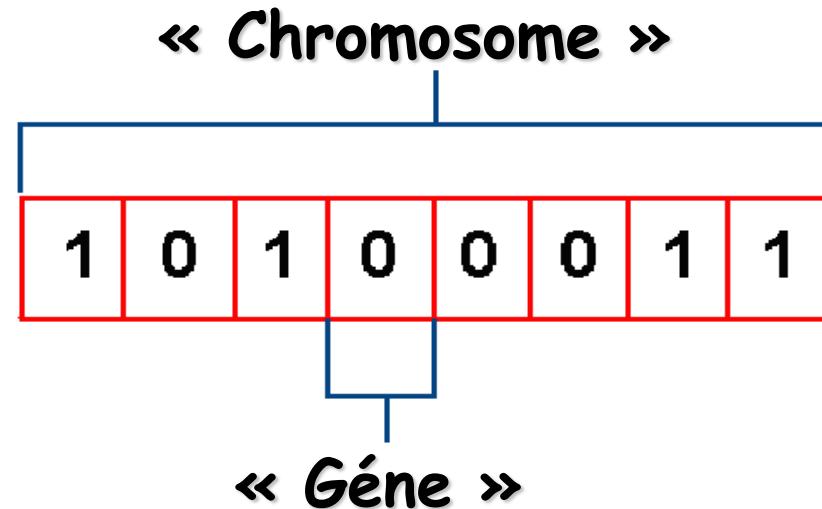
# Choix de la représentation (génotype)

- Comment représenter un individu (phénotype ou solution candidate) par un génotype?
- Nombreuses possibilités, **mais**, le codage doit être adapté au problème posé
- Choix de représentation dépend des autres étapes
  - génotype sera converti en phénotype et évalué
  - génotype manipulé par des opérateurs génétiques (mutation, cross-over, ...)

⇒ *Un codage inadapté peut compromettre l'évolution ...*

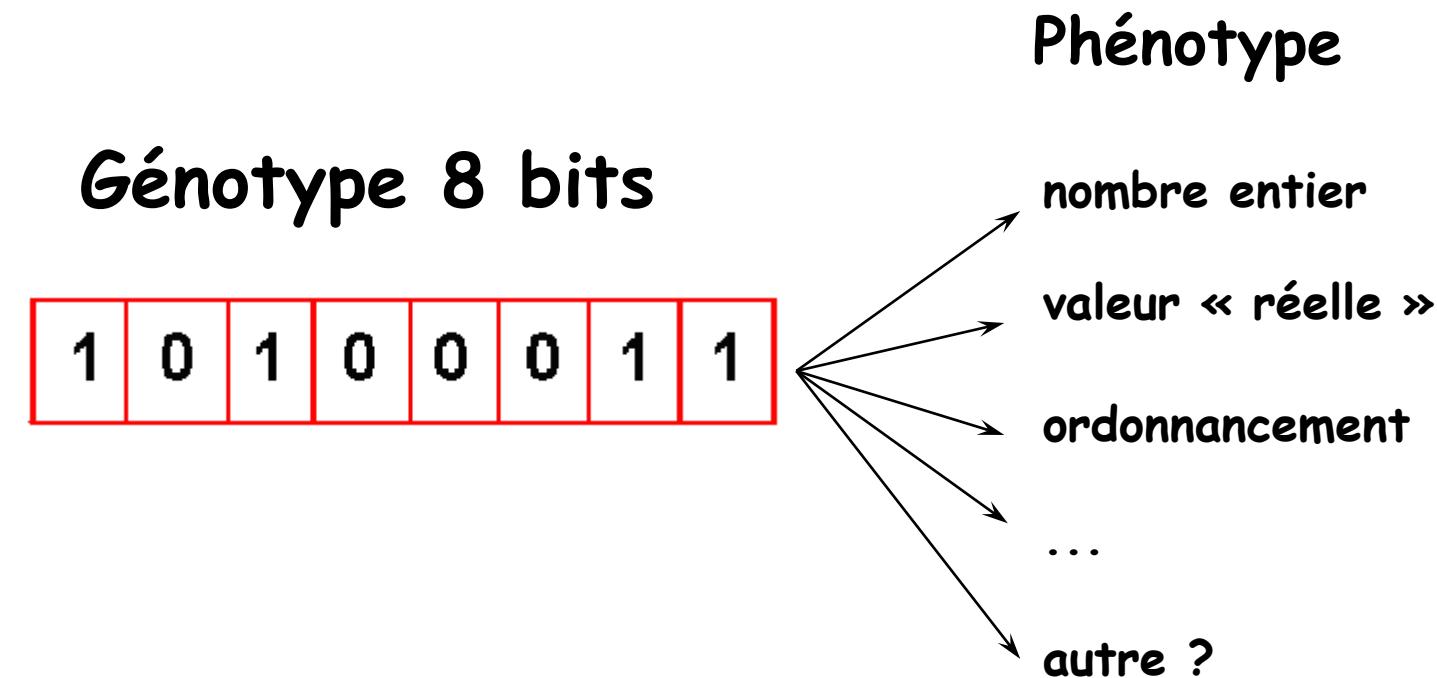
# Représentations discrètes

- Un individu peut être représenté par un ensemble de valeurs discrètes (codage binaire, entiers, ...)
- **Semble** très proche de la biologie (ACTG) mais ...
- Exemple : représentation binaire



# Représentations binaires

- Un génotype binaire peut conduire à une grande variété de phénotypes :



# Codage de « réels » à partir d'une représentation binaire

- Le codage de valeurs « réelles » est trivial si on connaît la dynamique des paramètres et la précision souhaitée
  - Codage d'un « réel » entre -0.5 et 0.5 avec une précision de 0.001
  - Codage sur 10 bits ([0-1023]) et normalisation  $((x/1023)-0.5)$
- Le codage binaire pose le problème des « falaises de hamming » (*hamming cliffs*)
  - Certaines transitions sont plus probables que d'autres (car le processus de mutation est ponctuel)
  - Cette probabilité n'est pas liée à la proximité des phénotypes mais à la proximité des génotypes (distance de hamming)
    - ⇒ *Un codage inadapté peut compromettre l'évolution ...*

# Code de Gray (*Gray Coding*)

- Codage qui permet à 2 entiers adjacents d'être distants d'une seule mutation (l'inverse n'est pas vrai !)
  - Pas de falaises de Hamming
- Recette de traduction entier binaire => Gray
  - commencer à gauche copier le 1er bit
  - ensuite écrire 1 si le bit change et 0 sinon

Exemple, nombre sur 3 bits :

Binaire	Valeur	Gray
000	0	000
001	1	001
010	2	011
011	3	010
100	4	110
101	5	111
110	6	101
111	7	100

# Code de Gray

- Frank Gray, Bell-Labs 1953 pour l'électromécanique, éviter des commandes transitoires pour des ensembles de relais
- Deux entiers adjacents ne diffèrent que de un bit
  - Deux phénotypes proches auront des génotypes proches
- La conversion bit par bit du binaire au code Gray
  - Chaine binaire :  $b_1, b_2, \dots, b_n$
  - Chaine gray :  $g_1, g_2, \dots, g_n = b_1, b_1 \oplus b_2, \dots, b_{n-1} \oplus b_n$

Entiers :	0	1	2	3	4	5	6	7
Binaire :	000	001	010	011	100	101	110	111
Gray :	000	001	011	010	110	111	101	100

# Cas particulier: représentation ordonnée

- Un individu est représenté par une liste de positions avec permutations
- Application : ordonnancement / séquencement
- Exemple : Le voyageur de commerce
  - Chaque ville reçoit un numéro d'ordre de 1 à  $n$
  - Les solutions sont représentées par un ordre (ex : 5, 4, 2, 1, 3).
    - ⇒ *Les représentations ordonnées doivent être utilisées avec des opérateurs génétiques spécifiques assurant la validité des individus*

# Représentation à valeurs réelles

- Si la solution recherchée est exprimée sous la forme d'une liste de valeurs réelles, le codage le plus naturel est d'exprimer le génome sous la forme d'une liste de réels ! (abstraction faite du codage binaire)
- Cas très courant couvrant un grand nombre d'applications (optimisation paramétrique)
  - ⇒ *La manipulation individuelle, par des opérateurs génétiques, des bits d'un codage flottant n'a aucun sens ...*

# Représentation à valeurs réelles

- Un individu  $X$  est un vecteur à valeurs réelles :

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_i \in R$$

- La fonction de fitness  $f$  permet d'exprimer la qualité d'un individu (fonction de  $R^n$  dans  $R$ )

$$f : R^n \rightarrow R$$

# Initialisation de la population

- *Si possible*: tirage aléatoire uniforme dans l'espace de recherche
  - Codage binaire : 0 ou 1 avec une probabilité de 0.5
  - Codage à valeurs réelles : tirage uniforme dans un intervalle donné (applicable uniquement dans un ensemble borné)
  - Attention : un tirage uniforme des génotypes ne garantit pas toujours un tirage uniforme des phénotypes

# Evaluation des individus

- Pour les applications réelles, cette étape est - de loin - la plus coûteuse
  - Difficulté de conception, temps de calcul, prix ...
  - Ne pas évaluer de nouveau les individus identiques ou non modifiés lors de leur reproduction
- Plusieurs méthodes en fonction du problème :
  - Évaluation directe (fonction de fitness),
  - Simulation (boite noire),
  - Processus externe (robot, tests en situation, ...)
    - ⇒ *Pour accélérer le processus il est possible d'utiliser une évaluation approchée ... au début du processus d'évolution ...*

# Cas particuliers de l'évaluation

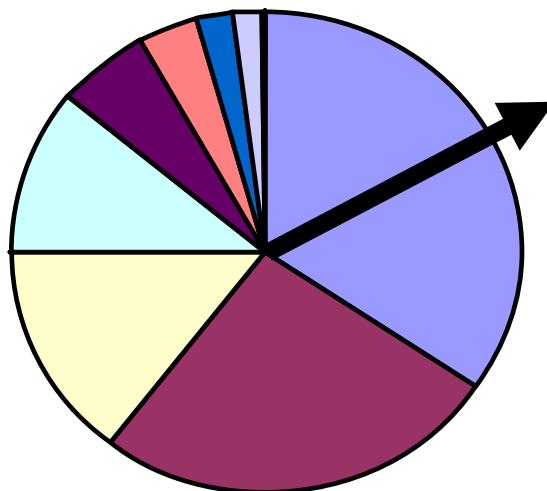
- Satisfaction de contraintes : que faire lorsque le phénotype rompt des contraintes du problème ?
  - Pénaliser l'individu (impact sur le fitness)
  - Utiliser des opérateurs génétiques spécifiques

# Stratégie de sélection « naturelle »

- But de la sélection naturelle :
  - Assurer que les meilleurs individus aient plus de chance d'avoir une descendance que les autres.
- La « pression de sélection » doit guider l'exploration
- Les moins bons individus doivent conserver une chance de se reproduire.
  - ils peuvent contenir des gènes intéressants.
  - => la pression de sélection doit préserver une part de bio-diversité.

# Sélection proportionnelle

- Un individu  $i$  de fitness  $f_i$  a  $n \cdot (f_i / \sum_j f_j)$  chances de se reproduire ( $n$  étant le nombre d'individus sélectionnés).
- Principe de la roulette biaisée.
  - Chaque individu reçoit une portion de roulette égale à sa fitness
  - La roulette est tirée  $n$  fois



- Individu 1
- Individu 2
- Individu 3
- Individu 4
- Individu 5
- Individu 6
- Individu 7
- Individu 8
- Individu 9
- Individu 10

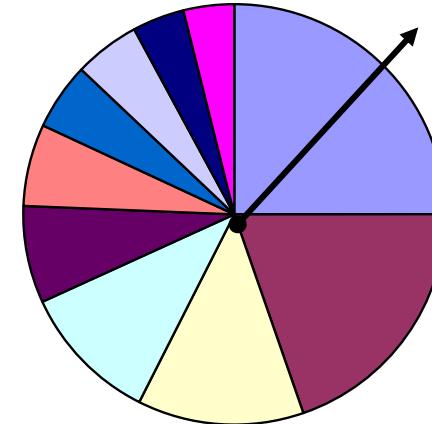
# Sélection proportionnelle

- Désavantages de la sélection proportionnelle :
  - Réduction rapide de la diversité
  - Risque de convergence prématuée (les individus les mieux adaptés vont rapidement phagocytter l'ensemble de la population)
  - La pression de sélection reste faible lorsque les fitness sont très similaires
  - Plusieurs solutions pour éviter la convergence prématuée (vers un optimum local)
    - ⇒ *Normalisation de la fonction de fitness ...*

# Sigma Scaling

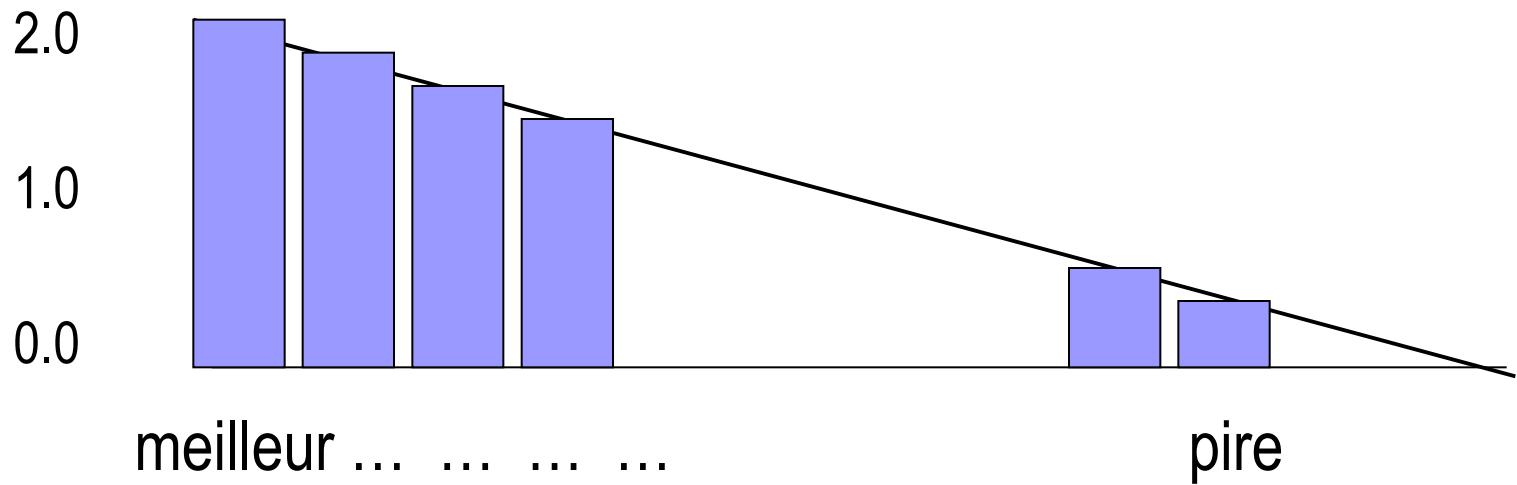
- But : éviter la perte de biodiversité si un individu est beaucoup mieux adapté que les autres
  - La grande majorité des descendants sont issus d'un seul individu
  - Risque de convergence prématuée
- Normalisation basée sur la variance (sigma scaling)
  - Uniformisation de la pression de sélection

$$\begin{cases} \text{Si } \sigma \neq 0 & Fit(x) = 1 + ((f(x) - Mean(f)) / 2 * \sigma) \\ \text{Sinon} & Fit(x) = 1 \end{cases}$$



# Sélection par le rang

- Sélection linéaire par le rang aligne la population selon le rang et donne une probabilité de sélection proportionnelle au rang

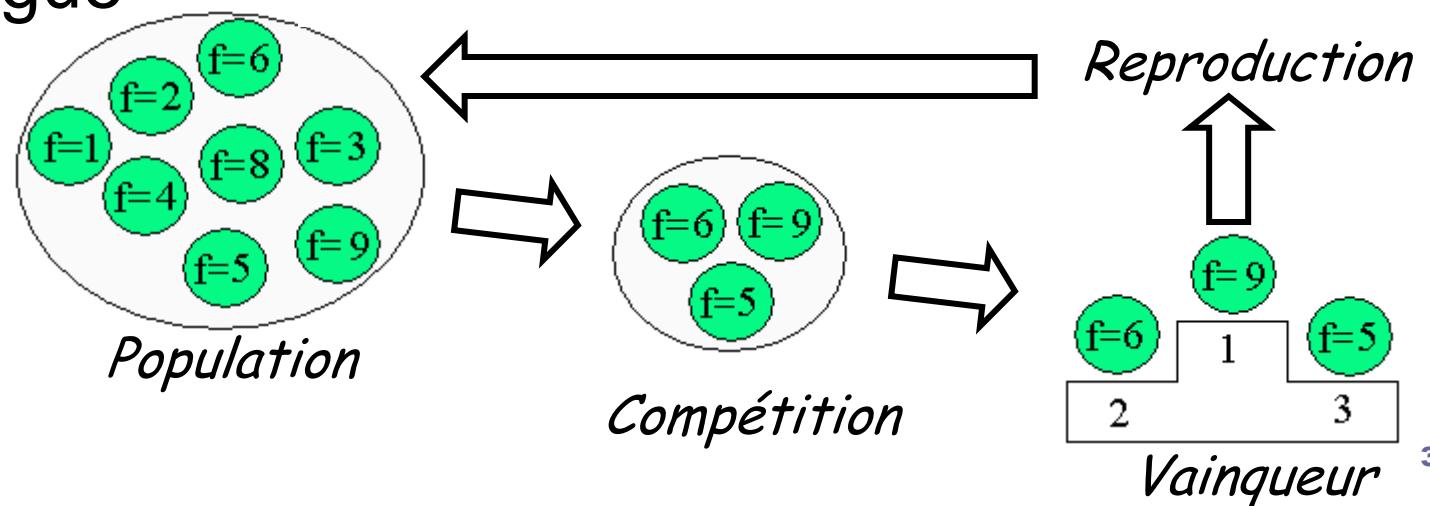


# Sélection par le rang

- La droite ne descend pas obligatoirement entre 2.0 et 0.0, par exemple entre 1.5 et 0.5
- Possibilité d'un rang non linéaire
- Plus couramment utilisé: rang linéaire avec une pente de 2.0 à 0.0
- Ainsi le **meilleur** se reproduit **2 fois** plus que le **médian**. Les solutions inférieures à la moyenne gardent une chance de se reproduire

# Sélection par compétition (tournament)

- Sélection basée sur un remplacement partiel de la population
  - Sélection aléatoire de  $k$  individus (sans remplacement),
  - Compétition entre les  $k$  individus,
  - Seul le meilleur se reproduit
- Très en vogue



# Stratégie de « remplacement » (mortalité)

- Outre le critère de sélection, la pression de sélection est fonction de la « mortalité » des individus
- En AG, la mortalité est généralement liée au « remplacement » (population constante)
- Deux approches :
  - Stratégies stochastiques
  - Remplacement déterministe
    - ⇒ *Élitisme : conservation systématique du (des) meilleur(s) individu(s) ...*

# Autres stratégies de remplacement

- Si  $N$  parents génèrent  $N$  enfants et sont tous remplacés, la sélection se fait uniquement sur le choix des parents
- Mélanger parents et enfants puis sélectionner pour le remplacement
- Générer plus d'enfants puis les sélectionner

# Intérêt de l'élitisme

- Comment assurer une croissance permanente du fitness ?
  - Élitisme : ré-introduire le ou les meilleur(s)
  - Préservation : « niche » pour le meilleur individu
- L'élitisme présente aussi un risque
  - Convergence prématuée due à la conservation d'un individu qui « pollue » en permanence la population

➔ *Intérêt principal de l'élitisme : éviter la frustration de l'utilisateur ...*

# Les opérateurs génétiques : la mutation

- Il peut y avoir un ou plusieurs opérateurs de mutation, mais :
  - un des opérateurs de mutation au moins doit permettre d'atteindre tous les points de l'espace de recherche
  - le taux de mutation est un facteur fondamental qui doit pouvoir être finement contrôlé
    - ⇒ *Les mutations doivent produire des chromosomes (i.e. des individus) valides (i.e. viables)*

# Mutation pour une représentation binaire

**Génotype**

**Avant**    **Après**

1 0 1 1 1 1 1	1 0 1 0 1 1 1
---------------	---------------

↑  
Gène mutant

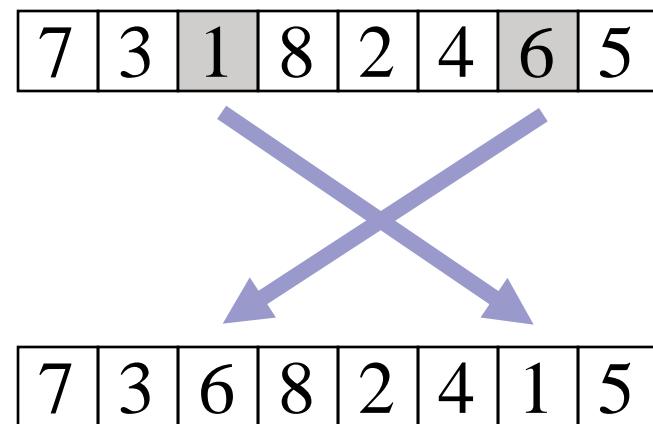
**Phénotype**

The diagram illustrates a mutation in an elephant's genotype. On the left, under 'Avant' (Before), is a sequence of seven binary digits: 1 0 1 1 1 1 1. An orange box encloses the first four digits. On the right, under 'Après' (After), is the same sequence with one change: 1 0 1 0 1 1 1. An orange box encloses the first five digits. An arrow points from the word 'Gène mutant' (Mutant gene) to the fourth digit from the left ('0'). To the right of the mutated genotype are two illustrations of elephants. The top one is grey, representing the original phenotype. The bottom one is black, representing the mutated phenotype. This visualizes how a single genetic change can lead to a visible physical difference.

→ Les mutations apparaissent avec une probabilité  $p_m$  pour chaque gène (taux de mutation).

# Le cas des représentations ordonnées

- Échanger deux gènes choisis aléatoirement (*gene swap*)



# Le cas des génomes à valeurs réelles

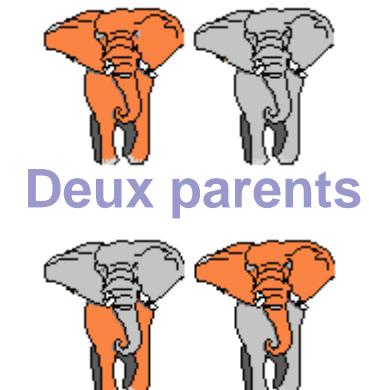
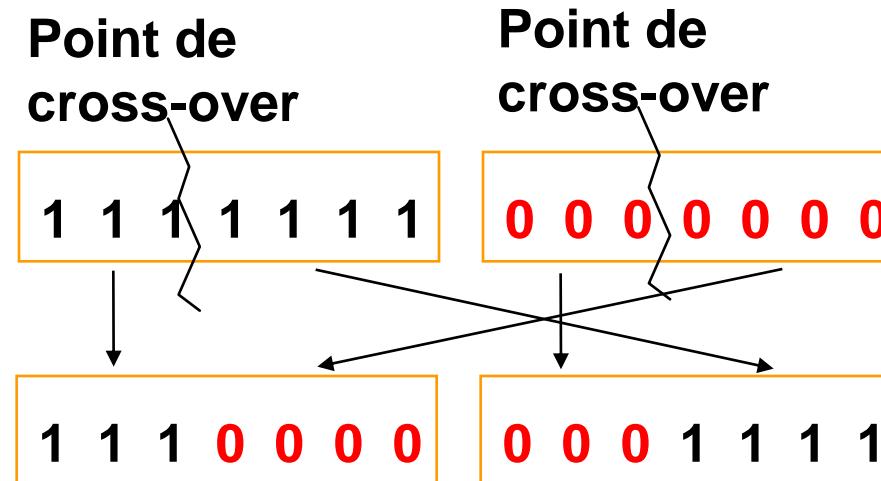
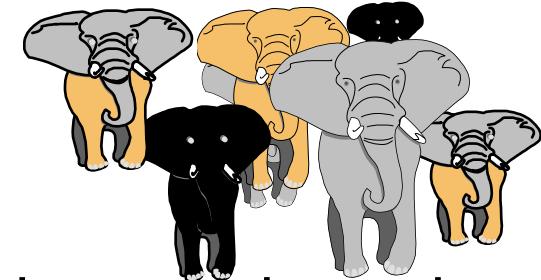
- Perturbation des gènes par addition d'un bruit aléatoire
- En général on utilise un bruit Gaussien obtenu par une loi normale  $N(0, \sigma)$ 
  - $0$  est la valeur moyenne du bruit
  - $\sigma$  est l'écart type
- Soit, pour chaque paramètre :  $x'_i = x_i + N(0, \sigma_i)$
- L'utilisation d'un bruit à distribution uniforme est possible

# Les opérateurs génétiques: le cross-over

- Il peut y avoir un ou plusieurs opérateurs de recombinaison (cross-over), mais :
  - les descendants doivent hériter de chacun des parents (sans quoi il s'agit d'une simple mutation)
  - les opérateurs de recombinaison doivent tenir compte de la représentation du génotype et de la fonction de transfert génotype/phénotype
- Le cross-over est souvent appelé «Crossing-over» ou «recombinaison»
  - En particulier en biologie
    - ⇒ *Les recombinations doivent produire des chromosomes (i.e. des individus) valides (i.e. viables)*

# Le cas des représentations binaires

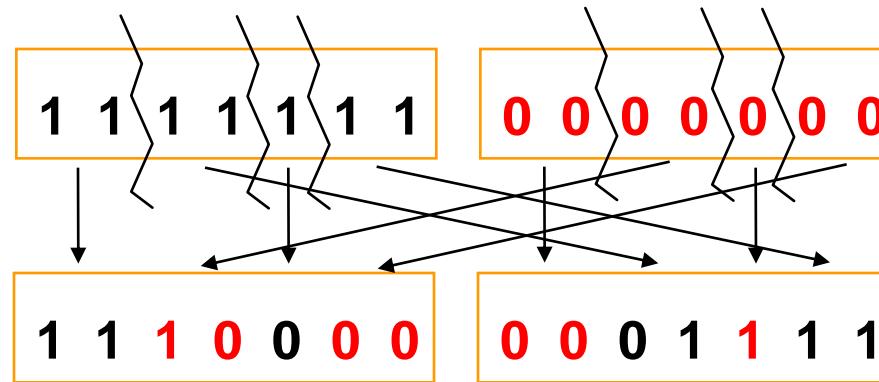
- Deux parents donnent deux enfants
  - Parfois un seul
- Chaque chromosome est découpé en deux parties qui sont recombinées pour former les descendants



# Variantes pour le cross-over discret

## ■ Cross-over multipoints

- $k$  points de cross-over tirés au hasard



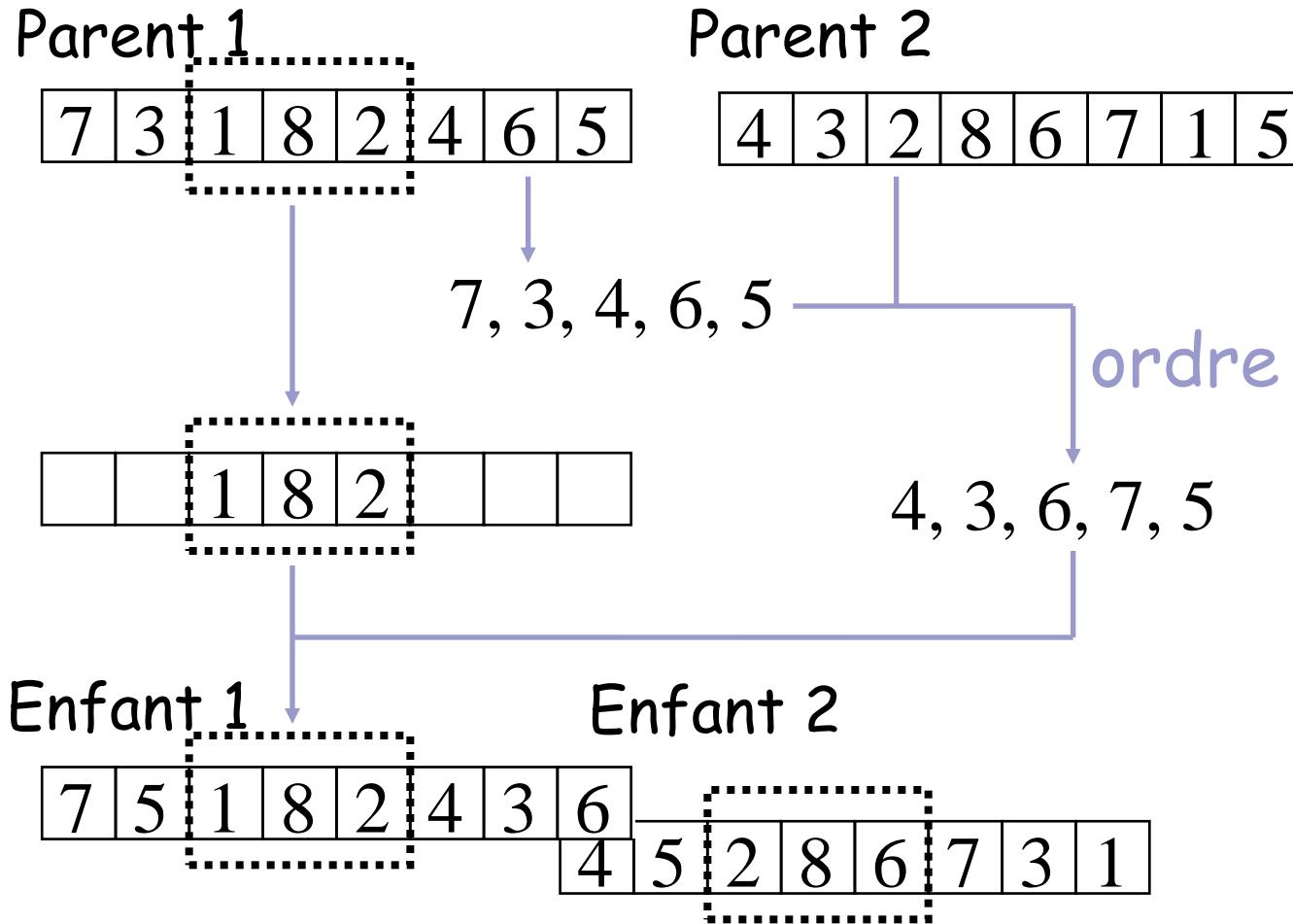
## ■ Cross-over uniforme

- Les gènes sont échangés suivant une probabilité  $p_c$

# Le cas des représentations ordonnées

- Choisir aléatoirement une partie du génome du premier parent et le recopier à l'identique
- Compléter le génome du descendant avec le deuxième parent
  - à partir du point suivant la partie recopiée
  - utiliser l'ordre des gènes du deuxième parent
  - balayage circulaire du chromosome
- Inverser le rôle des parents pour créer un deuxième descendant

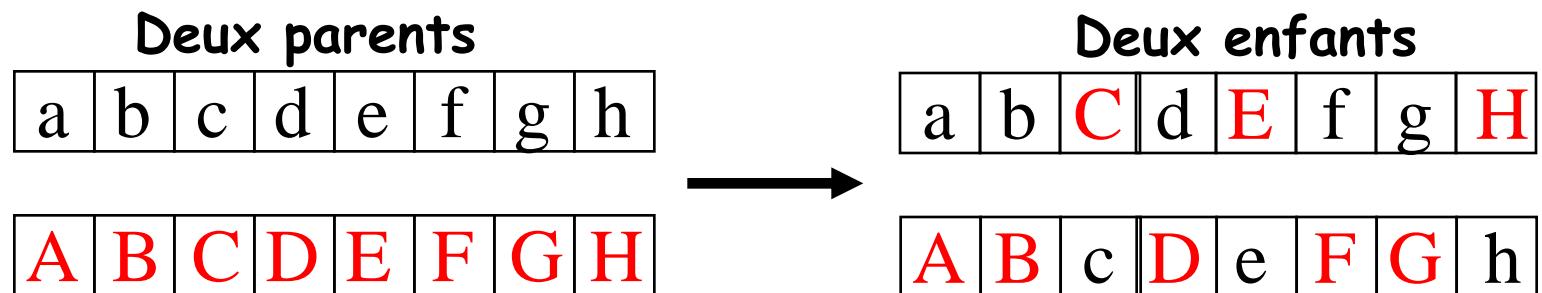
# Le cas des représentations ordonnées



# Le cas des génomes à valeur « réelle »

## ■ Recombinaison discrète :

- Recombinaison aléatoire des gènes à partir des deux parents
- Généralement pas de contraintes topologiques sur le génome



# Le cas des génomes à valeur « réelle »

- Recombinaison discrète :

Deux parents

a	b	c	d	e	f
---	---	---	---	---	---

A	B	C	D	E	F
---	---	---	---	---	---



Un enfant

(a+A)/2	(b+B)/2	(c+C)/2	(d+D)/2	(e+E)/2	(f+F)/2
---------	---------	---------	---------	---------	---------

# Intérêts relatifs de la mutation et du cross-over

## ■ Cross-over

- Faible part d'aléa, permet de ré-exprimer des caractères déjà présents dans la population,
- L'effet du cross-over diminue lorsqu'on se rapproche de la convergence,
- Opérateur d'**exploitation**

## ■ Mutations

- Indispensables pour éviter les maxima locaux,
- Opérateur d'**exploration** ...

☞ *La recombinaison est souvent plus difficile à mettre en œuvre ...*

# Intérêts relatifs de la mutation et du cross-over

- L'intérêt du cross-over varie beaucoup suivant les problèmes :
    - La fonction de fitness est-elle fiable ?
    - Les opérateurs de recombinaison ont-ils un sens compte-tenu du codage utilisé ?
- ⇒ *Il est possible de « mixer » mutation et cross-over en autorisant la parthénogénèse (recombinaison d'un individu avec lui-même) et la reproduction sexuée*

# Critère d'arrêt ...

- Difficile à déterminer puisque la qualité de la solution s'améliore de façon quasi-continue
  - Plusieurs « solutions »
- Le maximum est atteint
  - Suppose qu'il soit connu !
  - Il s'agit souvent d'un maximum *utile* ...
- Limite de temps de calcul
  - On ne peut pas toujours évaluer autant d'individus qu'on le souhaite
- Limite de patience de l'utilisateur
  - Lorsque rien n'évolue pendant plusieurs générations

# Choix paramétriques ...

- Outre les choix « algorithmiques », la mise en œuvre d'un algorithme génétique demande de fixer un certain nombre de paramètres
- $n$  : nombre d'individus dans la population
  - de l'ordre de la centaine (souvent fonction de la difficulté de calcul de la fonction de fitness)
- $T_m$  : taux de mutation
  - de l'ordre de 1/1000 (fonction de l'influence des différents gènes ou de l'ordre de 1 mutation par génome)
- $T_c$  : taux de cross-over
  - de l'ordre de 1/2 (fonction de l'intérêt/du sens du cross-over)

# Performance des algorithmes génétiques

- Du point de vue théorique :
  - Ne jamais tirer de conclusion à partie d'un test unique !
  - Les performances doivent être estimées statistiquement
  - Donc à partir d'un grand nombre d'essais

# Point clef de l'algorithme génétique

- Maintien de la diversité génétique au cours de l'évolution
    - Il faut maintenir des caractéristiques génétiques différentes dans la population (bio-diversité)
    - Lorsqu'on perd la diversité génétique, tous les individus deviennent semblables
    - Effet boule de neige
    - Convergence vers l'optimum local le plus proche
- ☞ *En théorie, les mutations permettent de continuer à explorer l'espace ... en pratique, la perte de diversité génétique est irréversible ...*

# Point clef de l'algorithmique génétique

## ■ Dilemme Exploration / Exploitation

- **Exploration** = tester les zones inconnues

Trop d'exploration revient à une marche aléatoire et compromet la convergence

- **Exploitation** = essayer d'améliorer le meilleur individu trouvé jusqu'ici

Trop d'exploitation revient à une recherche locale et conduit à une convergence vers un optimum local

⇒ *Dilemme exploration / exploitation*

# Avantages des AGs

- Bon rapport coût / résultat sur une grande classe de problèmes
- Parallélisme intrinsèque
- Robuste, tolérant aux fautes
- Applicable sans connaissance préalable du domaine d'application
- Simple à programmer
- Attention, « everything is problem dependent »
  - Les algorithmes génétiques ne sont pas **la** panacée universelle
- les AGs sont particulièrement adaptés lorsque le problème :
  - contient beaucoup de données / de paramètres
  - contient des paramètres interdépendants (problème complexe)
  - comporte des optimums locaux

# Inconvénients des AGs

- Pas de garantie de convergence en un temps fini
- Faiblesse des fondements théoriques et mathématiques
- Souvent gourmands en calcul donc lents (mais aisément parallélisables)
- Chaque individu doit être évalué, même les individus inadaptés (utilisation offline, sur simulateur)
- Produit toujours des individus inadaptés
  - ↳ *Le comportement de l'algorithme dépend d'un grand nombre de paramètres qui peuvent être difficiles à fixer*

*Réseaux de Neurones:*

- Modèles & Architectures*
- Apprentissage*

# Modèles & Architecture

---

- Neurone formel de McCulloch & Pitts
- Modèle non-linéaire d'un neurone
- Fonctions d'activation
- Architectures de réseaux
- Représentation des connaissances
- Apprentissage: Perceptron
- Apprentissage: Rétro-propagation

# Quelques notes historiques

---

références les plus importantes:

- J.J. Hopfield

*Neural Networks and physical systems with emergent collective computational abilities*

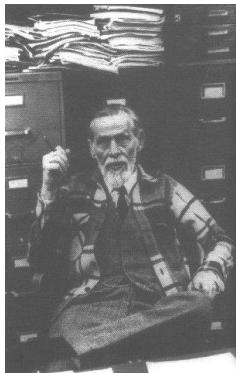
Proc. Natl. Acad. of Sciences, vol. 79, 1554-1558, 1982.

- D.E. Rumelhart & J.L. McClelland

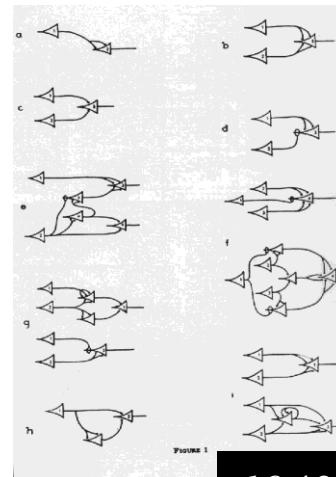
*Parallel Distributed Processing: Exploration in the Microstructure of Cognition*

MIT Press, Cambridge, 1986.

# Neurone formel de McCulloch & Pitts



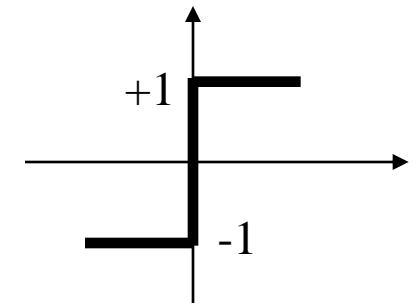
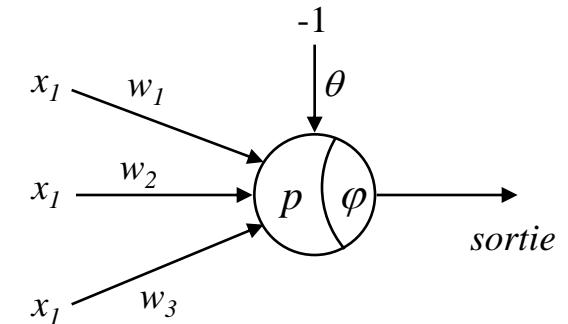
Warren S. McCulloch



William Pitts

$$p = \sum_i w_i x_i \quad \text{si } p > \theta \text{ then } \text{sortie} = +1$$

else  $\text{sortie} = -1$



fonction Signum

# Interprétation géométrique

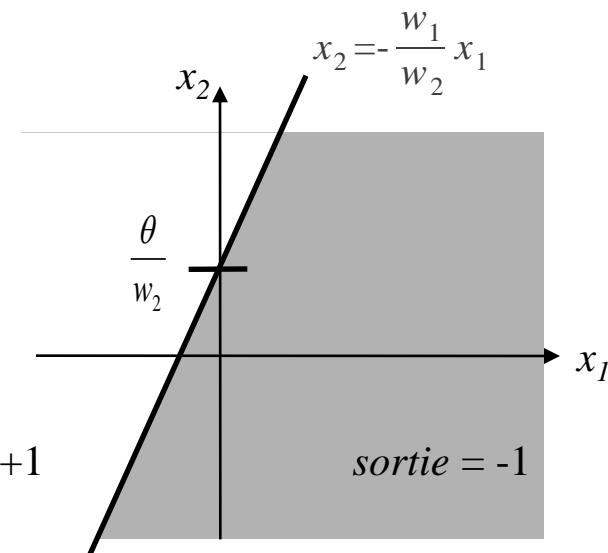
Soit un neurone formel de McCulloch&Pitts avec deux entrées  $x_1$  et  $x_2$

sa *sortie* est +1 si:  $w_1x_1 + w_2x_2 > \theta$

sa *sortie* est -1 si:  $w_1x_1 + w_2x_2 \leq \theta$

Géométriquement ces deux équations partitionnent le plan  $(x_1, x_2)$  par une droite définie par l'équation suivante:

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}$$



# Modèle non-linéaire de neurone

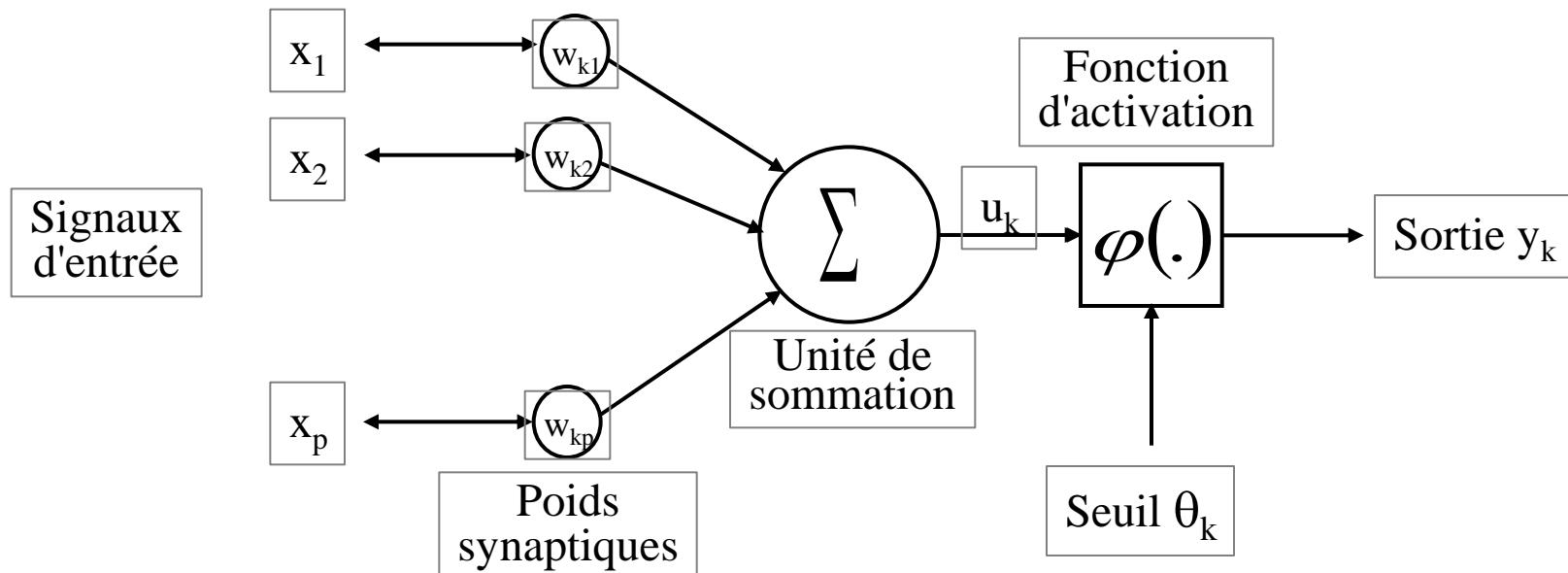
---

3 éléments fondamentaux:

- Un ensemble de *synapses* caractérisées par un poids  $w_{kj}$ 
  - $w_{kj} > 0 \Rightarrow$  synapse excitatrice,
  - $w_{kj} < 0 \Rightarrow$  synapse inhibitrice,
- un *additionneur* pour sommer les signaux d'entrée,
- une *fonction d'activation* pour limiter l'amplitude de la valeur de sortie.

# Modèle non-linéaire de neurone (cont.)

(version moderne du modèle de McCulloch & Pitts)



# Modèle non-linéaire de neurone (cont.)

---

Ce modèle est décrit mathématiquement par:

$$(1) \quad u_k = \sum_{j=1}^p w_{kj} x_j \quad \text{et} \quad y_k = \varphi(u_k - \theta_k)$$

où:

$x_1, x_2, \dots, x_p$  sont les *entrées*,

$w_{k1}, w_{k2}, \dots, w_{kp}$  sont les *poids synaptiques* du neurone k,

$u_k$  est la sortie de *l'additionneur*,

$\theta_k$  est le *seuil*,

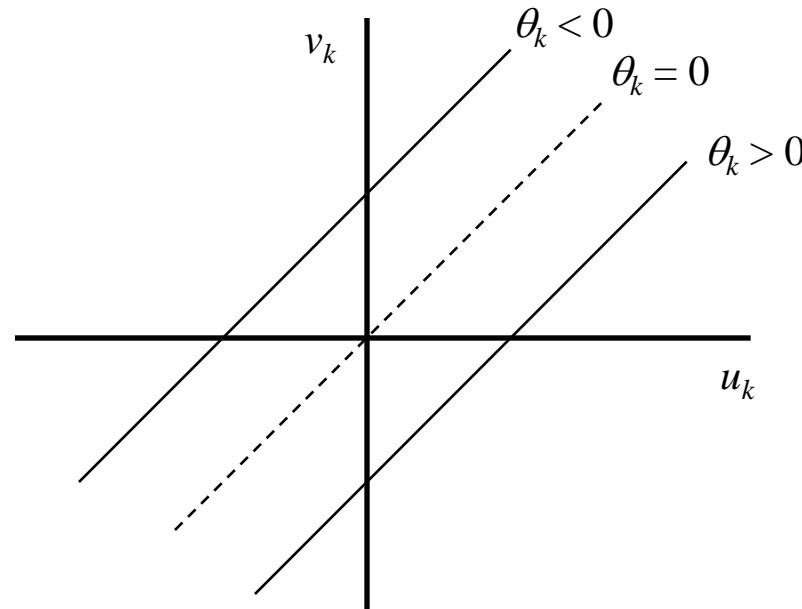
$\varphi(\cdot)$  est la *fonction d'activation*,

$y_k$  est la valeur de *sortie* du neurone.

# Effets de la valeur de seuil

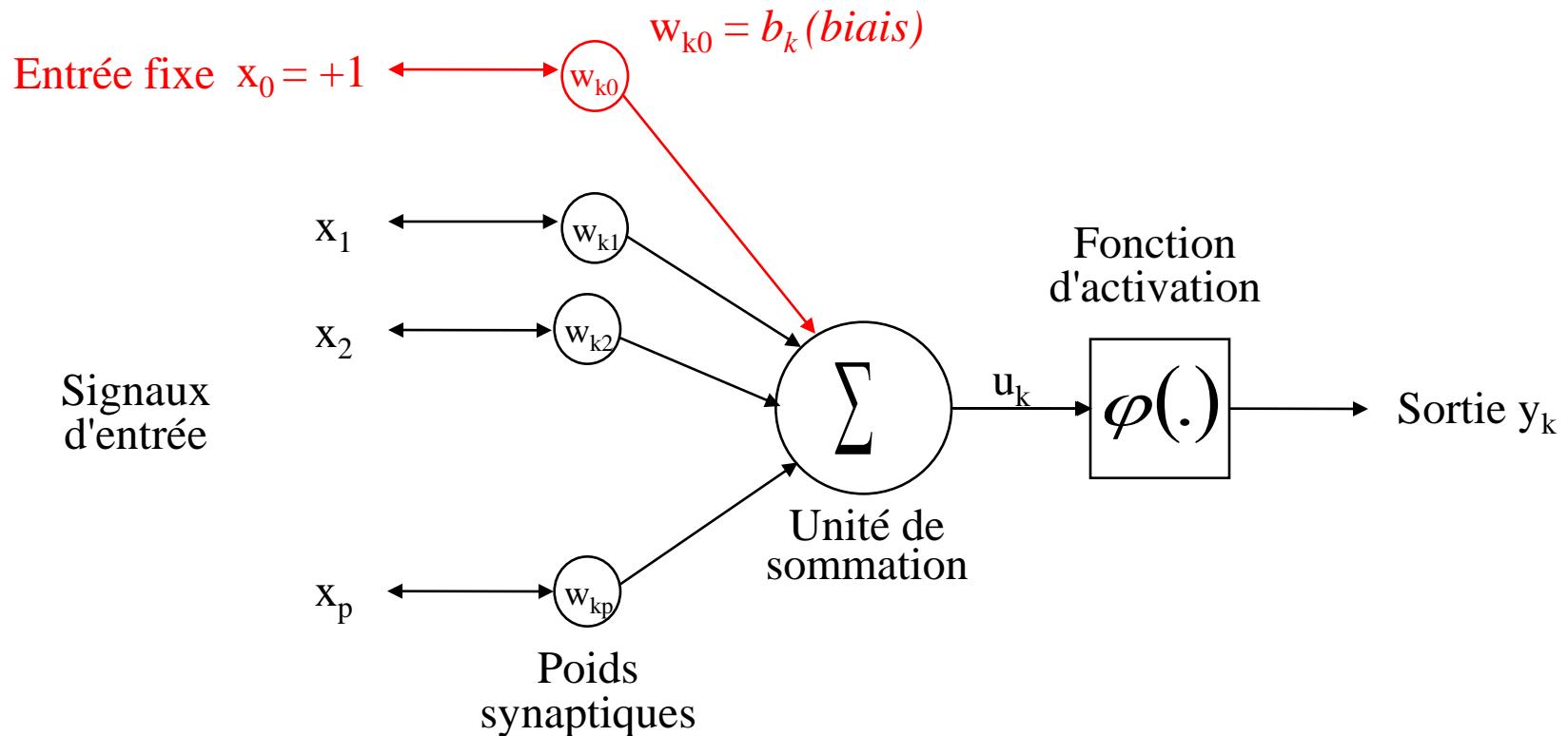
La valeur de seuil agit comme une *transformation affine* sur la valeur de sortie  $u_k$ :

$v_k$  = potentiel d'activation du neurone  $k$



$$v_k = u_k - \theta_k \quad \text{ou en utilisant l'équ. (1)} \quad v_k = \sum_{j=1}^p w_{kj} x_j - \theta_k$$

# Modèle étendu (alternative)



# Types de fonctions d'activation

---

La fonction d'activation définit la valeur de sortie d'un neurone en fonction des valeurs de ses entrées.

3 types de fonctions d'activation:

- *Fonction à seuil*

$$y_k = \varphi(v_k) = \begin{cases} 1 & \text{si } v_k \geq 0 \\ 0 & \text{si } v_k < 0 \end{cases}$$

- *Fonction 3 marches*

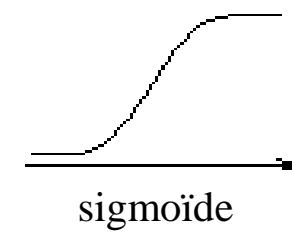
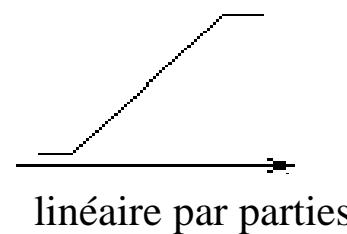
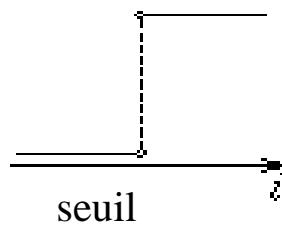
$$\varphi(v) = \begin{cases} 1 & v \geq \alpha \\ v & \alpha > v > \beta \\ 0 & v \leq \beta \end{cases}$$

# Types de fonctions d'activation (cont.)

- Fonction sigmoïde

$$\varphi(v) = \frac{1}{1 + e^{-av}} \quad \text{ou} \quad \varphi(v) = \tanh\left(\frac{v}{2}\right) = \frac{1 - e^{-v}}{1 + e^{-v}}$$

- graphiquement

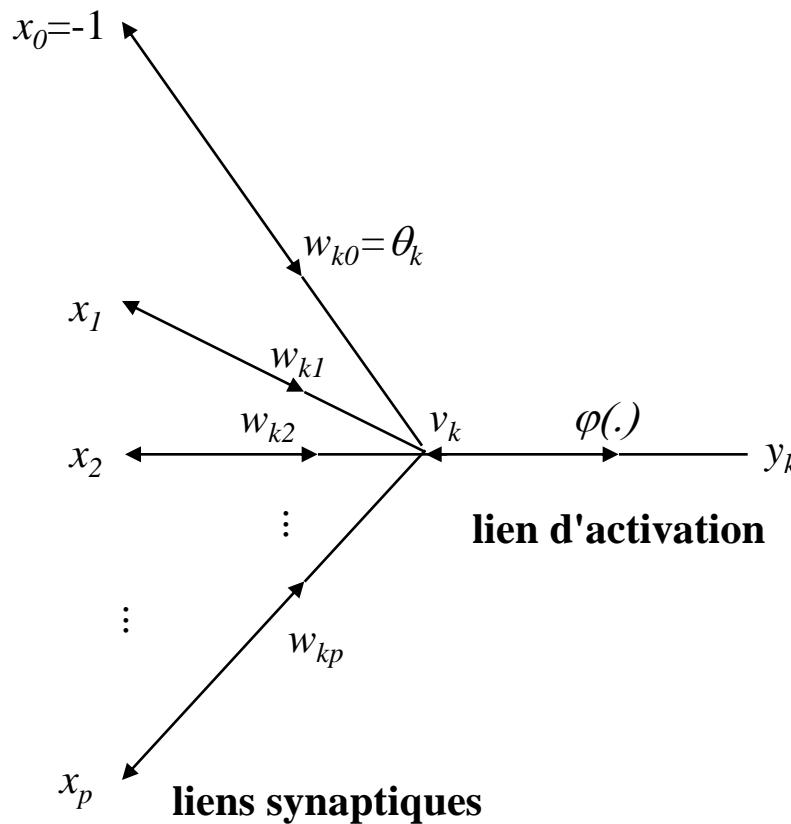


# Définition mathématique

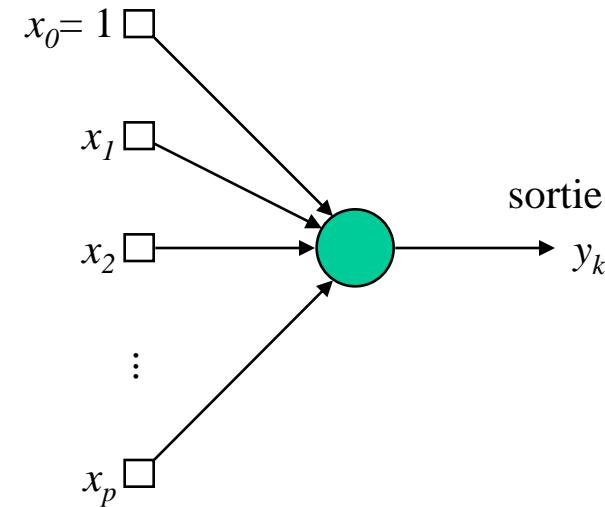
Un réseau neuronal est un *graphe orienté* constitué de *nœuds* liés par des *liens synaptiques* et *d'activation* caractérisé par les propriétés suivantes:

- un neurone est représenté par:
  - un ensemble de liens synaptiques linéaires,
  - un lien d'activation non-linéaire,
  - un seuil
- les liens synaptiques pondèrent leurs signaux d'entrée,
- la somme pondérée des signaux d'entrée = niveau d'activité interne du neurone,
- le lien d'activation transforme le niveau d'activité interne en valeur de sortie (= variable d'état du neurone).

# Diagrammes "signal-flow" & architecturaux



Graphe "signal-flow"



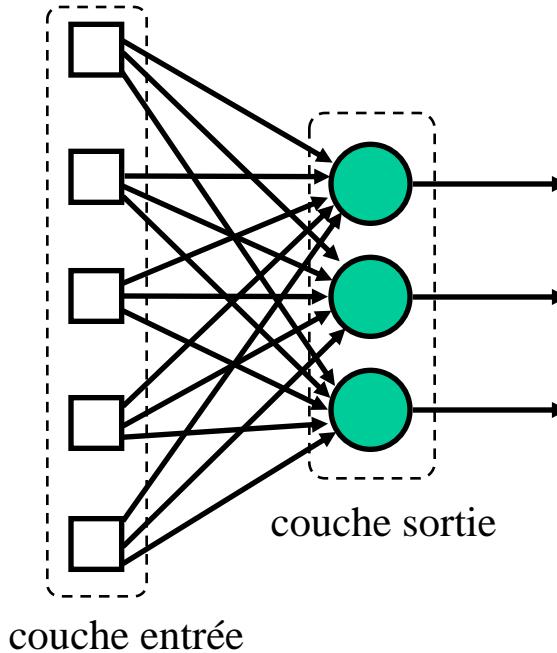
Graphe architectural

# Architectures de réseaux

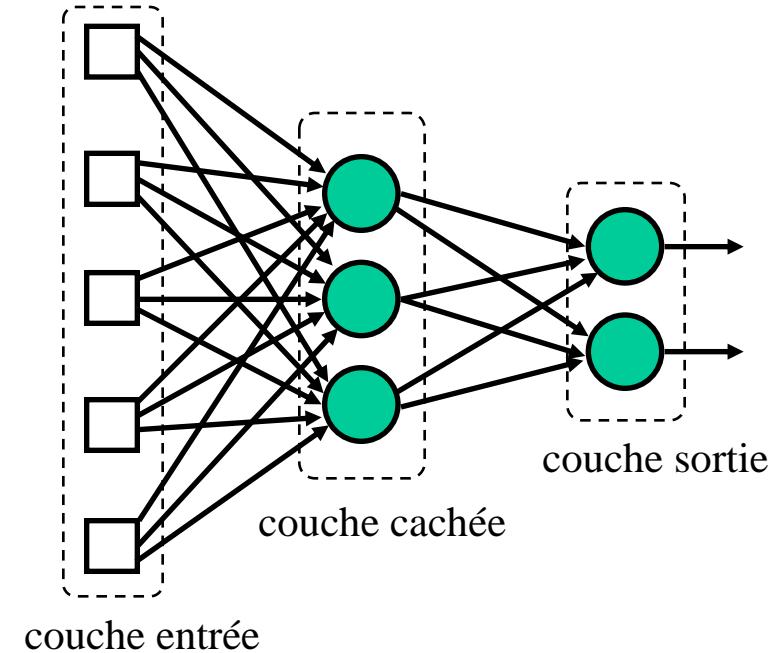
---

- Il y a 4 différentes classes d'*architectures*:
  - réseaux monocouche "feedforward",
    - une couche d'entrés de nœuds-source,
    - une couche de sortie,
    - "feedforward": entrée  $\Rightarrow$  sortie (pas vice versa)
  - réseaux multicouche "feedforward",
    - une couche d'entrés de nœuds-source,
    - une ou plusieurs couches cachées,
    - une couche de sortie,
  - réseaux récurrents,
    - au moins une boucle de rétro-action,
  - structures en treillis,
    - neurones organisés en matrice.

# *Architectures "feed-forward"*

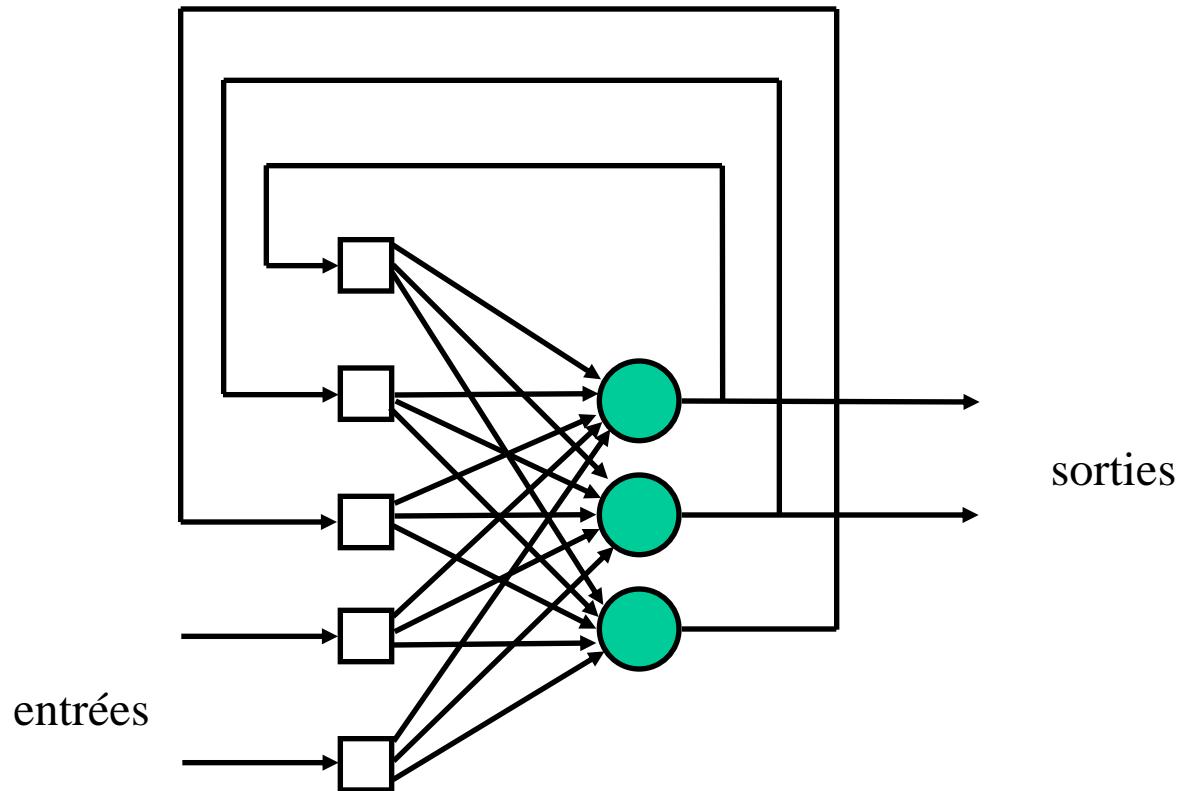


Réseau monocouche "feedforward"



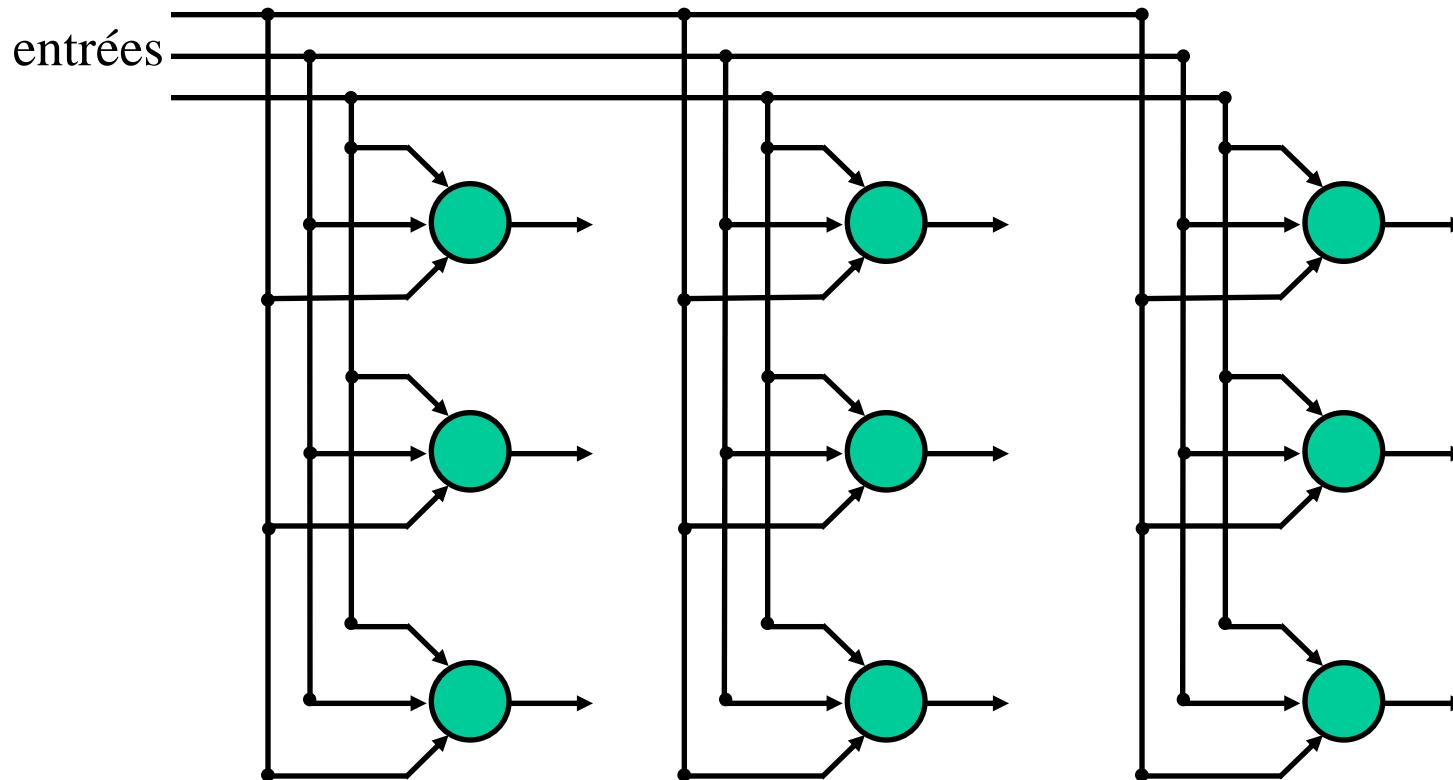
Réseau multicouche "feedforward"

# Architecture récurrente



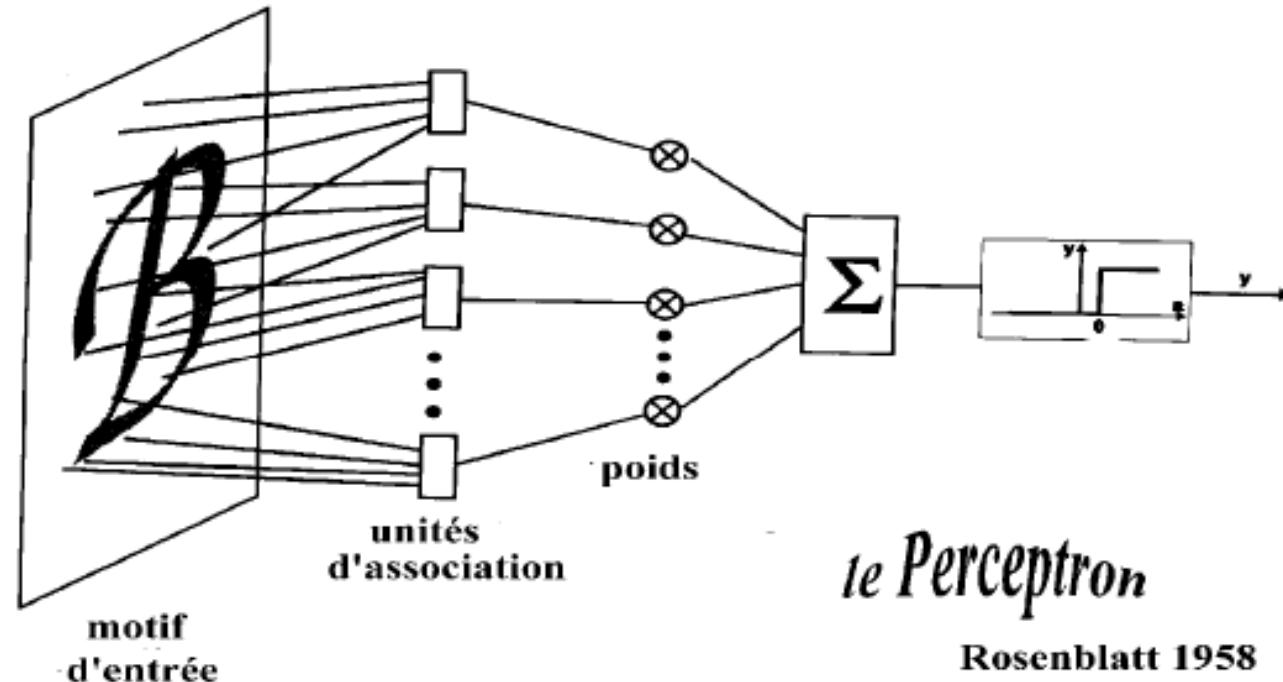
Réseau récurrent avec neurones cachés

# Architecture en treillis



Réseau 2-D 3x3

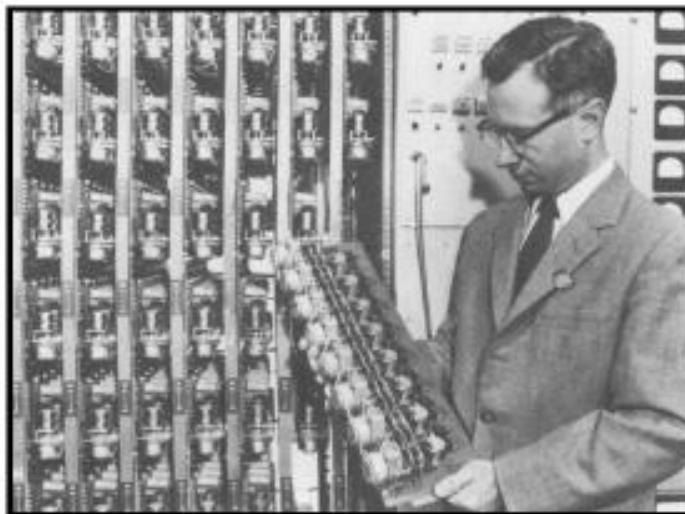
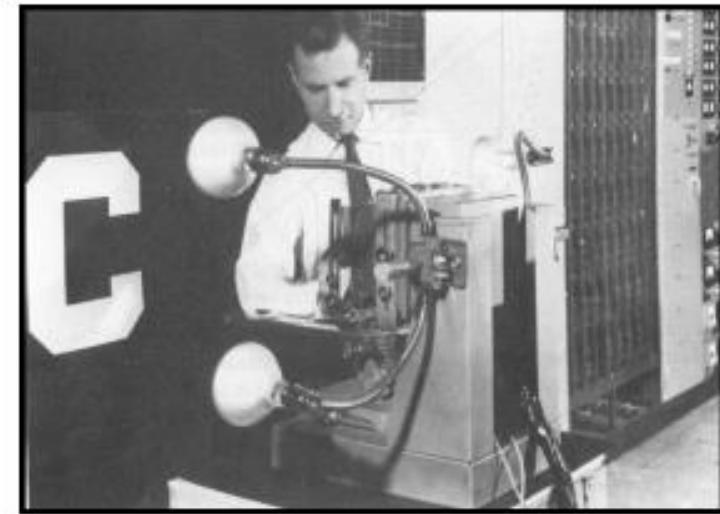
# Le Perceptron (F. Rosenblatt, 1958)



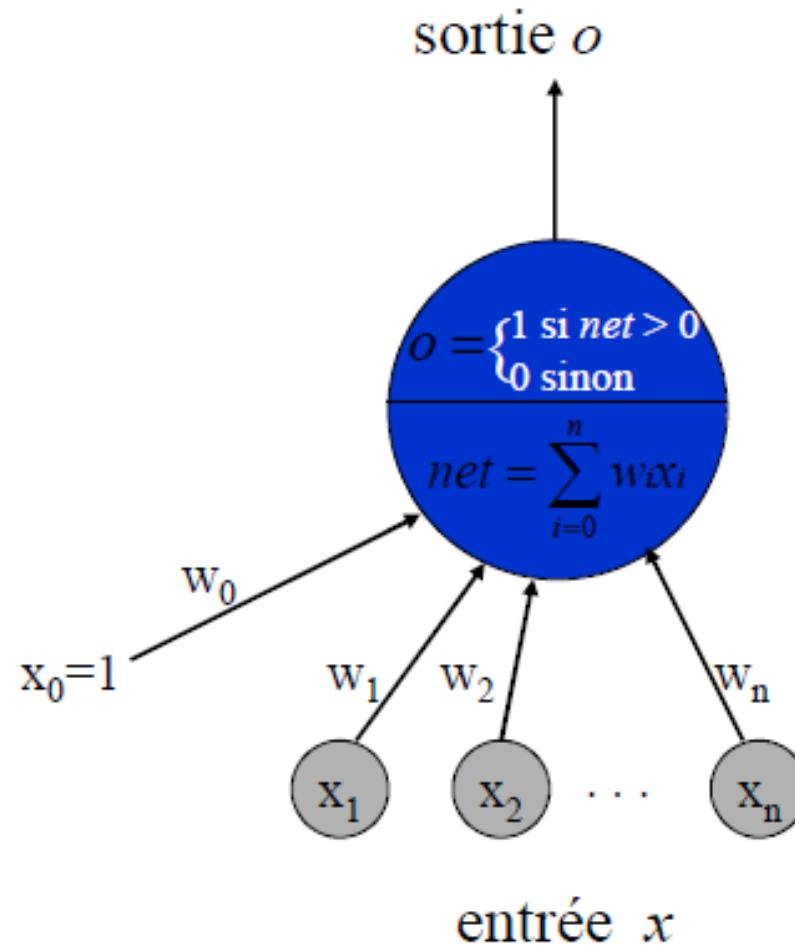
# Le Mark I Perceptron



F. Rosenblatt

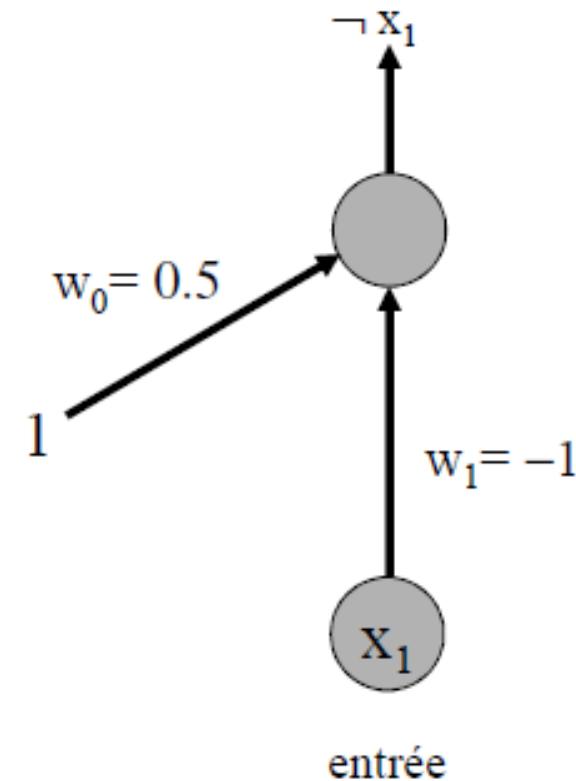


# Le modèle "Perceptron"



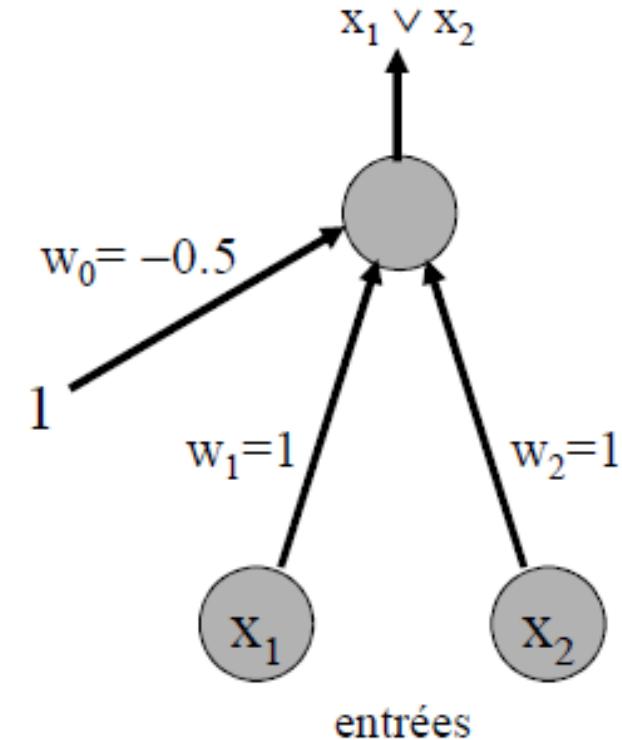
# Exemple: négation logique

entrée $x_1$	sortie
0	1
1	0



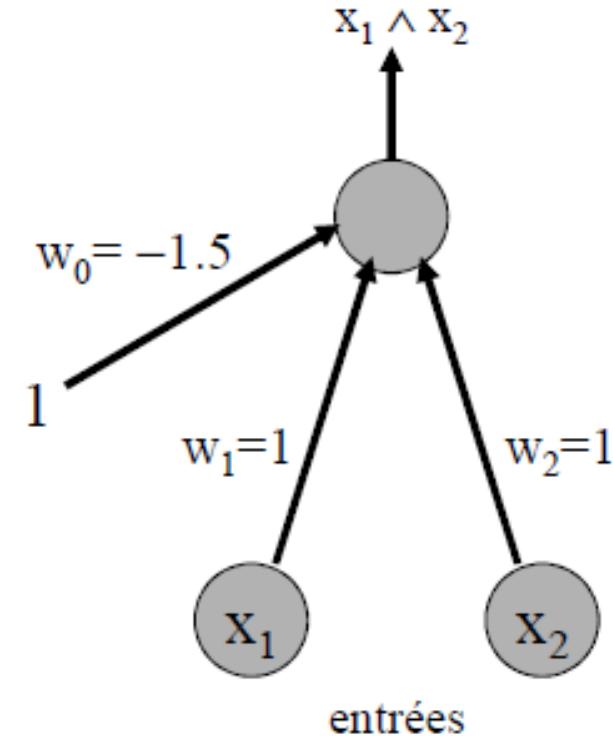
## Exemple: "ou" logique

entrée $x_1$	entrée $x_2$	sortie
0	0	0
0	1	1
1	0	1
1	1	1



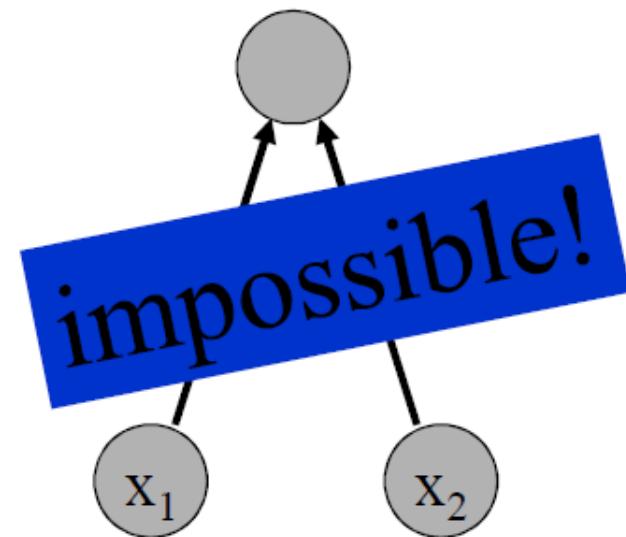
# Exemple: "et" logique

entrée $x_1$	entrée $x_2$	sortie
0	0	0
0	1	0
1	0	0
1	1	1



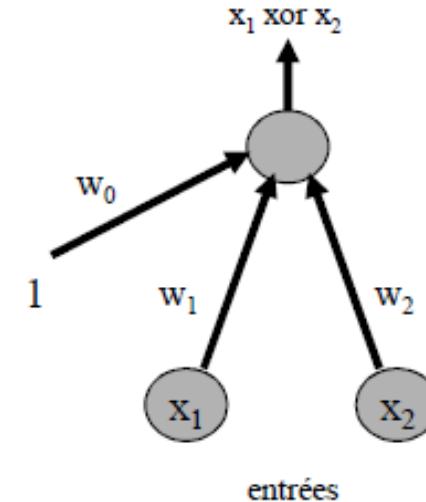
# Exemple: "xor" logique (ou exclusif)

entrée $x_1$	entrée $x_2$	sortie
0	0	0
0	1	1
1	0	1
1	1	0



# Pourquoi "impossible" ?

entrée x1	entrée x2	sortie
0	0	0
0	1	1
1	0	1
1	1	0

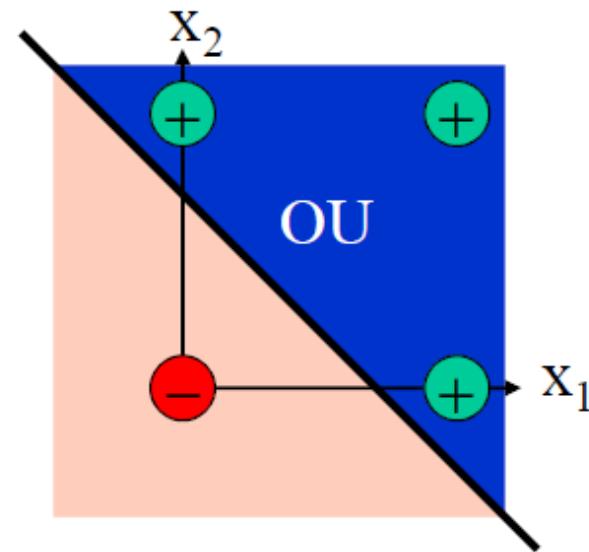


Le système d'équations à résoudre est:

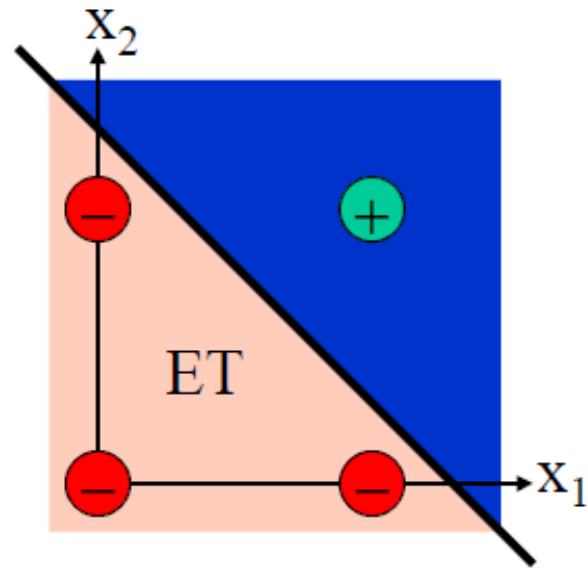
$$\begin{aligned} w_0 + 0.w_1 + 0.w_2 &\leq 0 \\ w_0 + 0.w_1 + 1.w_2 &> 0 \\ w_0 + 1.w_1 + 0.w_2 &> 0 \\ w_0 + 1.w_1 + 1.w_2 &\leq 0 \end{aligned}$$

Il n'y a aucune valeur possible pour les coefficients  $w_0$ ,  $w_1$  et  $w_2$  qui satisfasse les inégalités ci-contre.  
**xor ne peut donc pas être représenté!**

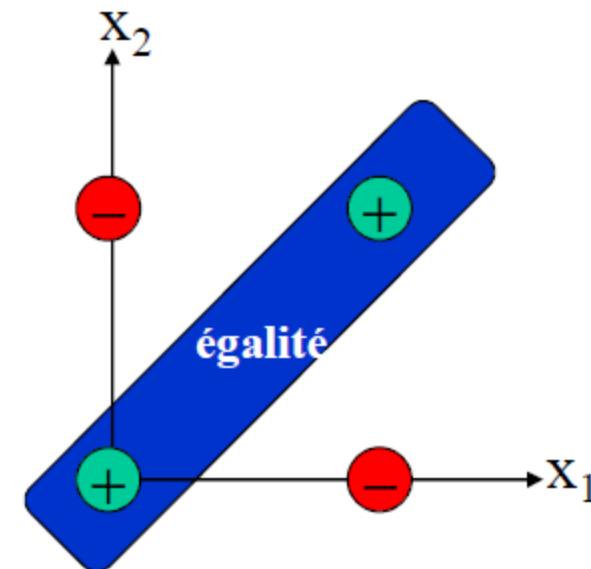
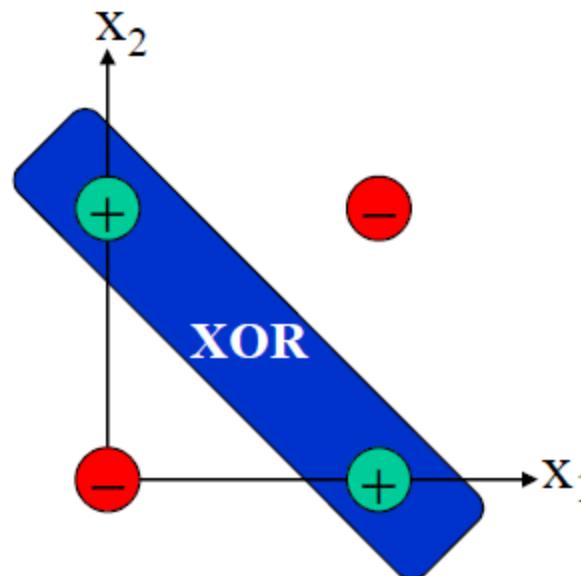
# Séparabilité linéaire



# Séparabilité linéaire

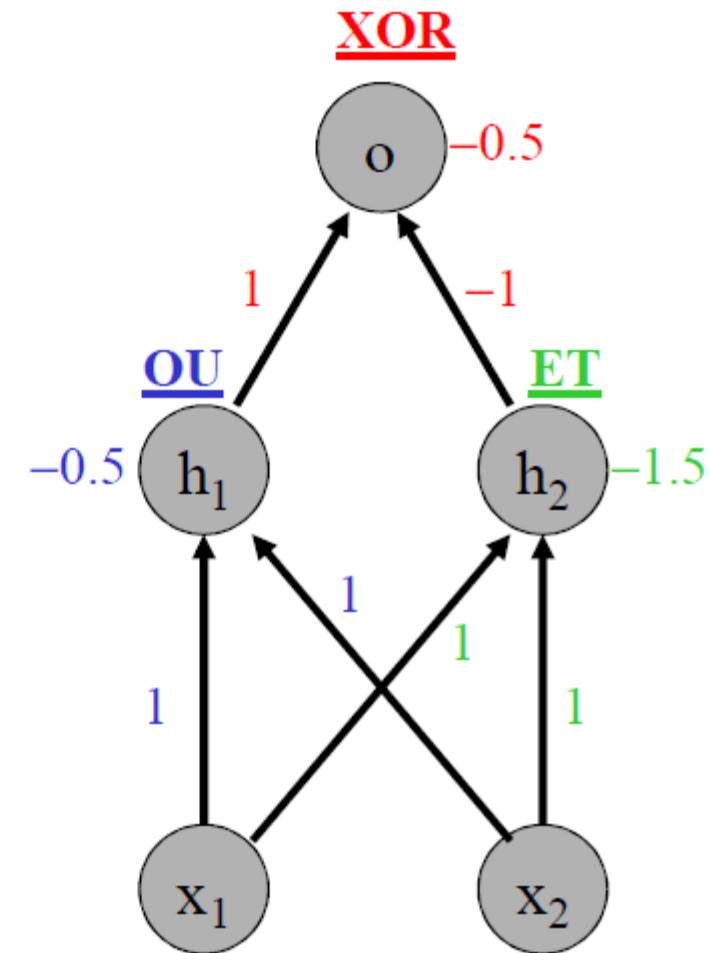


# Non-séparabilité linéaire



# Exemple: "xor" logique (revu)

entrée x1	entrée x2	sortie
0	0	0
0	1	1
1	0	1
1	1	0



# Théorèmes du Perceptron

---

## Théorème de représentation

*Un réseau "feedforward" à une seule couche (Perceptron) peut uniquement représenter des fonctions linéairement séparables. C'est-à-dire celles pour lesquelles la surface de décision séparant les cas positifs des cas négatifs est un (hyper-)plan.*

## Théorème d'apprentissage (F. Rosenblatt)

*Étant donné suffisamment d'exemples d'apprentissage, il existe un algorithme qui apprendra n'importe quelle fonction linéairement séparable.*

# Algorithme d'apprentissage du Perceptron

Entrées: ensemble d'apprentissage  $\{(x_1, x_2, \dots, x_n, t)\}$

## Méthode

initialiser aléatoirement les poids  $w(i)$ ,  $0 \leq i \leq n$

répéter jusqu'à convergence:

pour chaque exemple

calculer la valeur de sortie  $o$  du réseau.

ajuster les poids:

$$\Delta w_i = \eta (t - o) x_i$$

$$w_i \leftarrow w_i + \Delta w_i$$

Règle d'apprentissage  
du *Perceptron*

# Algorithme d'apprentissage du Perceptron (résumé)

$$w_i \leftarrow w_i + \Delta w_i$$

nouveau poids      ancien poids      modification  
 $\downarrow$                    $\downarrow$                    $\downarrow$   
 $w_i$      $\leftarrow$      $w_i$     +     $\Delta w_i$

$$\Delta w_i = \eta (t - o) x_i$$

modification      taux      valeur      entrée  
 $\uparrow$                    $\uparrow$                    $\uparrow$                    $\uparrow$   
 d'apprentissage      d'apprentissage      attendue      de sortie  
 $\uparrow$                    $\uparrow$                    $\uparrow$                    $\uparrow$   
 valeur      entrée      attendue      du Perceptron

# Erreur quadratique

---

- La règle d'apprentissage du *Perceptron* effectue une descente de gradient dans l'espace des poids.
- Considérons une unité linéaire simple pour laquelle

$$o = w_0 + w_1 x_1 + \dots + w_n x_n$$

- définissons l'erreur comme:

$$E[w_0, w_1, \dots, w_n] = \frac{1}{2} \sum_{e \in \text{Exemples}} (t_e - o_e)^2$$

(*erreur quadratique*)

# Convergence

*La convergence est garantie car l'erreur  $E$  est une forme quadratique dans l'espace des poids. Elle possède donc un seul minimum global et la descente du gradient assure de le trouver.*

Convergence si:

- ... les données d'apprentissage sont linéairement séparables
- ... le taux d'apprentissage  $\eta$  est suffisamment petit
- ... il n'y a pas d'unités "cachées" (une seule couche)