Algorithmique avancée

Laurent Lemarchand
Lab-STICC/UBO
Laurent.Lemarchand@univ-brest.fr

Optimisation combinatoire
cours

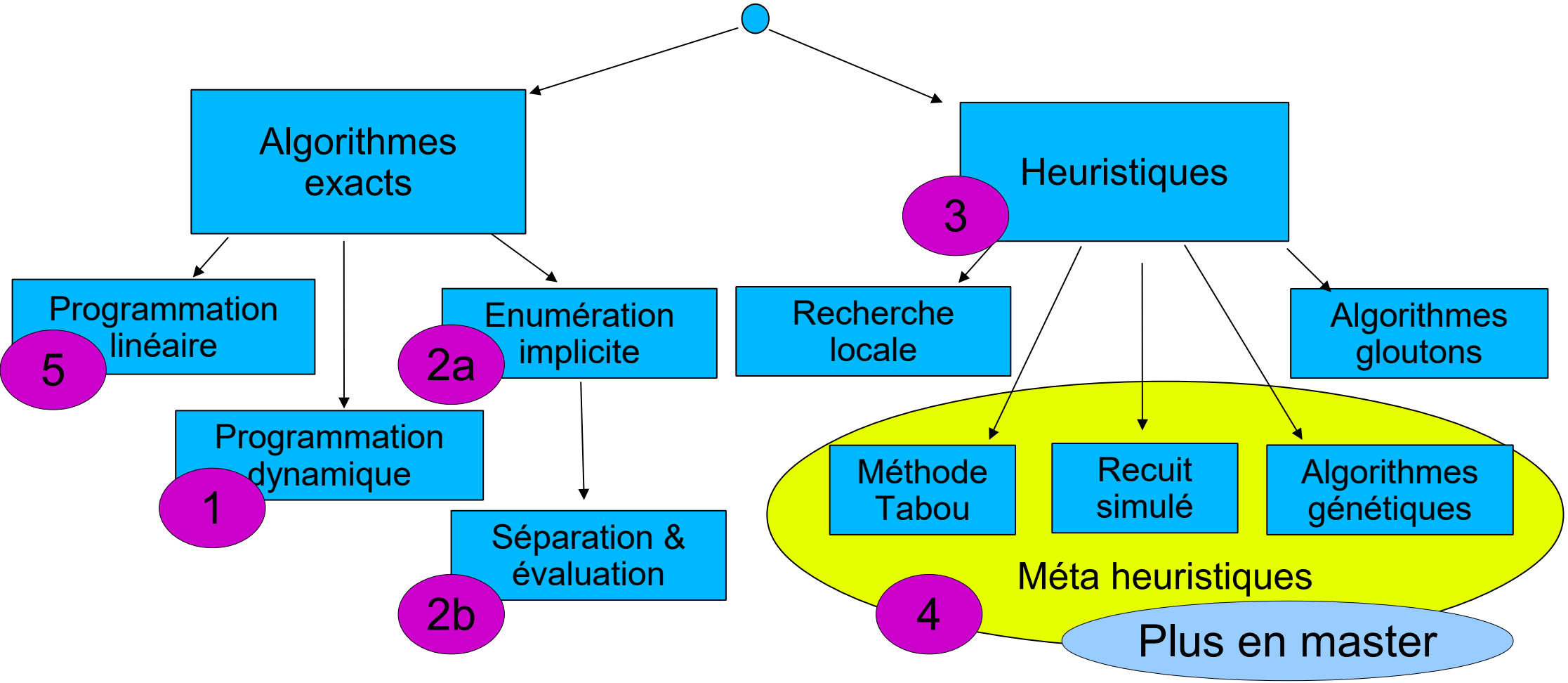
- Cours
  - Méthodes d'optimisation (suivant objectifs qualitatifs et critères simples ou multiples)
- Exemples
  - Problèmes classiques d'optimisation
  - Applicables tels quels (rarement) ou exploitables pour modéliser un problème plus complexe et/ou le résoudre en partie ou totalement

Optimisation combinatoire
bibliographie

- Méthodes d'optimisation combinatoire
  - I.Charon, A.Germa, O.Hudry
- Optimisation combinatoire (tome programmation discrète)
  - M.Sakarovitch

- Web ...

Algorithmique avancée
méthodes de recherche – plan



1

Programmation dynamique
principe (Bellman)

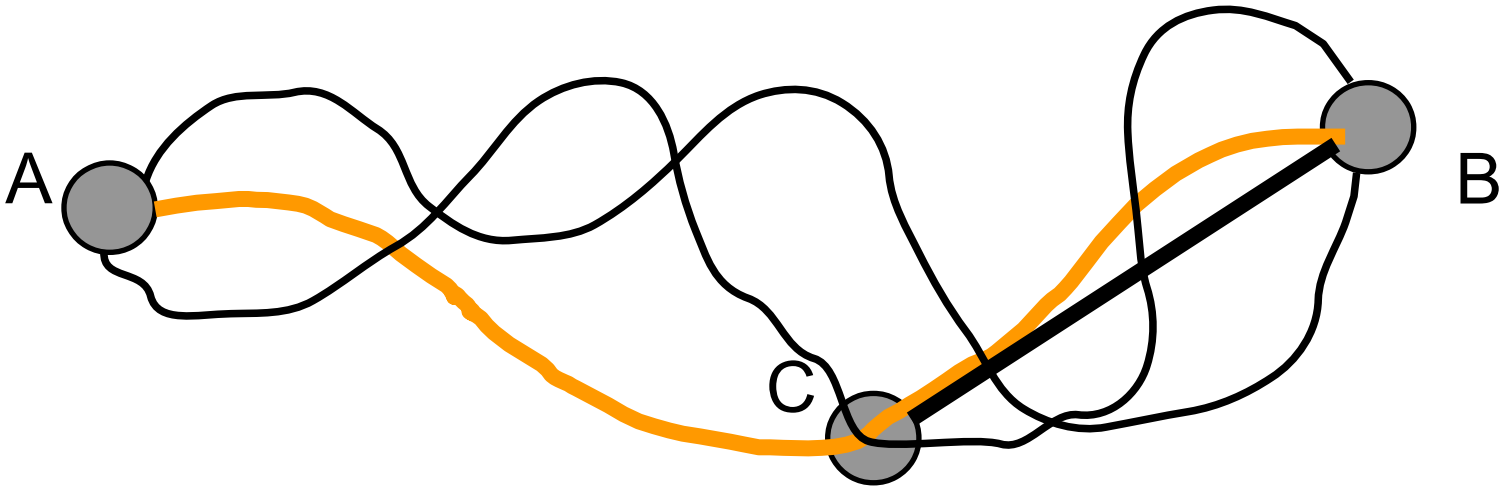
- Décomposer le problème P en sous-problèmes
  $P_1, P_2, \ldots, P_n$
- Résoudre les sous-problèmes pour obtenir les solutions
  $v_1, v_2, \ldots, v_n$
- Combiner ces solutions pour résoudre P :
  $V^* = f(v_1, v_2, \ldots, v_n)$
- Mécanisme récursif

Programmation dynamique
conditions d'utilisation

- Principe d'optimalité

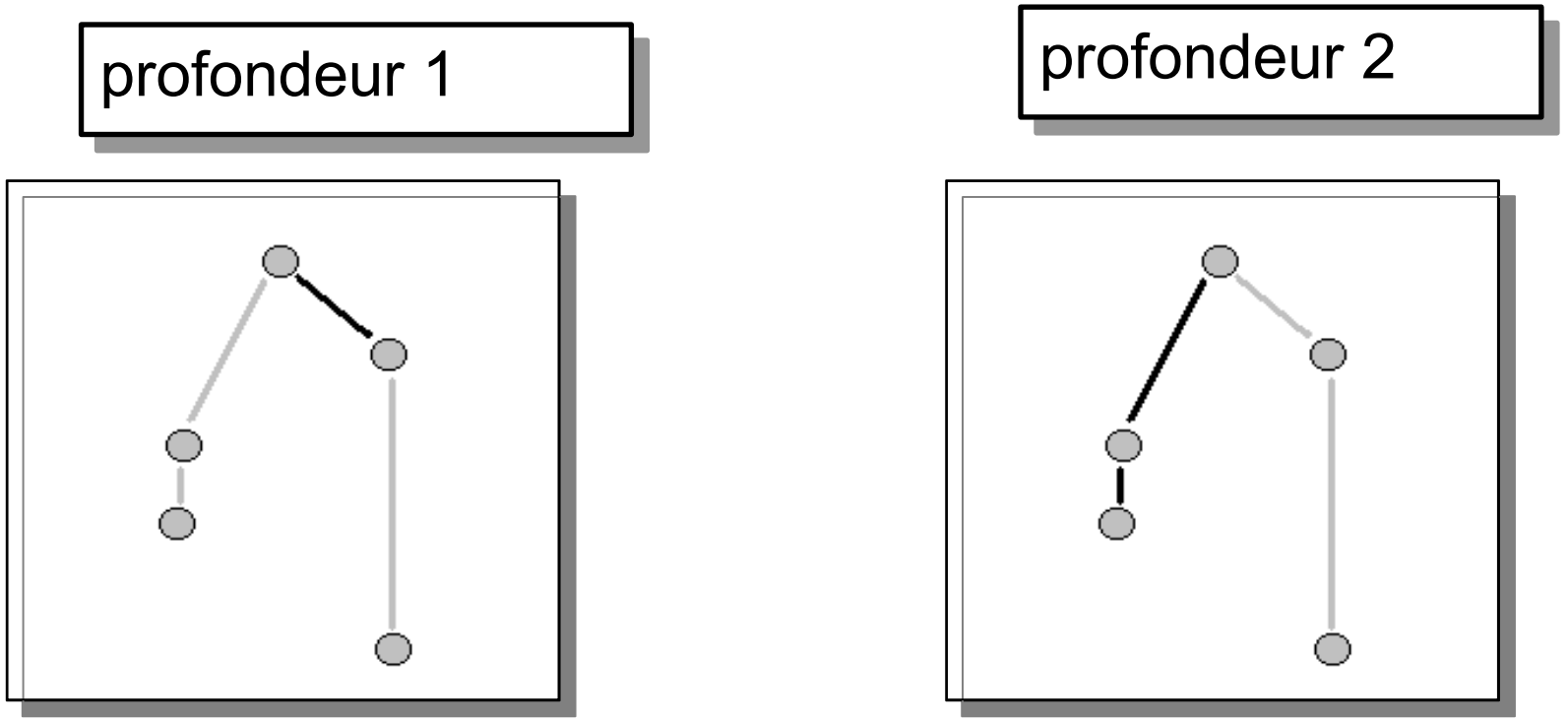
> Une solution est optimale ssi.
> ses sous-solutions sont optimales

- Exemple : plus court chemin : (A,B) est optimal ssi (A,C) et (B,C) sont optimaux.



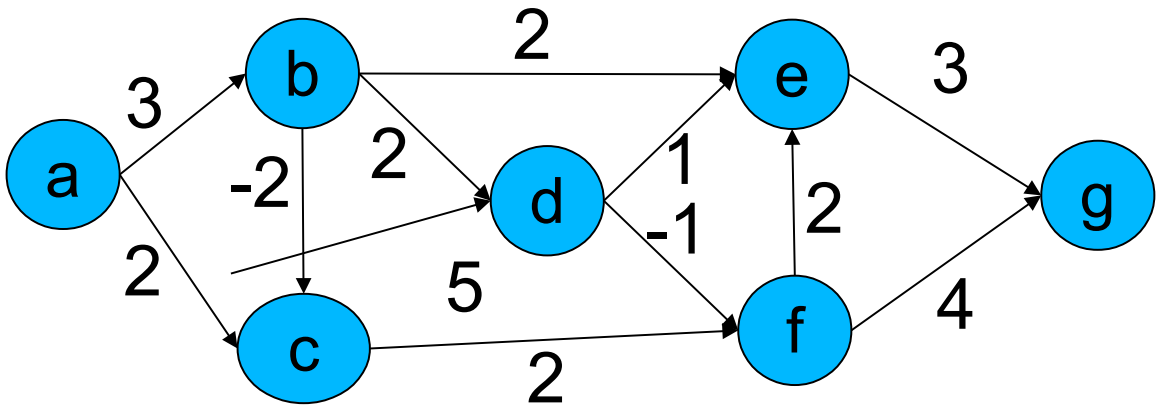
Principe d'optimalité
contre exemple

- Recherche de plus court chemin de profondeur donnée dans un arbre



•Il faut examiner toutes les solutions partielles

Programmation dynamique
plus court chemin : Ford-Bellman

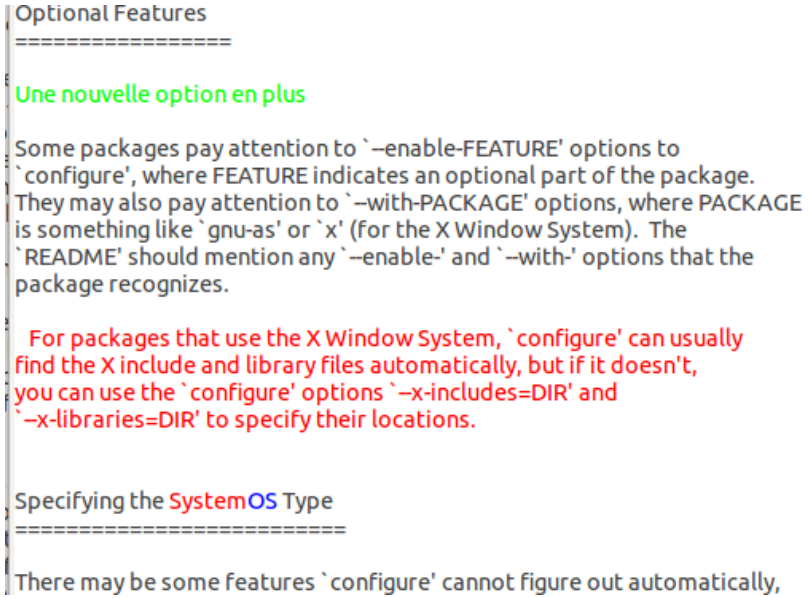


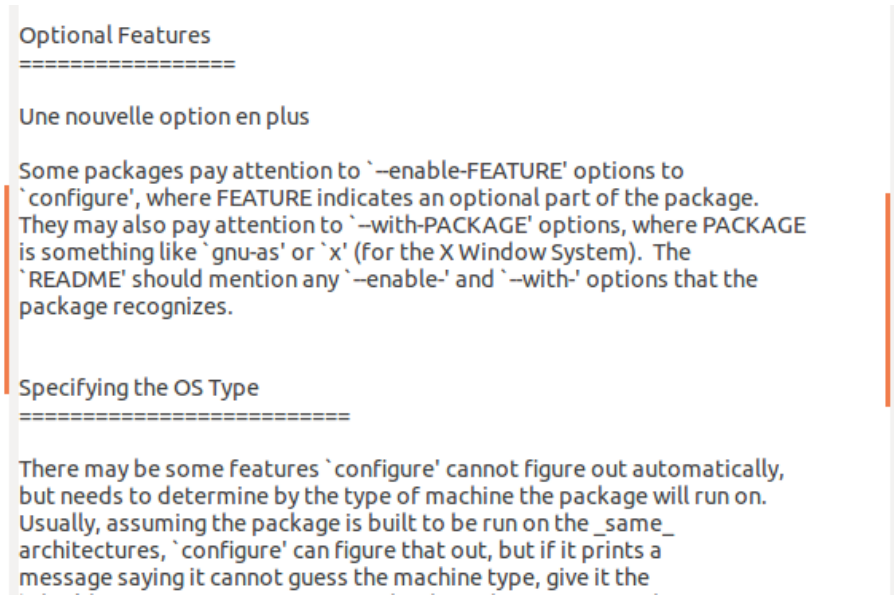
- Calcul de chemins : plus court chemin de **a** à l'un de ses successeurs
- Récursivement *:* $\Pi(s) = min_{s' \in preds(s)} \Pi(s') + w_{s' \to s}$
- Autres exemples *: Chortle-crf,* allocation de ressources, gestion de stock ...

Programmation Dynamique
Edition distance : Levenshtein metric

- String metric for computing differences between 2 strings
- Number of edition operations to do for rewriting string A into B
  - Insertion: poe > pome
  - Deletion: pome > ome
  - Substitution: ome > ame
- diff and agrep commands

poe to ame distance
is 3 – minimal ?





Dynamic programming
Levenshtein algorithm

```
algo Levenshtein

input
string str1[1..n1]
string str2[1..n2]
table d[n1+1,n2+1]

init
d[0,0] = 0
// if one string is null
// dist = other's length
∀i ∈ [1.. n1] d[i,0] = i
∀j ∈ [1.. n2] d[0,j] = j

for i = 1 to n1
    for j = 1 to n2
        if str1[i-1] == str2[j-1] then  cost = 0
        else cost = 1 endif

        d[i, j] := minimum(
                        d[i-1, j  ] + 1,        // delete
                        d[i,   j-1] + 1,        // insert
                        d[i-1, j-1] + cost      // substitute
                    )
    endfor
endfor

return d[n1, n2]
end Levenshtein
```

Dynamic programming
Levenshtein algorithm example

- Computing d[i,j]
  - Left + 1: d[i-1, j] + 1. (insert)
  - Above +1: d[i, j-1] + 1 (delete)
  - Diagonal : d[i-1,j-1]+ (0 or 1) substitution (if needed)

		C	H	I	E	N	S
	0	1	2	3	4	5	6
N	1	0	0	0	0	0	0
I	2	0	0	0	0	0	0
C	3	0	0	0	0	0	0
H	4	0	0	0	0	0	0
E	5	0	0	0	0	0	0

First step →

		C	H	I	E	N	S
	0	1	2	3	4	5	6
N	1	1	2	3	4	4	5
I	2	0	0	0	0	0	0
C	3	0	0	0	0	0	0
H	4	0	0	0	0	0	0
E	5	0	0	0	0	0	0

.....

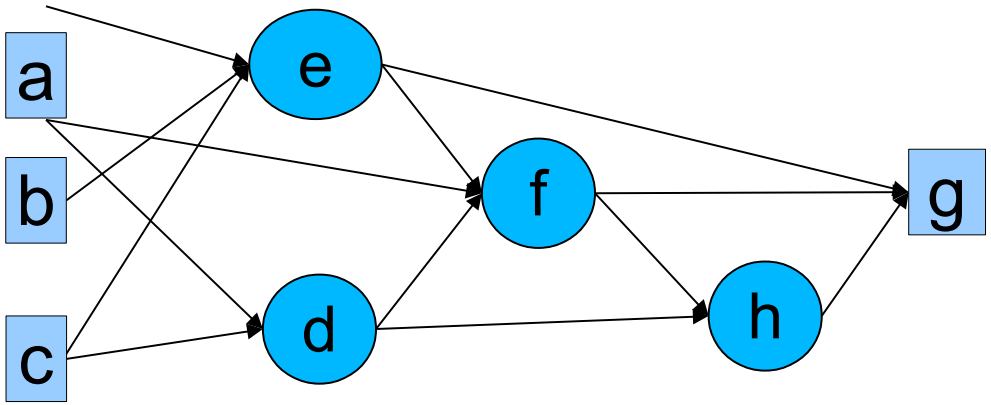
		C	H	I	E	N	S
	0	1	2	3	4	5	6
N	**1**	1	2	3	4	4	5
I	**2**	2	2	2	3	4	5
C	3	**2**	3	3	3	4	5
H	4	3	**2**	**3**	4	4	5
E	5	4	3	3	**3**	**4**	**5**

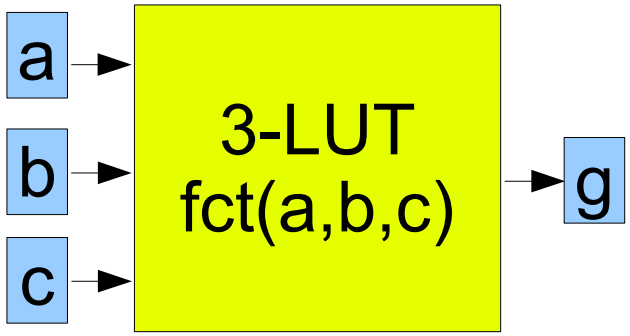
dist(NICHE to CHIEN) = 5

Example : wikipedia

Programmation dynamique
Exemple : Chortle-crf

- Implantation de circuits logiques sur FPGA

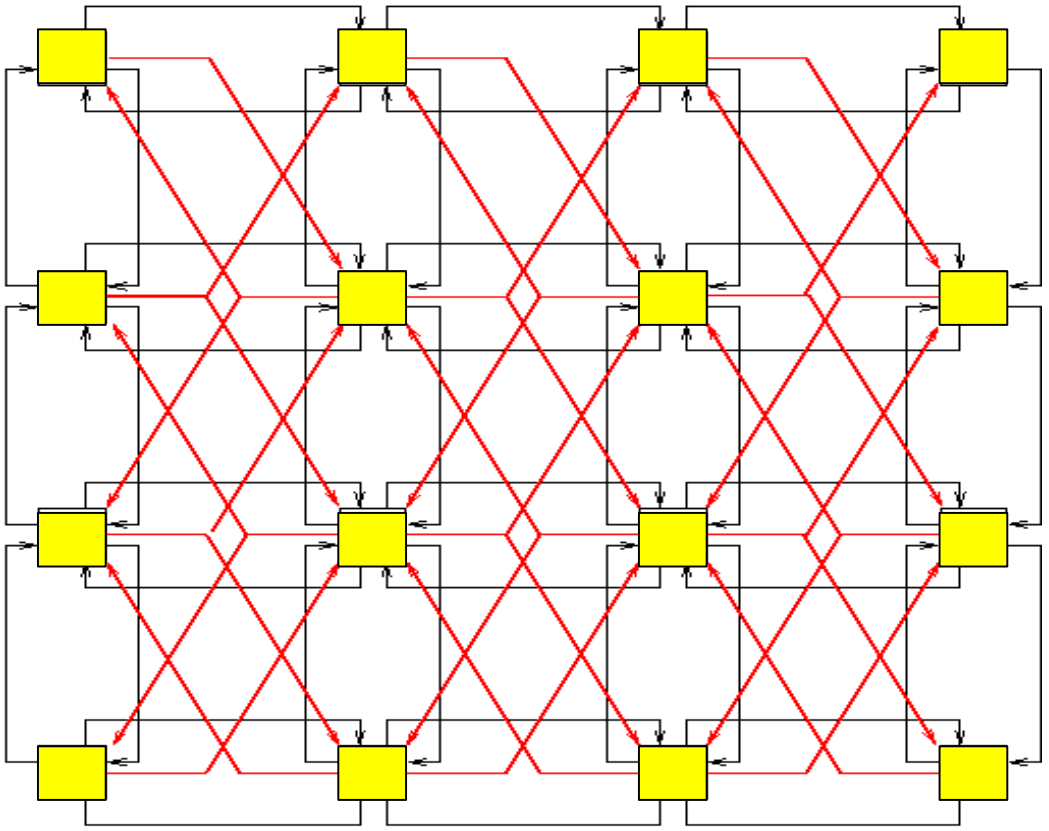




- Grouper les noeuds logiques en k-LUTs : une LUT peut implanter n'importe quelle fonction logique d'au plus k entrées (table de vérité)

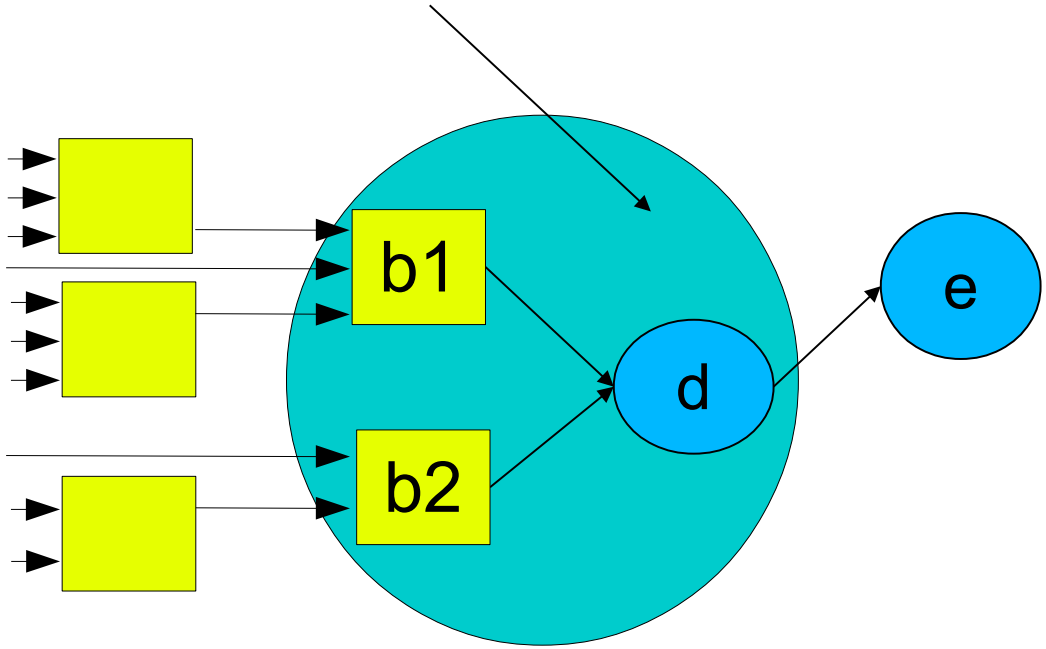
Chortle-crf
FPGA (Field Programmable Gate Array)

- Minimiser le nombre de LUTs occupées
- Placer et router les LUTs après leur définition



Programmation dynamique
Exemple : Chortle-crf

- Progression des entrées vers les sorties



- La solution pour d+b1+b2 doit
  - Minimiser le nombre de LUT
  - Minimiser le fanin de la LUT de tête

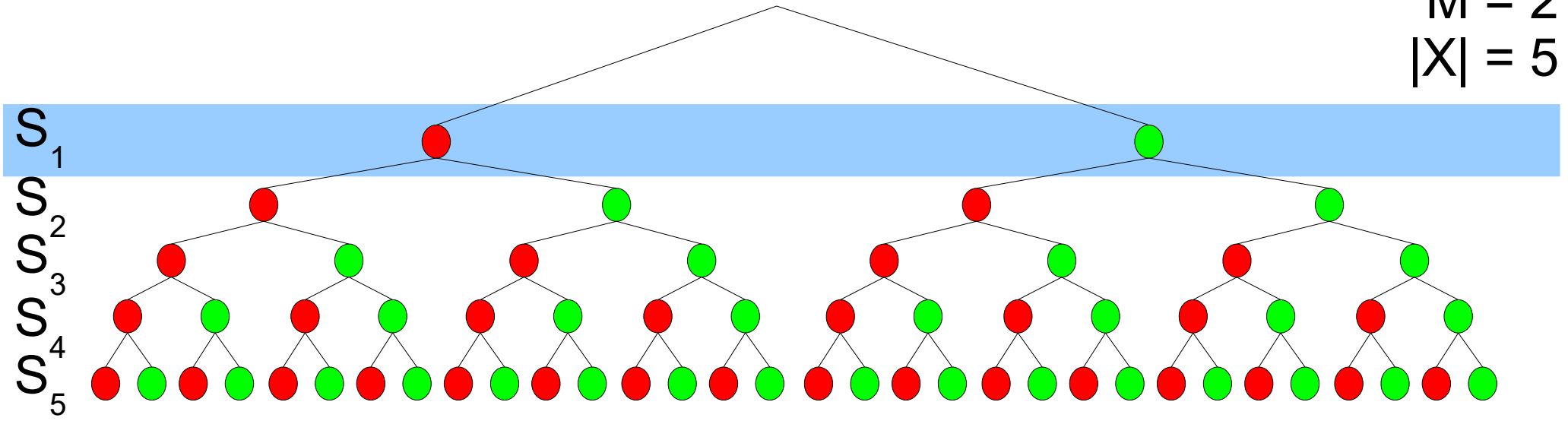
2a

Méthodes par énumération implicite
arbre de recherche

- Enumération implicite de toutes les solutions
- Découper les solutions en sous-ensembles
  - Sous espaces de recherche S_1, S_2, ..., S_n
  - → *arbre de recherche*

- Exemple: coloration d'un graphe en 2 couleurs

M = 2
|X| = 5

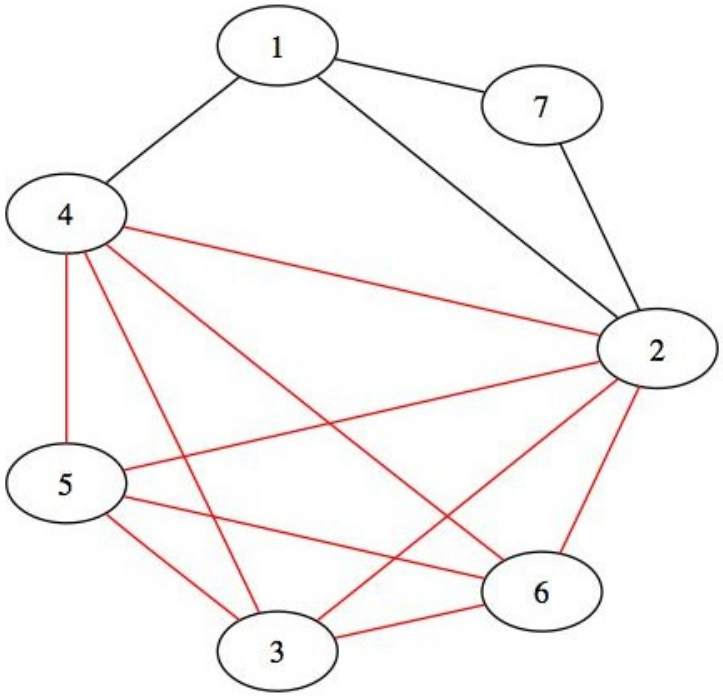


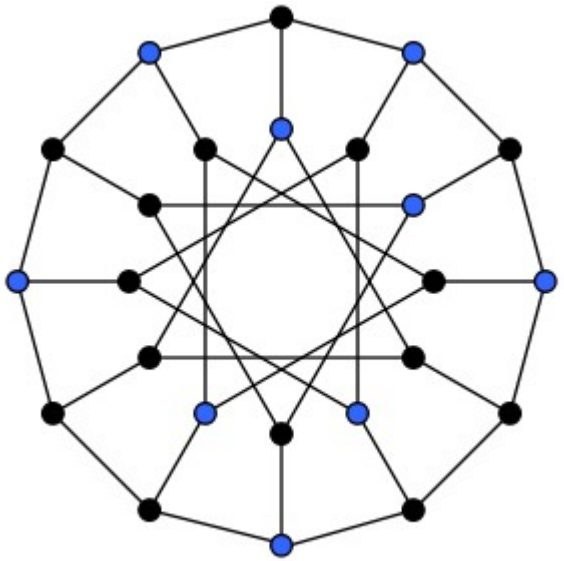
Arbre de recherche pour la coloration de 5 sommets en 2 couleurs

Méthodes par énumération implicite
stable ou clique maximale

- G = (X, U)
  - Stable: ensemble de sommets non connectés entre eux
  - Clique: ensemble de sommets tous connectés entre eux

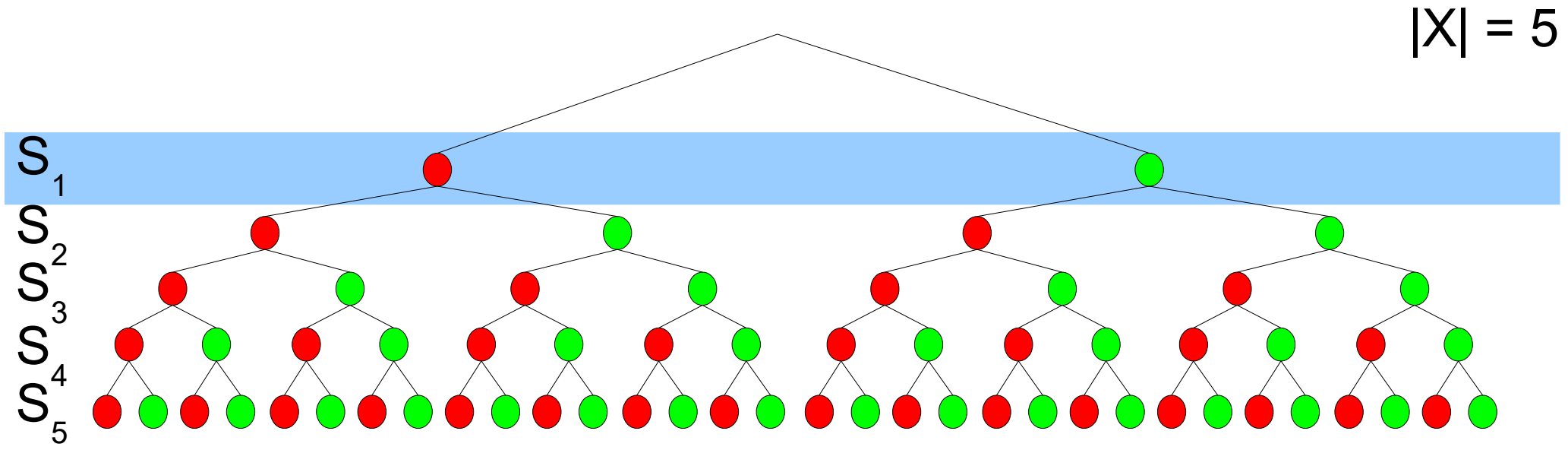
  $\rightarrow$ clique(G) $\leftrightarrow$ stable(G' = (X, $\overline{U}$))





Méthodes par énumération implicite
arbre

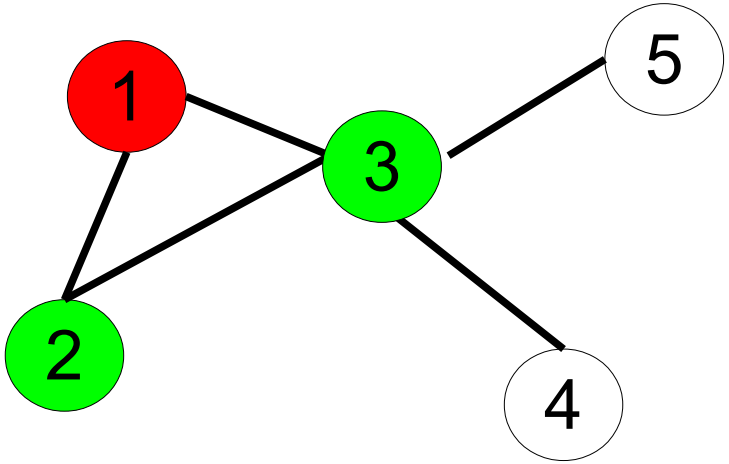
- Le sommet appartient il au stable ?
  - Oui
  - Non

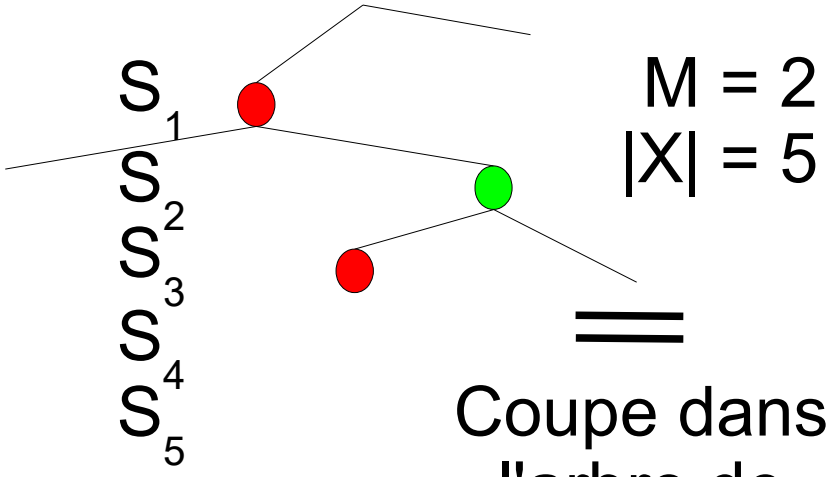


Arbre de recherche

Méthodes par énumération implicite
coupe

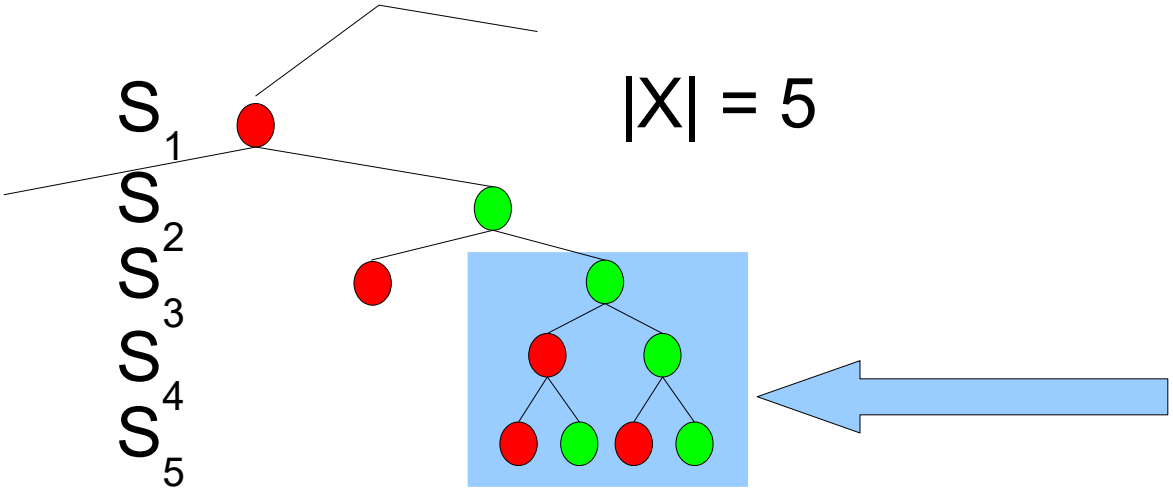
- Le sommet appartient il au stable ? Oui Non





M = 2
|X| = 5

Coupe dans l'arbre de recherche



|X| = 5

Solutions énumérées implicitement

Méthodes par énumération implicite
algorithme

- S[i] = 0/1/2/3 → ND / Oui / Non / terminé
- G = (X, U)

$\forall S_i \in X$, S[i] ← 0
i ← 1
fin ← faux

tantQue $\overline{fin}$ faire
    S[i] ← S[i] + 1
    si S[i] = 1 et
        ∃ j∈voisins(i)
            t.q S[j] = 1
    alors S[i] ← 2

    si (S[i] ≤ 2) et (i < |X|)
        i ← i + 1

    si (S[i] ≤ 2) et (i = |X|)
        afficher(S)

    si (S[i] = 3) et (i < |X|)
        S[i] ← 0
        i ← i - 1

    si (S[i] = 3) et (i = |X|)
        fin ← vrai
fin tantQue

Avancer dans l'arbre

Une solution trouvée

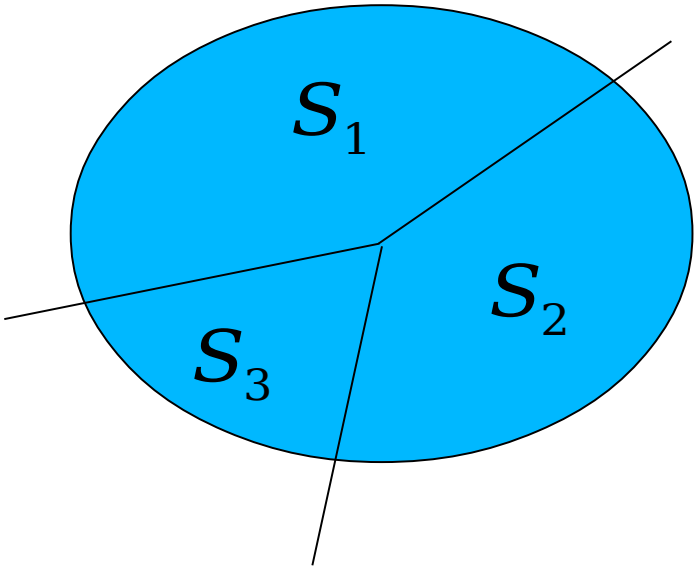
Non coloriable

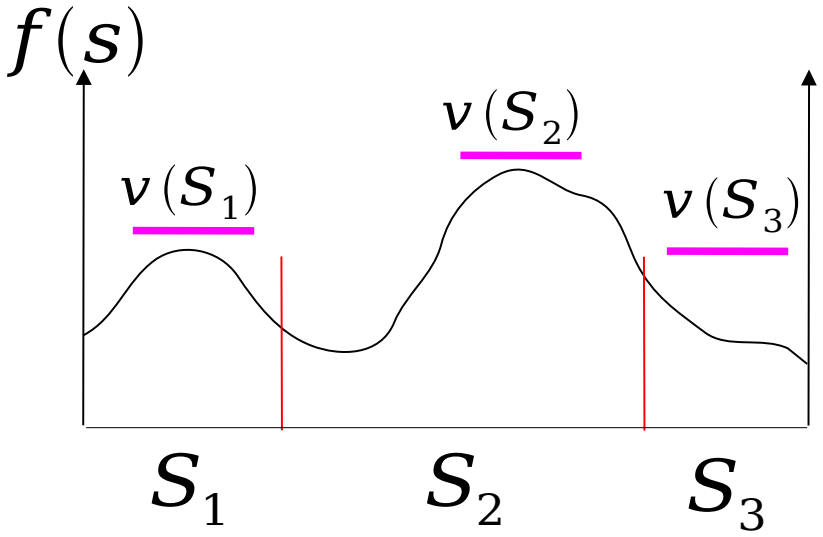
Retour à la racine

2b

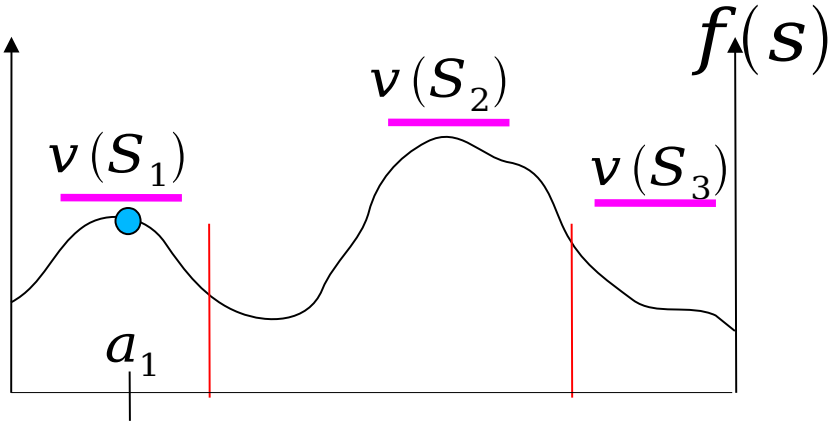
Méthodes par séparation et évaluation
principe B&B

- Enumération implicite de toutes les solutions
  - Sous espaces de recherche S_1, S_2, ..., S_n
  - *Qualité* pour chaque sous espace $v(S_{1)}$, $v(S_{2)}$, ..., $v(S_n)$





Méthodes par séparation et évaluation
principe B&B



- Supposons
  - f(s) à **maximiser**
  - Une solution réalisable a_1 connue, avec $v(a_1) = 12$
- $v(S_3) = 11$ implique que il n'est pas nécessaire d'explorer S_3
- $v(S_i)$ est une **borne supérieure** sur la qualité des solutions réalisables de S_i

Méthodes par séparation et évaluation
algorithme B&B

- Il faut, en plus de la fonction objective f(s) (*max*)
  - Une fonction de partitionnement d'un espace S en sous espaces

branch

$$S_1, S_2, \ldots, S_n : S = \bigcup_n S_i$$

bound

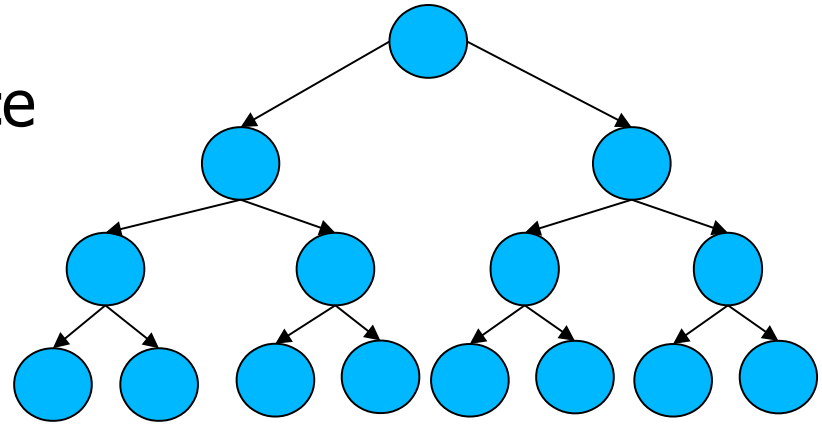
- Une fonction d'évaluation de qualité pour un espace donnant une borne supérieure sur S :

$$f : \quad \forall s \in S \;\; f(s^*) \leq v(S)$$

  - Une stratégie de recherche pour choisir quel sous espace S_i évaluer à l'itération suivante.

Méthodes par séparation et évaluation
arbre de recherche

- Le déroulement de l'algorithme correspond à l'exploration d'un arbre de recherche :
  - Chaque noeud correspond à un sous espace
  - Les feuilles sont des solutions ou des espaces sans solution réalisable
  - Les fils S_1, S_2, ..., S_n de S résultent du partitionnement de S



Méthodes par séparation et évaluation
algorithme

Espace S,
Évaluation v(S)(*min*),
séparer()
suivant()
liste sommets L = {S}
Borne initiale U = ∞
Meilleure solution

$S_{best} = \infty$

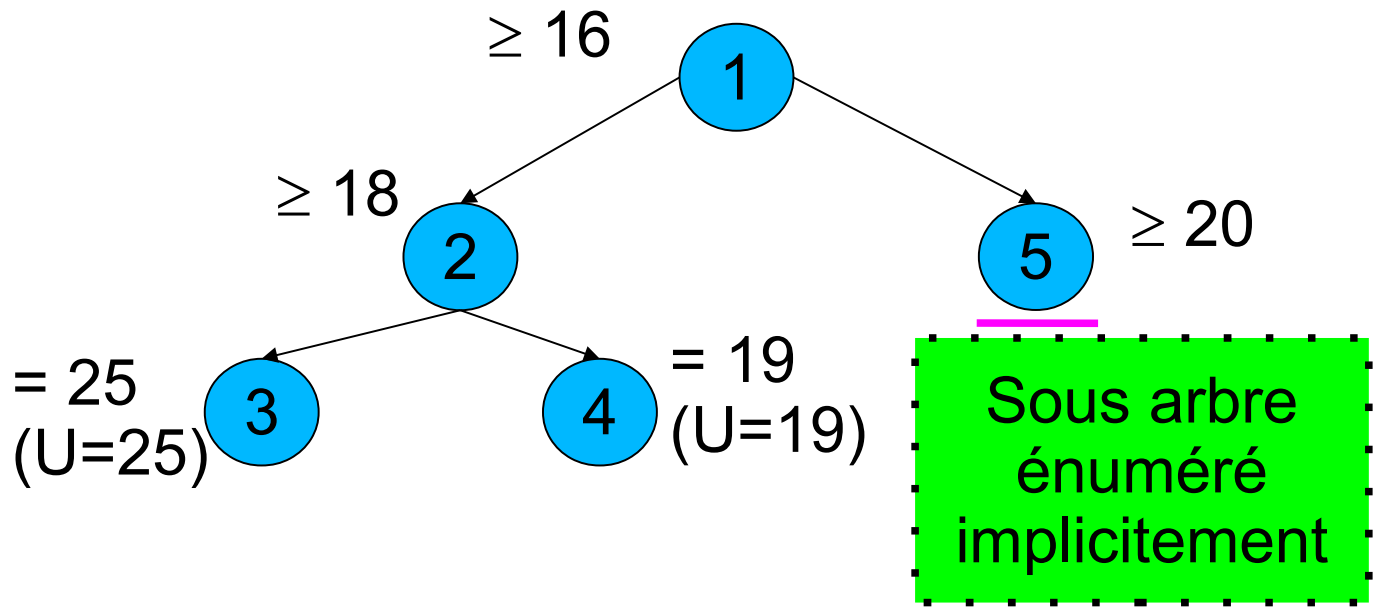
Algo B&B
tant que $L \neq \varnothing$ **faire**
S = suivant(L)
$S_1, S_2, \ldots, S_n$ = séparer(S)

pour chaque S_i **faire**
si $v(S_i) > U$ ou S_i non réalisable
éliminer S_i
sinon si S_i réalisable
$S_{best} = S_i$
$U = v(S_i)$ $(= f(S_i))$
sinon
$L = L \cup \{S_i\}$

fin

Méthodes par séparation et évaluation
coupes

- Coupes dans l'arbre de recherche si $v(S_i) > U$ (si *min*)
  - Énumération implicite



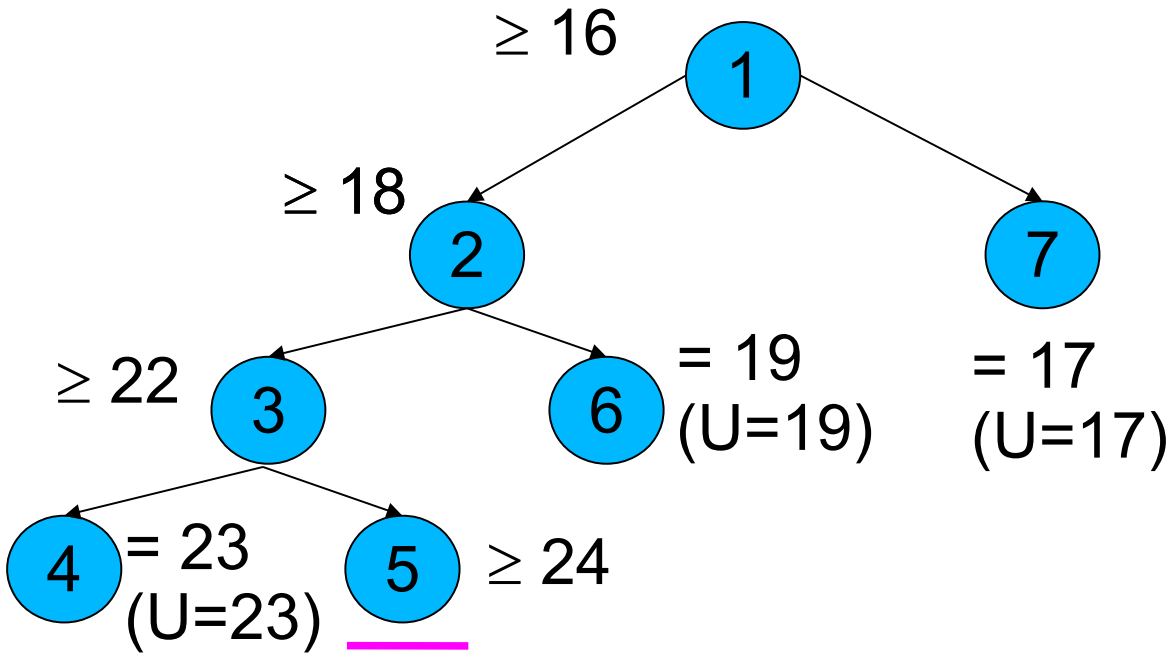
Méthodes par séparation et évaluation
parallélisation

- Coupes dans l'arbre de recherche si $v(S_i) > U$
  - Énumération implicite

- Parallélisation possible
  - Anomalies favorables ou défavorables suivant la fonction d'évaluation et la stratégie de parcours
  - Mise en oeuvre centralisée ou distribuée
  - Problème mise à jour de U

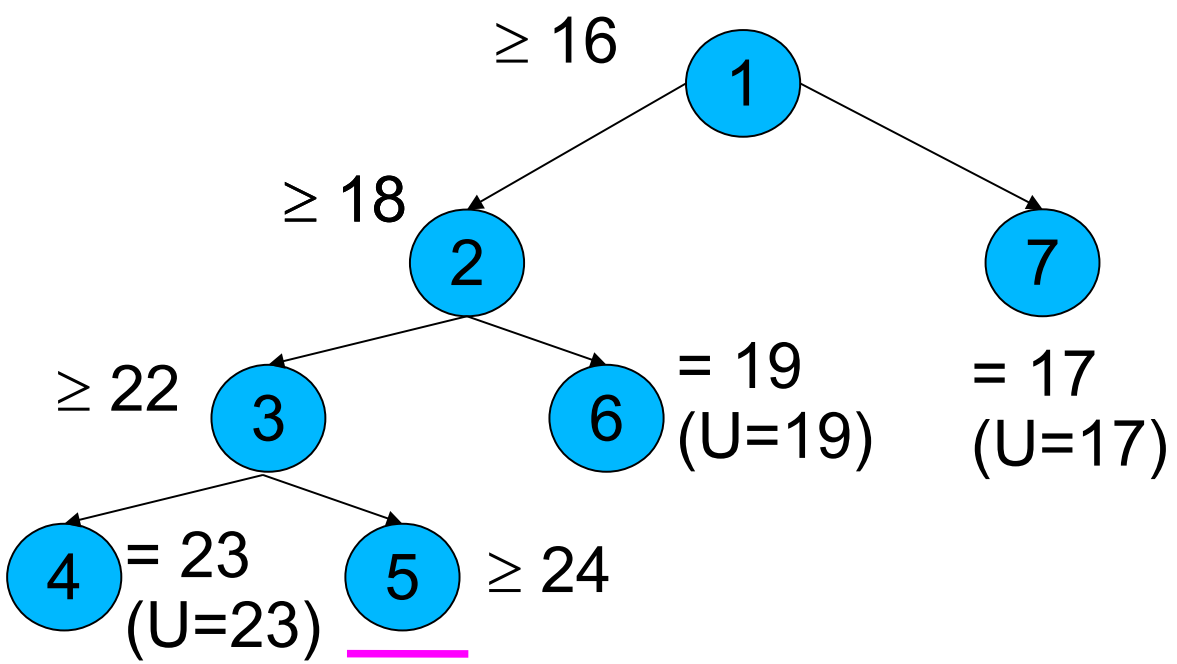
Méthodes par séparation et évaluation
accélération et parallélisation

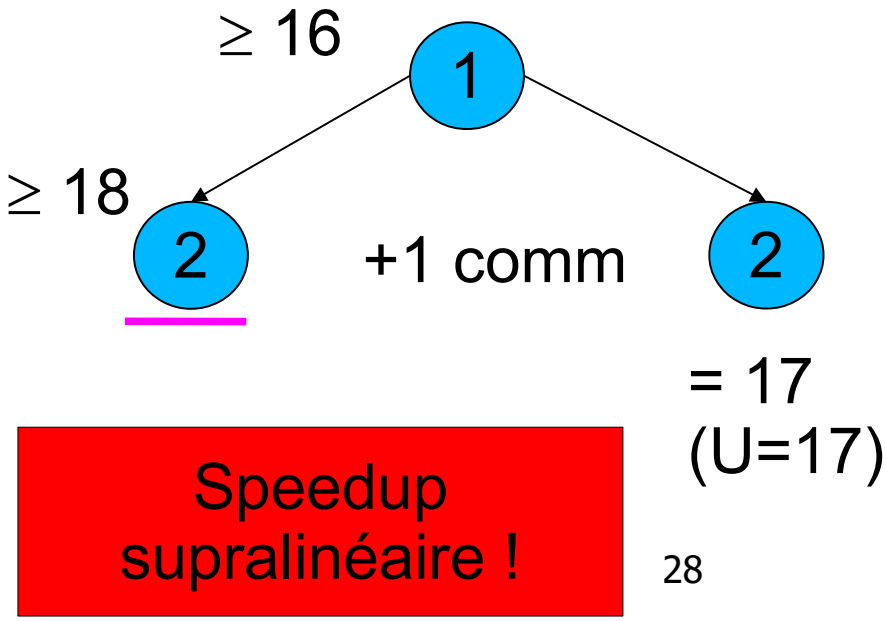
- Exemple avec mise en oeuvre centralisée
  - 1 maitre, 2 esclaves, comms 1 unité de temps en // ...
  - Parcours profondeur d'abord de l'arbre de recherche
    7 unités de temps en séquentiel



Méthodes par séparation et évaluation
accélération et parallélisation

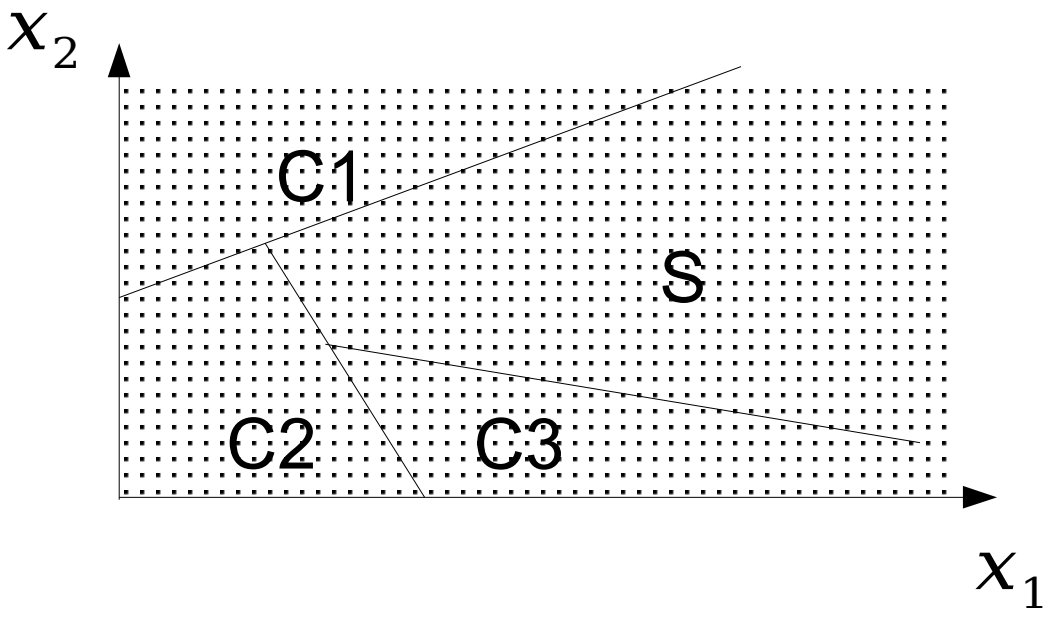
- Exemple avec mise en oeuvre centralisée
  - 1 maitre, 2 esclaves, comms 1 unité de temps en // ...
  - Parcours profondeur d'abord de l'arbre de recherche
    7 unités de temps en séquentiel
    2 unités de temps en // sur 2 processeurs





Exemples d'utilisation B&B
application à la programmation linéraire

- Le simplexe donne des valeurs dans $\mathbb{R}$
  - Comment résoudre un problème dans $\mathbb{N}$?



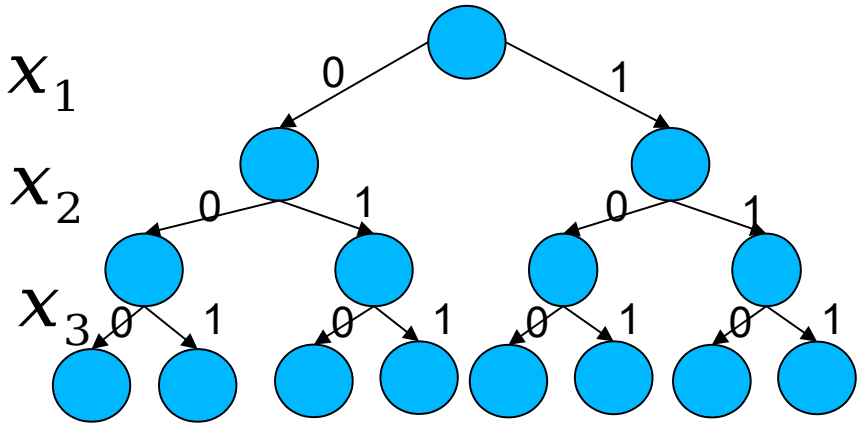
$$x_1 + 3x_2 \leqslant 15$$
$$-2x_1 - x_2 \leqslant -12$$
$$-3x_1 - 11x_2 \leqslant -66$$
$$min \quad 2x_1 + 4x_2$$
$$x_1, x_2 \in \mathbb{R}$$

$$x_1, x_2 \in \mathbb{N}$$

Exemples d'utilisation B&B
application à la programmation linéraire

- $v(S_{\mathbb{R}}) \leq v(S_{\mathbb{N}})$ car $\mathbb{N} \subset \mathbb{R}$

  - Le simplexe fournit une borne pour la solution dans $\mathbb{N}$
- On sépare sur les variables
  - bivalentes (*0-1 IP*)
  - ou dans un intervalle entier quelconque (*IP*)

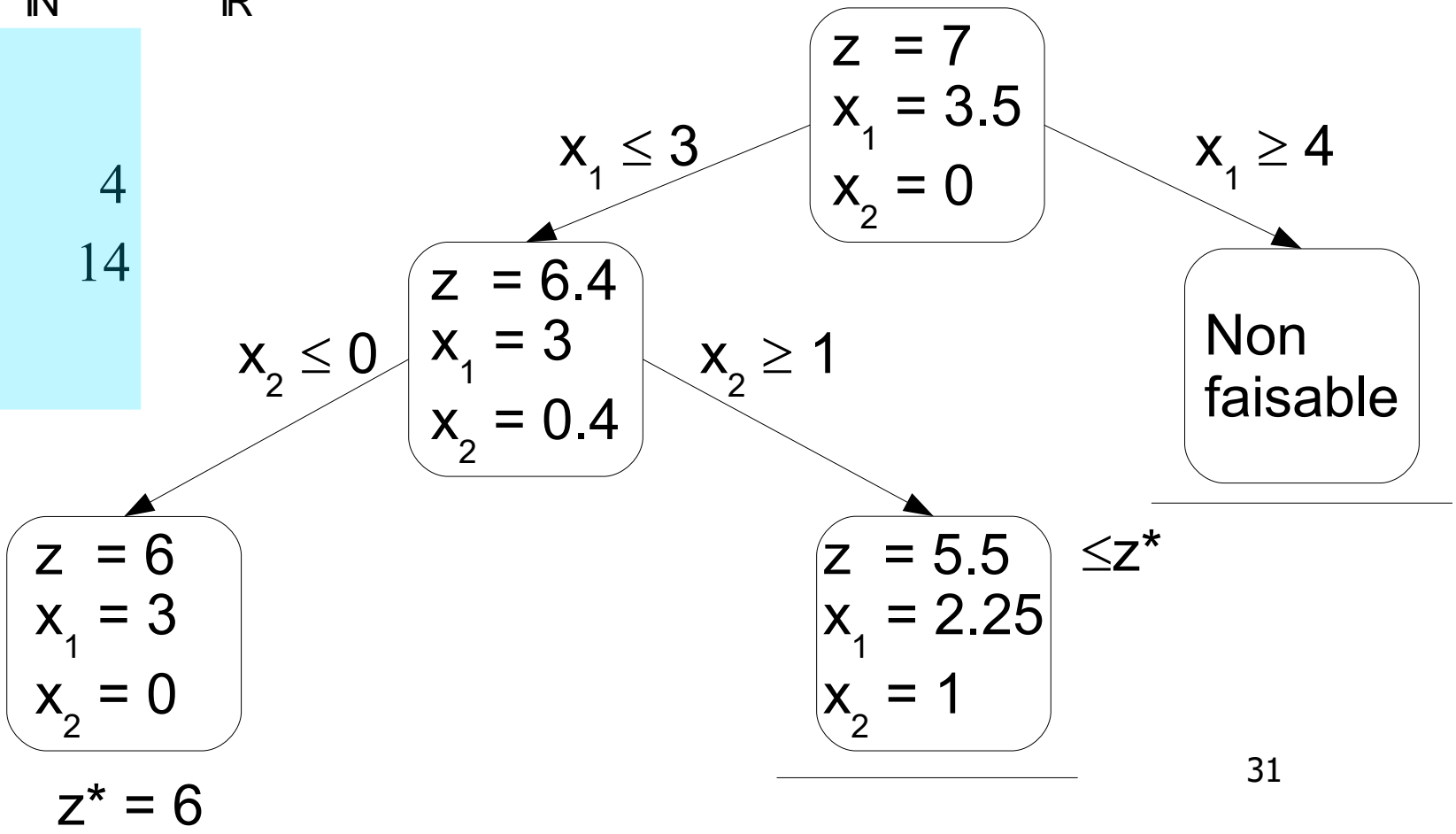


Programmation Linéaire
exemple

- Solution dans $\mathbb{N}$, B&B + simplexe (solution dans $\mathbb{R}$)
- Pour z max, $v(S_{\mathbb{N}}) \leq v(S_{\mathbb{R}})$ car $\mathbb{N} \subset \mathbb{R}$

$$max \quad 2x_1 + x_2$$

$$s.t. \begin{cases} x_1 - x_2 \leqslant 4 \\ 4x_1 + 5x_2 \leqslant 14 \\ x_1 \geqslant 0, \quad x_2 \geqslant 0 \end{cases}$$



Exemples d'utilisation B&B
problème du voyageur de commerce

- Première utilisation du principe de séparation et évaluation(algorithme de Little)

- TSP asymétrique problème P

	A	B	C	D	E	F
A	∞	1	7	3	14	2
B	3	∞	6	9	1	24
C	6	14	∞	3	7	3
D	2	3	5	∞	9	11
E	15	7	11	2	∞	4
F	20	5	13	4	18	∞

Exemple : Algorithme et Complexité #4 – c 2002 – Michel Van Caneghem

Exemples d'utilisation B&B
évaluation

- On soustrait de chaque ligne son minimum puis idem pour les colonnes : problème P_2

- Constantes
  - Ne changent pas l'ordre des solutions
  - $z^*(P) = z^*(P_2) + d$

    $d = \sum_{lignes} \min(l) + \sum_{colonnes} \min(c)$

    $d = 16 + 3$

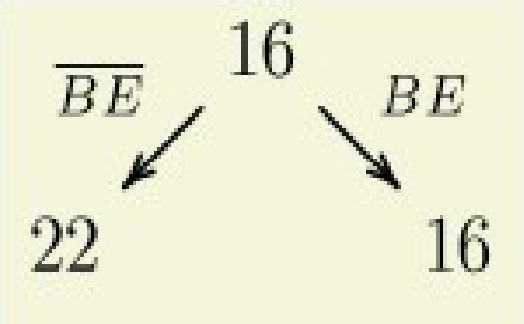
	A	B	C	D	E	F
A	∞	0	3	2	13	1
B	2	∞	2	8	0	23
C	3	11	∞	0	4	0
D	0	1	0	∞	7	9
E	13	5	6	0	∞	2
F	16	1	6	0	14	∞

d est un minorant

Exemples d'utilisation B&B
séparation

- Séparation : favoriser les tournées les - chères : arcs à 0.

- Surcoût si l'arc est exclu : partir de son sommet initial et arriver à son sommet final (min ligne + min col) … choix du plus coûteux en espérant le couper ensuite



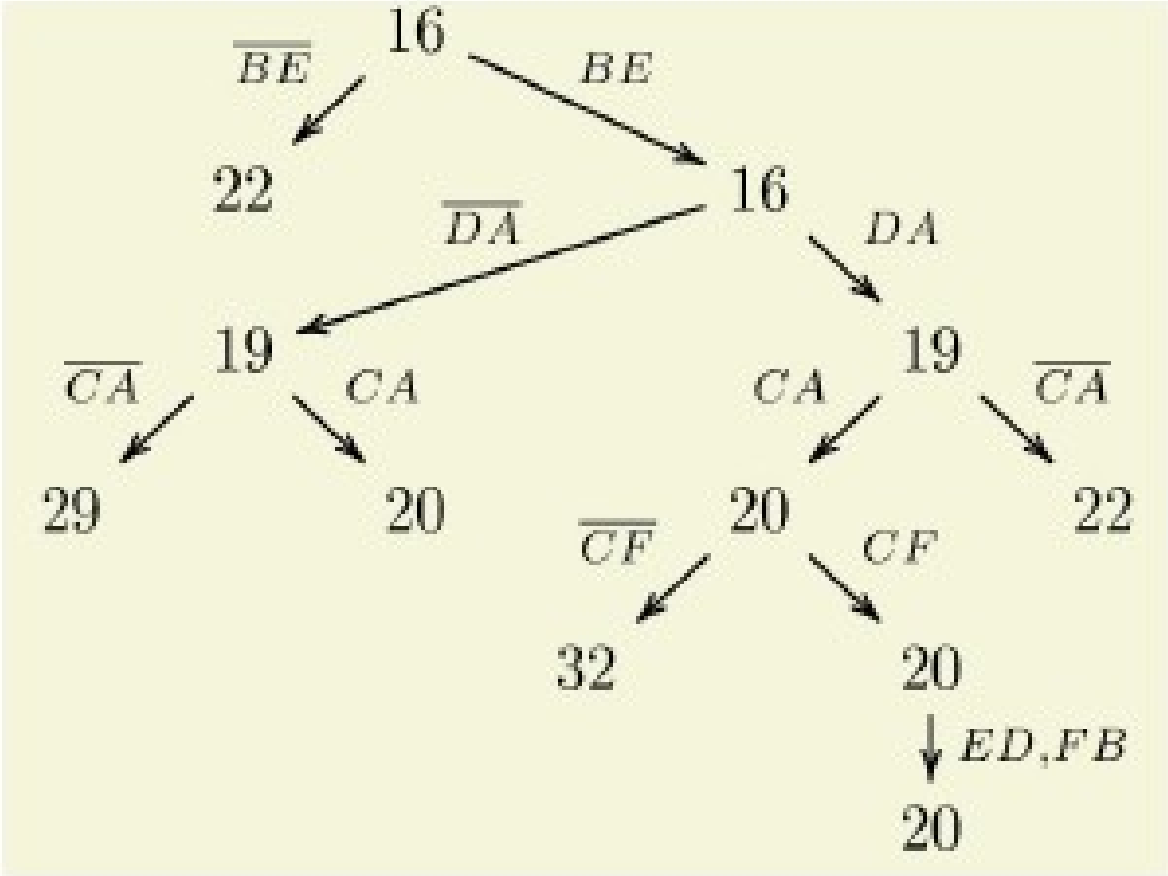
	A	B	C	D	E	F
A	∞	0(2)	3	2	13	1
B	2	∞	2	8	0(6)	23
C	3	11	∞	0(0)	4	0(1)
D	0(2)	1	0(2)	∞	7	9
E	13	5	6	0(2)	∞	2
F	16	1	6	0(1)	14	∞

Exemples d'utilisation B&B
itération

- Suppression de la ligne et la colonne pour la branche positive, ∞ pour l'arc retour (sous-tournée)
- ∞ pour l'arc pour la branche négative

	A	B	C	D	F
A	∞	0(2)	3	2	1
C	3	11	∞	0(0)	0(1)
D	0(3)	1	0(3)	∞	9
E	13	∞	6	0(2)	2
F	16	1	6	0(1)	∞

Arbre de recherche complet



Exemples d'utilisation B&B
une autre borne (arbres)

- Trouver une tournée $t \in T$ de longueur minimale dans un graphe quelconque G = (V, E)
  - Trouver un arbre $a \in A$ de recouvrement minimum sur le graphe G' = (V', E) $\quad V' = V \setminus \{V_0\}$
  - Relier V_0 à a pour former $a^{*'} \in A'$

- Toute tournée t privée d'une arête est un arbre, donc :
  - $T \subseteq A'$ et $\forall\, t \in T,\ v(a^{*'}) \leq v(t)$

  - L'algorithme de calcul de $a^{*'}$ permet de fournir un minorant sur un sous espace de T

Exemples d'utilisation B&B
P-Median

- Placer p villes par rapport à une population demandeuse. Minimiser la distance des demandeurs aux point de service le plus proche :

- h_i is the weight of the demand of the customer i,
- d_{ij} is the distance between customer i and facility j,
- X_i is a decision variable, indicating if the facility i is selected or not,
- Y_{ij} is a decision variable, indicating if the customer i is served by facility j or not,
- p is the number of facilities to be selected.

$$\text{minimize} \sum_{i=1}^{D}\sum_{j=1}^{N} h_i d_{ij} Y_{ij} \qquad (1)$$

$$\text{subject to} \sum_{j=1}^{N} Y_{ij} = 1 \qquad \forall i, 1 \leq i \leq D \qquad (2)$$

$$\sum_{j=1}^{N} X_j = p \qquad (3)$$

$$Y_{ij} - X_j \leq 0 \qquad \forall i, j,\ 1 \leq i \leq D,\ 1 \leq j \leq N \qquad (4)$$

$$X_j \in \{0,1\} \qquad \forall j,\ 1 \leq j \leq N \qquad (5)$$

$$Y_{ij} \in \{0,1\} \qquad \forall i, j,\ 1 \leq i \leq D,\ 1 \leq j \leq N \qquad (6)$$

Exemples d'utilisation B&B
P-Median

- Jeu de tests jusqu'à 1000 sommets :

Table 1. Results for the Beasley benchmark (40 graphs)

	CPlex	SA	Genetic algorithms			Volume
			basic-GA	hyper-GA	imp-GA	
			1000 iterations	1000 iterations	100 iterations	
# optimal	40	11	20	20	23	5
Deviation in %	0.0	2.35	0.1	0.14	0.2	4.2
Total time (secs)	64329	2294	70	5246	4862	112

- Cas réel 188K demandes, 2K points candidats :
  - Impossible avec CPLEX

    → heuristiques

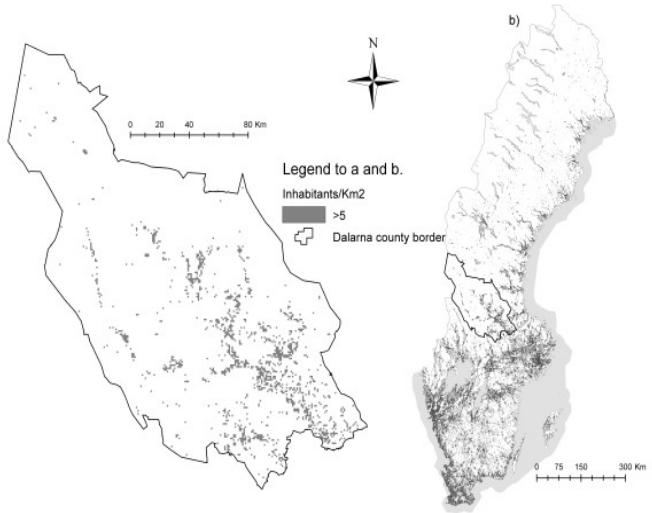


Fig. 1. Distribution of two populations, Dalecarlia province on the left, Sweden on the right

Diviser Pour Régner
GAP pour IoT (internet des objets)

- Stocker les données dans le réseau. Choisir pour chaque donnée un nœud S parmi les nœuds de stockage possibles (stores, avec leur capacité)
  - A partir du nœud où elles sont générées (production P)
  - En tenant compte du lieu où elles seront utilisées (consommation C)
- Transfert par le chemin le plus rapide à chaque fois

- Données du problème : liste Data (avec nœud P et C), Liste Stores, réseau d'interconnnexion, avec latence des liens → liste des stockages S des données
- GAP : *Generalized Assignment Problem*

Diviser pour régner
GAP pour IoT (internet des objets)

- Résolution mathématique très longue (GAP)
  → milliers d'équations si nombreux capteurs/données

Latence de Pi à Sj + de Sj à Ci

Data i sur nœud j ?

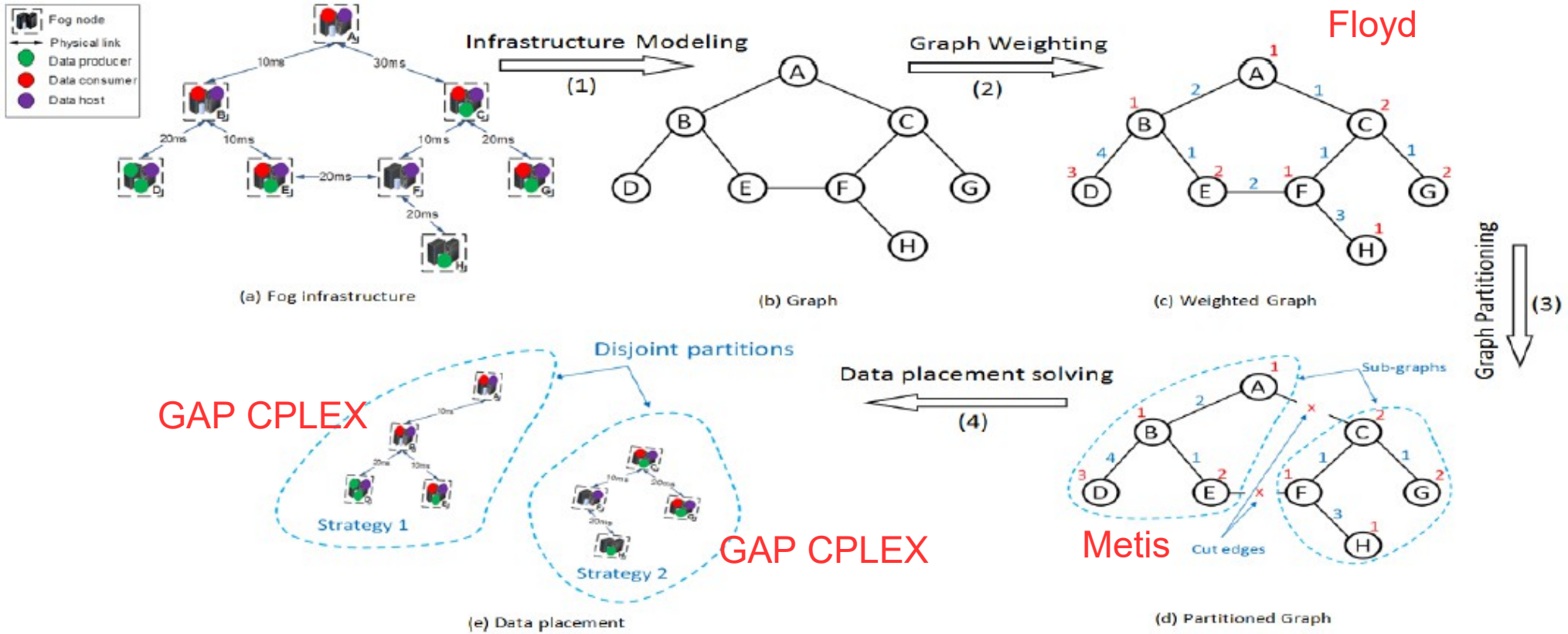
$$
\begin{cases}
Minimize & \sum_{i\in[1..l]} \sum_{j\in[1..n]} \alpha_{i,j}.(x_{i,j}) & \\
Subject\ to & & \\
 & \sum_{i\in[1..l]} s_{d_i}.x_{i,j} \leq f_{dh_j} & \forall j \in J = [1..n] \\
 & \sum_{j\in[1..n]} x_{i,j} = 1 & \forall i \in I = [1..l] \\
 & x_{i,j} \in \{0,1\} & \forall i \in I, \forall j \in J
\end{cases}
$$

Capacité des nœuds de stockage

Data sur un et un seul noeud

Diviser pour régner
GAP pour IoT (internet des objets)

- Diviser pour Régner : sous-problèmes résolus indépendamment
- Floyd, Metis et CPLEX peuvent être parallélisés

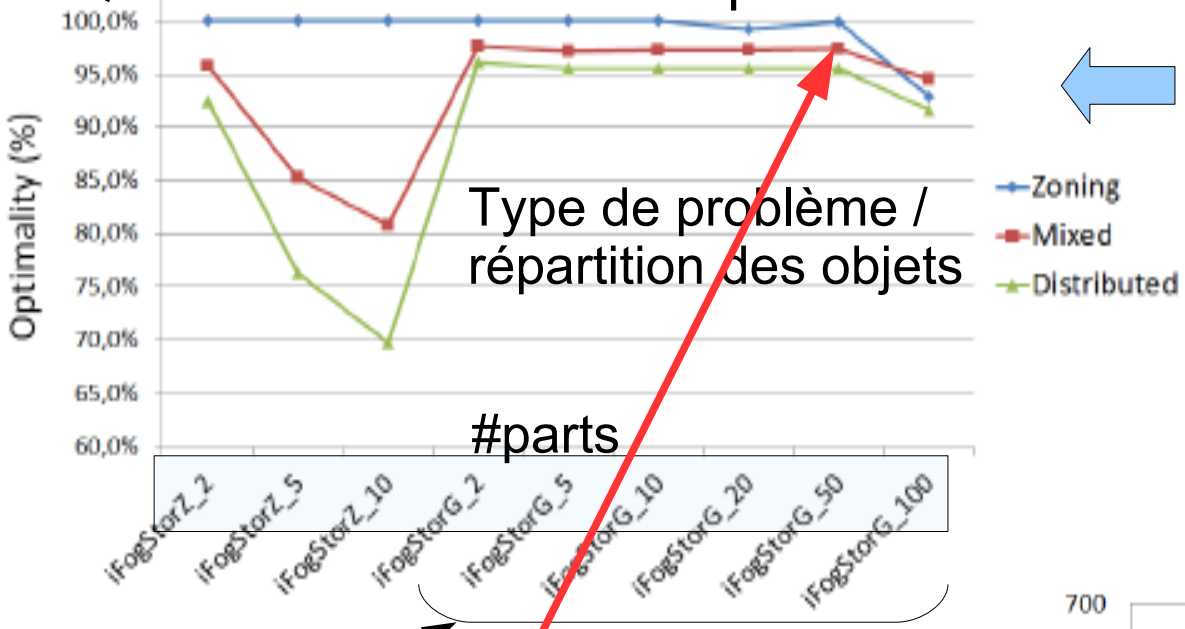


Diviser pour régner
GAP pour IoT (internet des objets)

- Construire le graphe qui correspond au réseau
- A partir des caractéristiques du réseau, calculer tous les temps de transferts entre nœuds (latences)
  → plus court chemins de tous vers tous (Floyd)
- Compter pour chaque lien le nombre de routes (PCC) P → S et S → C qui l'utilise :
  → poids des arêtes du problème de partitionnement
- Compter pour chaque nœud du réseau le nombre de P, C, S qu'il accueille
  → poids des sommets du problème de partitionnement
- Partitionner le réseau (Metis) et résoudre les GAP (CPLEX) indépendamment
- Les données sont placées

Diviser pour régner
GAP pour IoT (internet des objets)

Qualité : à combien de l'optimum ?



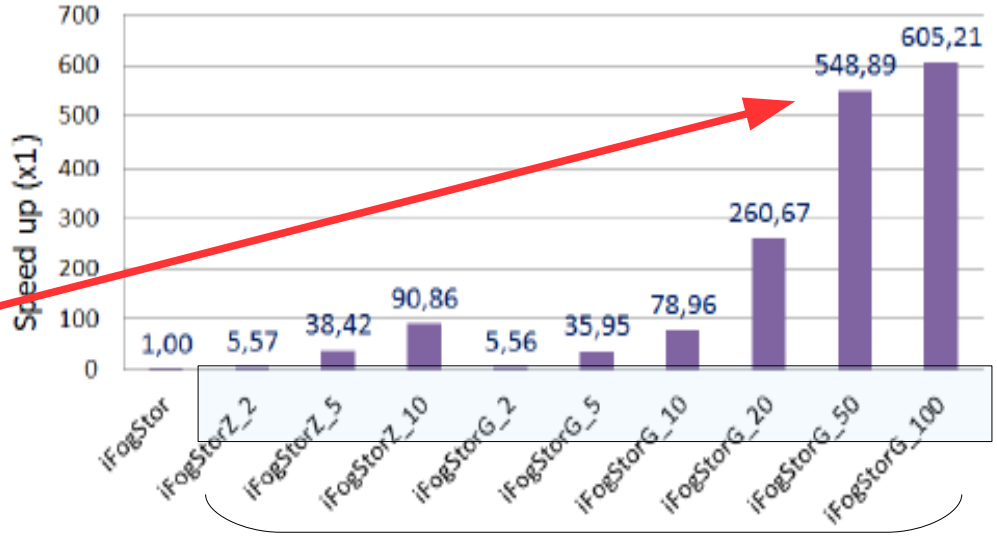
Type de problème / répartition des objets

#parts

Partitionnement de graphe plus efficace que partitionnement géographique

Avec partitionnement de graphe

97% de l'optimum
550 x plus vite



Avec partitionnement

3

Optimisation combinatoire
méthodes heuristiques

- Donnent un résultat approché
  - Parfois seule technique connue (ex: optimisation de programmes)
  - Ou méthodes exactes uniquement pour modèle approximatif (ex: test de circuit)
- Adaptées si
  - Explosion combinatoire
  - Objectifs multiples
  - Rapidité prime sur performance

Optimisation combinatoire
Algorithmes gloutons

- *Greedy* : construisent une solution pas à pas, sans jamais revenir sur les choix effectués
  - Souvent loin de l'optimum
- Très rapides
  - Améliorables par recherche locale
- Exemples
  - Sac à dos
  - Recouvrement
  - Stable maximum
  - Voyageur de commerce

Algorithmes gloutons
sac à dos

- *knapsack* : remplir un sac à dos de poids limité en choisissant parmi une liste d'objets ceux les plus intéressants
- Formulation linéaire
  - Interpréter les variables

    $a_1x_1 + a_2x_2 \ldots + a_nx_n \leq b$

    max $c_1x_1 + c_2x_2 \ldots + c_nx_n$

    $x_j \leq \beta_j \qquad X_j \in \mathbb{N}$

- Classer par profit $c_1 / a_1 \leq c_2 / a_2 \ldots \leq c_n / a_n$
- Poser $x_1 = \min(\beta_1, b/a_1)$; $b = b - a_1x_1$
- Recommencer sur $x_2, x_3, \ldots, x_n$

Algorithmes gloutons
recouvrement

- Couvrir n éléments par au moins un parmi m objets, chacun avec un coût donné. Minimiser le coût total.
- A, matrice de couverture. Interpréter les variables A_i^j

  $A_i^j \geq 1$

  $min \;\; c_1x_1 + c_2x_2 + \ldots + c_mx_m$

  $x_j \leq 1 \quad$ et $\quad x_j \in \mathbb{N}$

- Trouver k t.q $c_k/a_j = min_{\; j \in 1..n} \; c_j/a_j \qquad$ avec $\; a_j = \sum_{i=1..m} A_i^j$
- Eliminer la colonne k et toutes les lignes tq $A_i^k = 1$
- Recommencer sur le problème réduit

Algorithmes gloutons
voyageur de commerce

- Algorithme du plus proche voisin
  - Première ville choisie au hasard
  - Chemin vers la ville suivante la plus proche non déjà incluse dans la tournée
  - Rebouclage entre la dernière et la première ville

Algorithmes gloutons
coloration de graphe

- Algorithme de base
  - Coloriser les sommets dans un ordre quelconque
  - Utiliser une des couleurs existantes si possibles
  - Créer une nouvelle couleur si besoin
- Amélioration
  - Trier d'abord les sommets par degré décroissant
    → sommets à contraintes fortes coloriés d'abord

Algorithmes gloutons
stable maximal d'un graphe

- Inclusion itérative de sommets

stable(G = (X, U))

$E \leftarrow \varnothing$

$S \leftarrow \varnothing$

TQ $\exists s \in X,\ s \notin E$

$S \leftarrow S \cup \{ s \}$

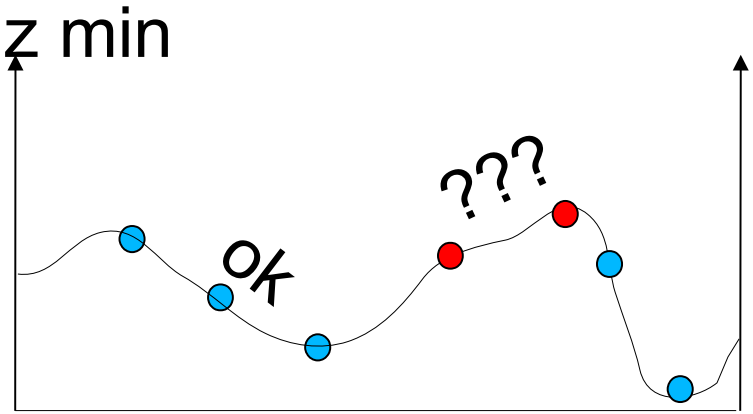
$E \leftarrow E \cup \{ s \}$

$\forall s' \in$ voisins(s)

$E \leftarrow E \cup \{ s' \}$

Méthodes par voisinage (recherche locale)
principe

- Problème : sortir des extrema locaux
  - Exploration aléatoire, même si coûteuse
  - Il faut finir par converger
- Evolution d'une solution
  - Descente simple
  - Recuit simulé
  - Méthode tabou
- Ou de plusieurs solutions
  - Algorithmes génétiques
  - Fourmis



Méthodes par voisinage
voisinage

- Définir une fonction de voisinage $V : S \rightarrow S^n$
  - Fournit un ensemble de n solutions similaires à $s \in S$
  - Explorer ces n solutions pour en trouver une meilleure que s

- Algorithme pas forcement polynomial
- Problème d'optimalité

- Faire de l'exploration à partir de plusieurs solutions générées aléatoirement

Méthodes par voisinage
descente simple

- Fonction objective *min* $f : S \rightarrow \mathbb{R}$
- A partir d'une fonction de voisinage $V : S \rightarrow S^n$
  - Fournit un ensemble de n solutions similaires à $s \in S$

génerer une solution initiale s_0

$s = s_0$

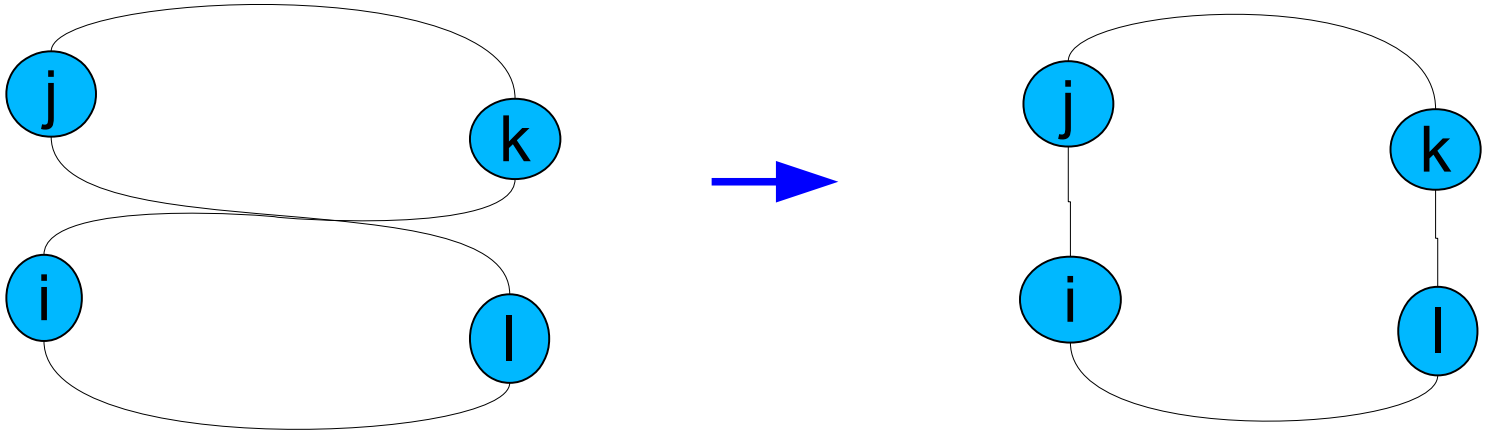
Tant que non fin

$s' = \min_{s \in V(s)} f(s)$

Exploration locale
problème du voyageur de commerce

- Exemple : *2-opt* pour TSP (Lin, 1965, n(n - 3)/2)
  - Voisinage de n^2 tournées

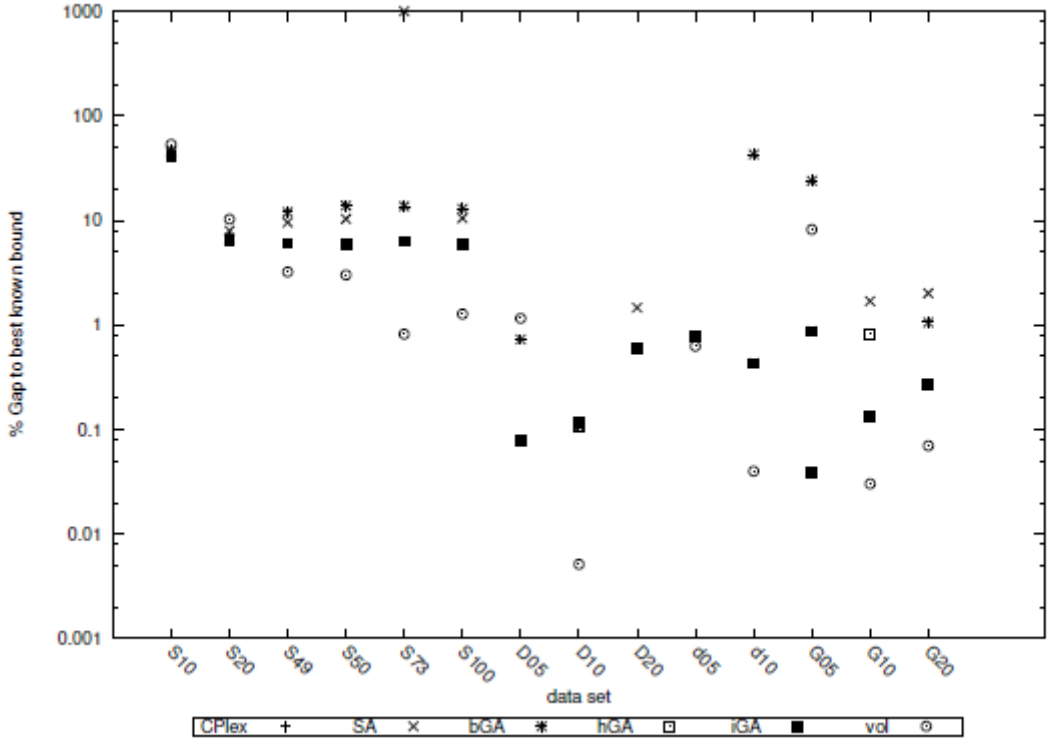
    $T' = T \cup \{ i \rightarrow k, j \rightarrow l \} \setminus \{ i \rightarrow j, k \rightarrow l \}$



- *3-opt* possibles, mais voisinage trop large n(n – 3)(n - 2)

Exploration locale
PMedian

- Hyper-mutation (pas une mutation!)

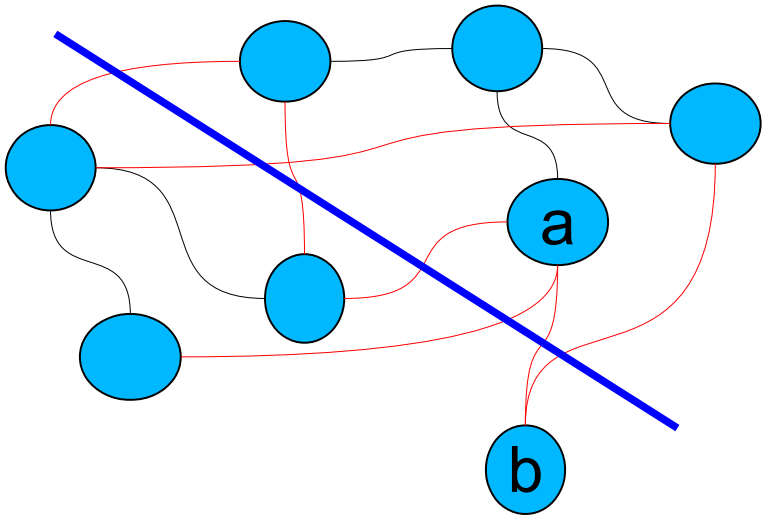



```
begin
    Let A be the chromosome;
    Choose randomly F facilities that appear in A;
    foreach facility among the P chosen nodes do
        for 0 ≤ i ≤ N do
            replace the selected facility by facility i (if i is not already in A);
            if fitness(A′) < fitness(A) then
                A ⟵ A′ ;
            end
        end
    end
end
```

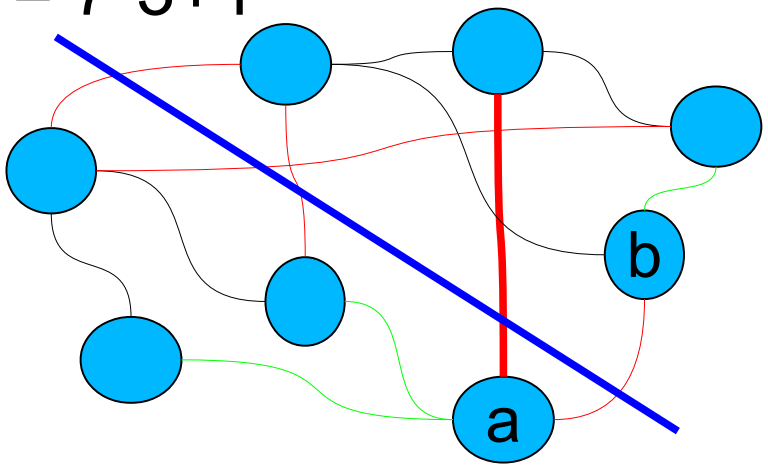
Exploration locale
problème du 2-partitionnement

- Dans un graphe $G = (X, E)$ avec $|X| = 2n$ trouver une partition $X = X_1 \cup X_2$ t.q $|X_1| = |X_2| = n$ qui minimise le nombre d'arêtes entre les parties
  - Voisinage par échange de paires

c=7

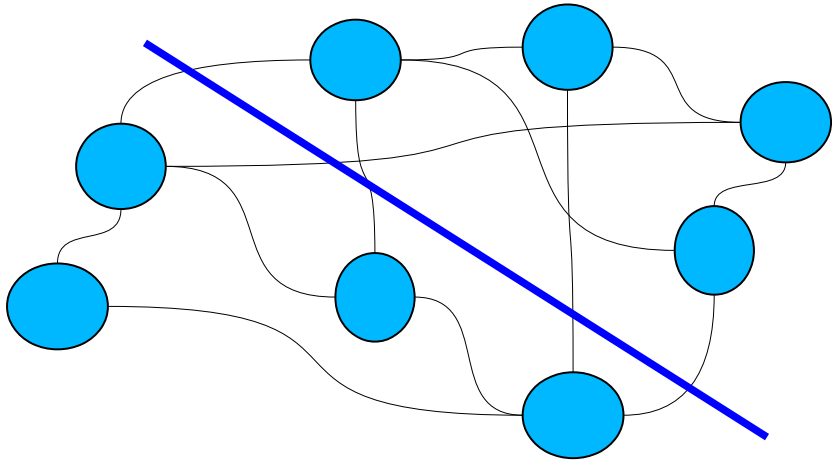


c=5 = 7-3+1



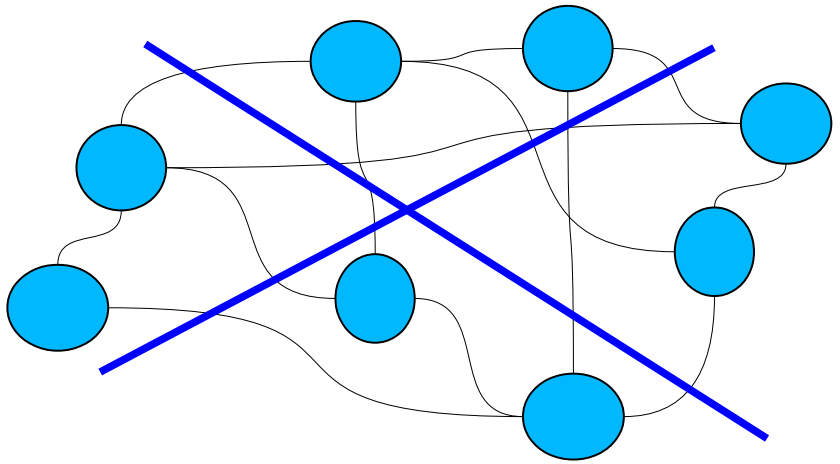
2-partitionnement
heuristique de Kernighan-Lin

- A chaque étape, choisir l'échange qui maximise le gain en terme de coupe
  - Contrainte : un sommet ne peut être échangé qu'une seule fois
  - N/2 étapes au maximum



multi-partitionnement
techniques

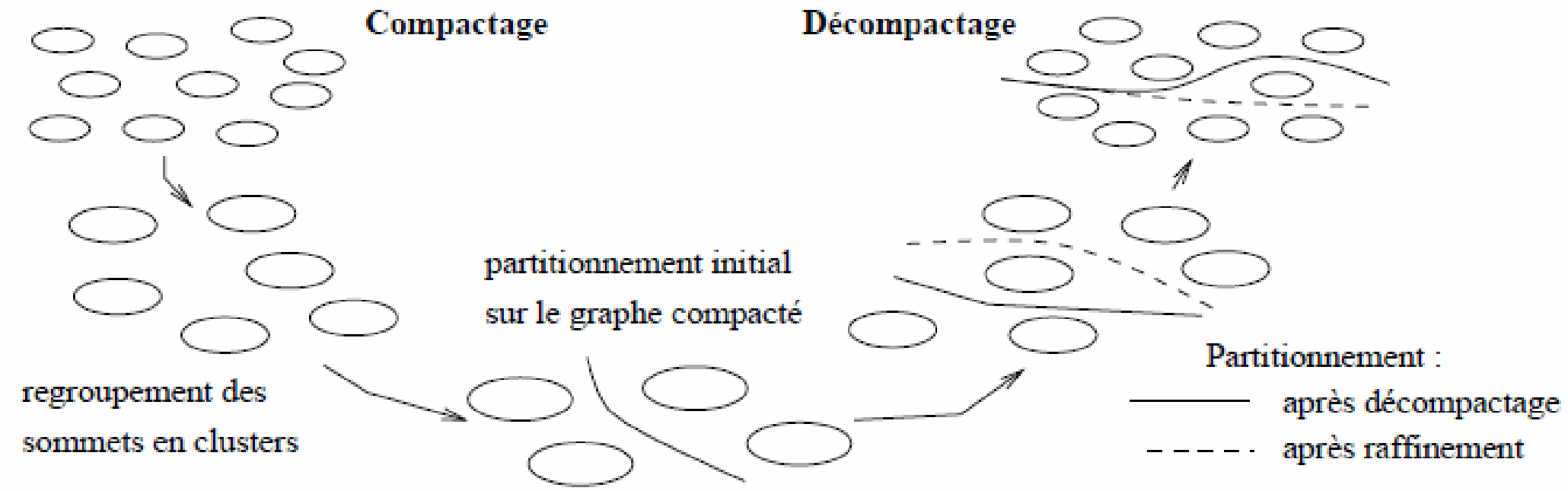
- Extensions du bi-partitionnement
  - Par récursion
  - Par adaptation directe de Kernighan Lin
- k-partitionnement :
  - Minimiser la coupe globale
  - Équilibrer la taille des parties



- Possibilités de partitionnement multi-niveaux (Métis)

multi-partitionnement
HMétis

- Partitionnement multi-niveaux



multi-partitionnement applications

- Répartition de trafic aérien
  - noeuds : avions, tours, points de virage
  - arcs : routes
  - minimiser les interactions entre contrôleurs
- Partitionnement de circuits
  - noeuds : portes logiques
  - arcs/arêtes : connexions
  - créer et optimiser des sous-systèmes

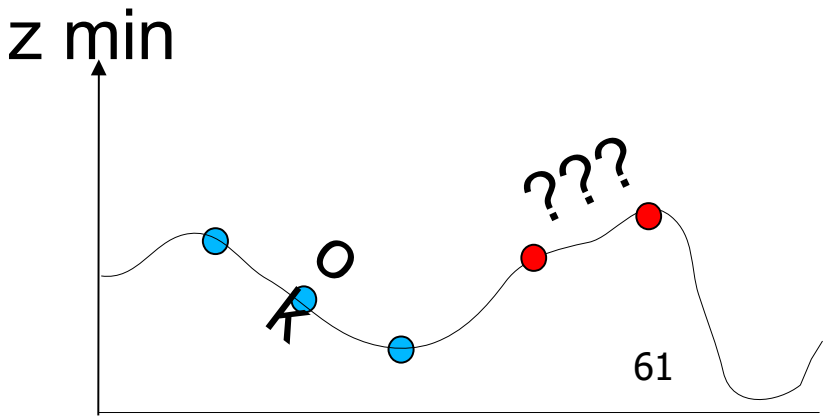
Amélioration des méthodes de descente
problématique

- Rappel : antagonisme entre
  - Améliorer une solution courante
  - Explorer suffisament l'espace de recherche

Exploitation ⟷ Exploration

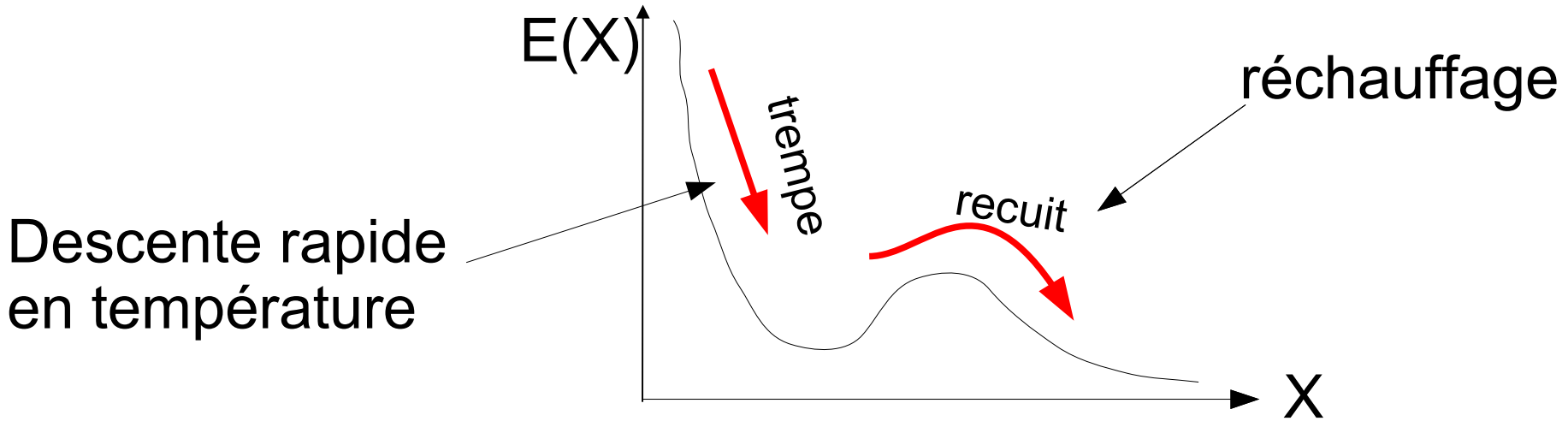
- Toujours le problème des extrema locaux

- Politique de compromis



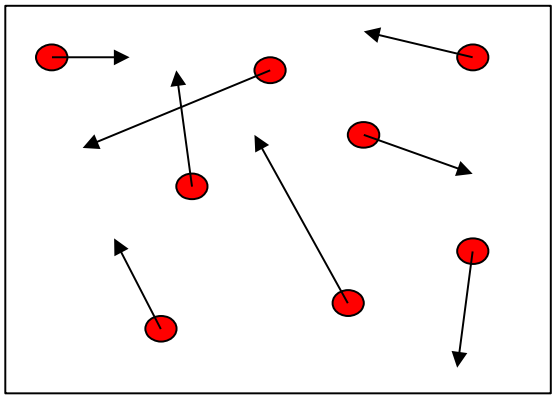
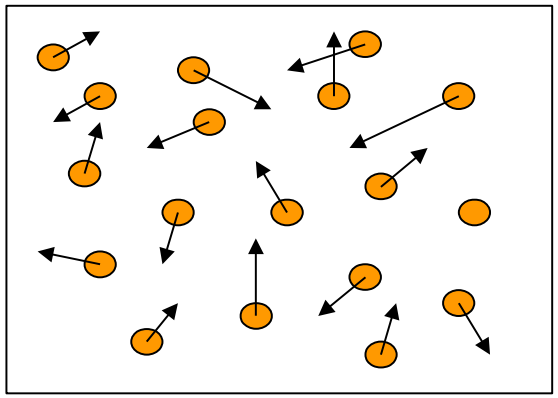
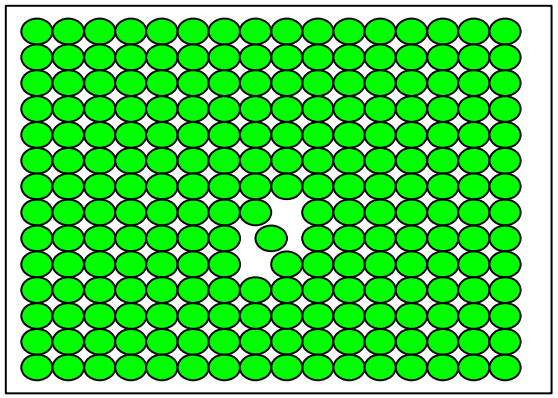
Recuit simulé
simulated annealing (SA)

- Méta heuristique
  - Principe d'évolution du compromis exploit./explor.
  - Basé sur la cuisson des métaux pour obtenir un cristal
- Evolution des niveaux d'énergie à l'intérieur du métal

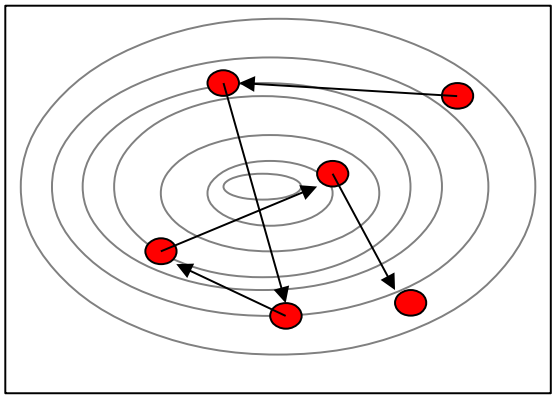
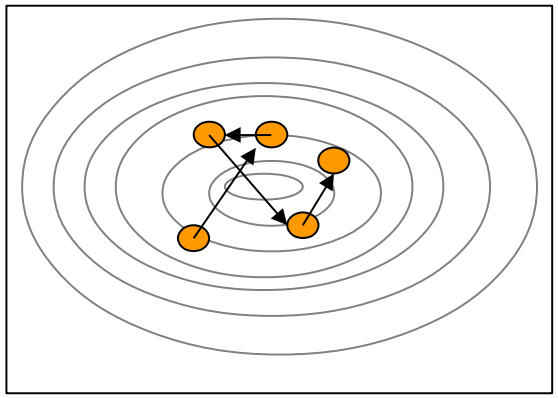
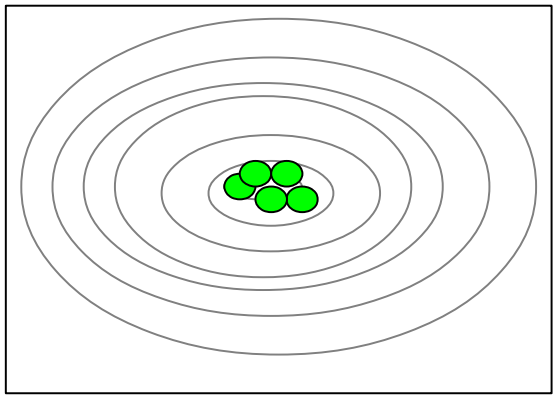


Recuit simulé
analogie physique/optimisation

physique

optimisation

Recuit simulé
principe

- Explorer au hasard l'espace de recherche
- Faire varier un degré de non déterminisme (température)
  - Haut au début : comportement très aléatoire
  - Bas à la fin : exploration de type algorithme de descente

Recuit simulé
un pas de l'algorithme principal

- A partir de la solution courante s_i, explorer le voisinage pour obtenir s_{i+1}
- Si $f(s_{i+1}) - f(s_i) < 0$, accepter s_{i+1} (*minimisation*)
- Sinon, accepter s_{i+1} avec la probabilité

$$p(s_i \rightarrow s_{i+1}) = e^{\frac{-(f(s_{i+i}) - f(s_i))}{T}}$$

$\approx$ *Distribution de Gibbs-Bolzmann*

Recuit simulé
Algorithme

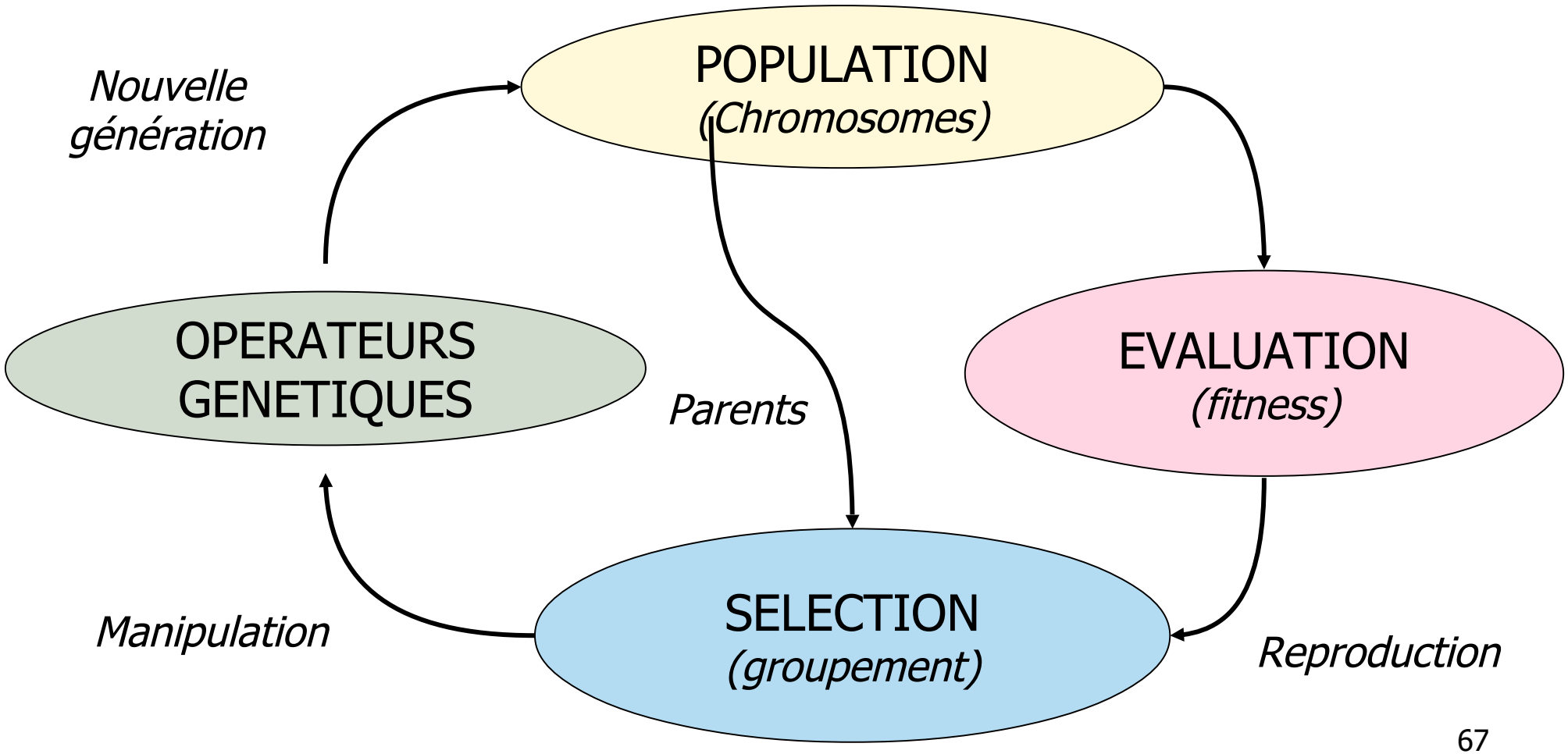
Espace S,
Évaluation f(s)(*min*),
température(T)
accepter(Δf,T)
température initiale
$T = T_0$
Meilleure solution
$s = s_{best}$ = glouton()

Algo SA
tant que critère1 **faire**

tant que critère2 **faire**
s_2 = voisin(s, S)
$\Delta f = f(s_2) - f(s)$
si $f(s_2) < f(s_{best})$
$s_{best} = s_2$
si $\Delta f < 0$ **ou** accepter(Δf, T)
$s = s_2$
fin tant que

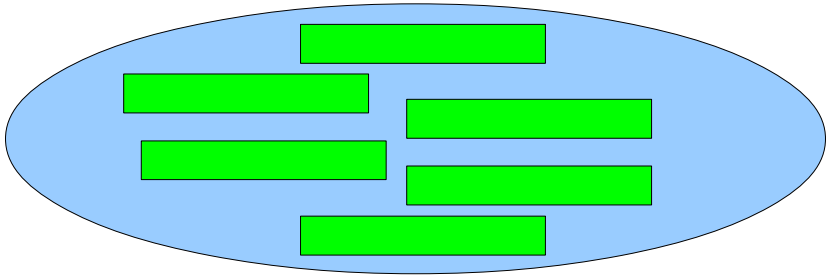
T = température(T)
fin tant que
fin

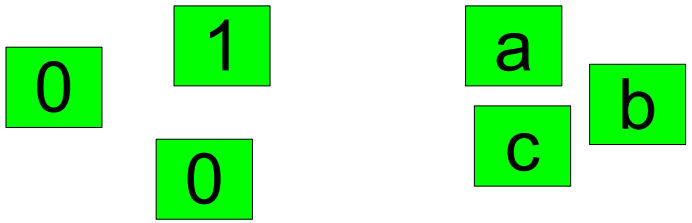
Algorithmes génétiques
principe de base



Algorithmes génétiques
données

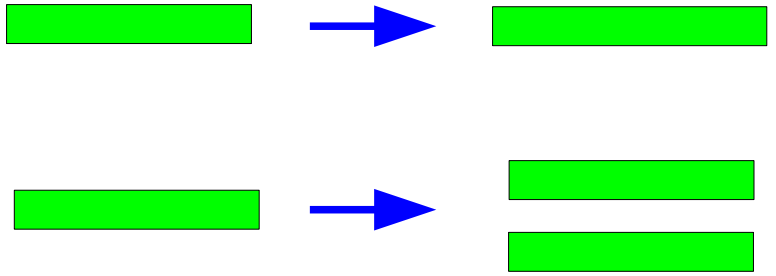
- Population
  - Ensemble d'individus
- Individu
  - Code une solution
  - Représentation par chromosomes
- Chromosomes
  - Découpés en allèles

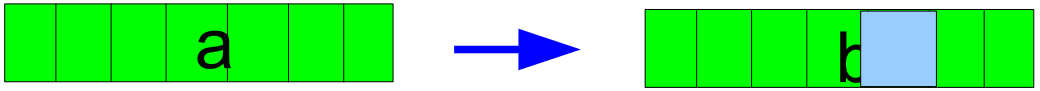


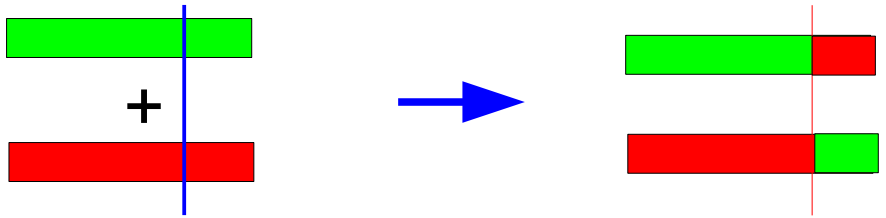


Opérations génétiques
basées sur les chromosomes

- Population
  - Reproduction
  - duplication
- Individu
  - Mutation
  - Crossing over







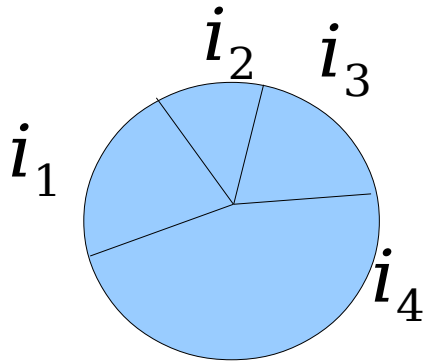
Algorithmes génétiques
Algorithme

Espace S,
Évaluation f(s)(*min*),
crossover(P)
mutation(P)

taux de mutation et
de crossover

Algo GA
générer population initiale P dans S
tant que critère fin non atteint **faire**
reproduction(P) suivant f()
crossover(P)
mutation(P)
fin tant que
meilleure solution de population
fin

Opérations génétiques
choix aléatoires



- Population
  - Pour la reproduction (*roulette wheel*)

    Chances de reproduction dépendant de fitness
- Opérations
  - Mutation suivant un taux

    exemple: 0,5%
  - Crossing over

    % de population

    sites aléatoires

Exploration de l'espace de recherche

Algorithmes génétiques
mise en oeuvre

- Technique empirique
- Grande variété pour
  - Codage (*building blocks*)
  - Probabilités
  - Opérations
  - Couplage avec d'autres techniques
- Possiblités de parallélisme

Algorithmes génétiques
Un exemple

- P-median (p points à choisir parmi N candidats)
- Codage chromosome C[]
  - $\forall i\ 1 \leq i \leq p,\ 1 \leq C[i] \leq N$
  - $\forall i,j\ 1 \leq i,j \leq p,\ i \neq j,\ C[i] \neq C[j]$
- Opérations
  - Mutation : $C[i] \leftarrow n$
    choix aléatoire de i t.q $1 \leq i \leq p$,
    choix aléatoire de n t.q $\forall j$ in $1 \leq j \leq p$, $C[j] \neq n$.
  - Crossover C[] et D[]: choix d'un site $1 < c < p$
    - Forall $1 \leq i \leq c$, $E[i] \leftarrow C[i]$
    - Forall $c < i \leq p$, $E[i] \leftarrow D[i]$
    - Muter les doublons éventuels dans E[]

Algorithmes génétiques
parallélisme

- Possibilités de parallélisme
  - Modèle à grain fin maitre/esclave (// sur l'évaluation des individus)
    - Application d'heuristiques d'amélioration locale possible
  - Modèle à gros grain (// sur plusieurs populations distinctes et indépendantes)
    - Modèle des ilots (transfert éventuel d'individus dans des populations distantes)

Algorithmes génétiques
parallélisme – un exemple

- Sur des problèmes de test de TSPLIB
  http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html
  - Instances en 2D (distances euclidiennes)
  - Modèle à ilots, 10 stations en ethernet 100Mb/s, MPI (// interprocs)
    - 1000 générations
    - 1000 individus
    - Probabilité de mutation 30%, (cross over ?)
    - Migration circulaire, 100 itérations, 20% de la pop.
    - Crossover à 1 point, sélection proportionnelle

Algorithmes génétiques
parallélisme – un exemple

Comparison of Parallel Metaheuristics for Solving the TSP
M. Lazarova, P. Borovska
CompSysTech'08

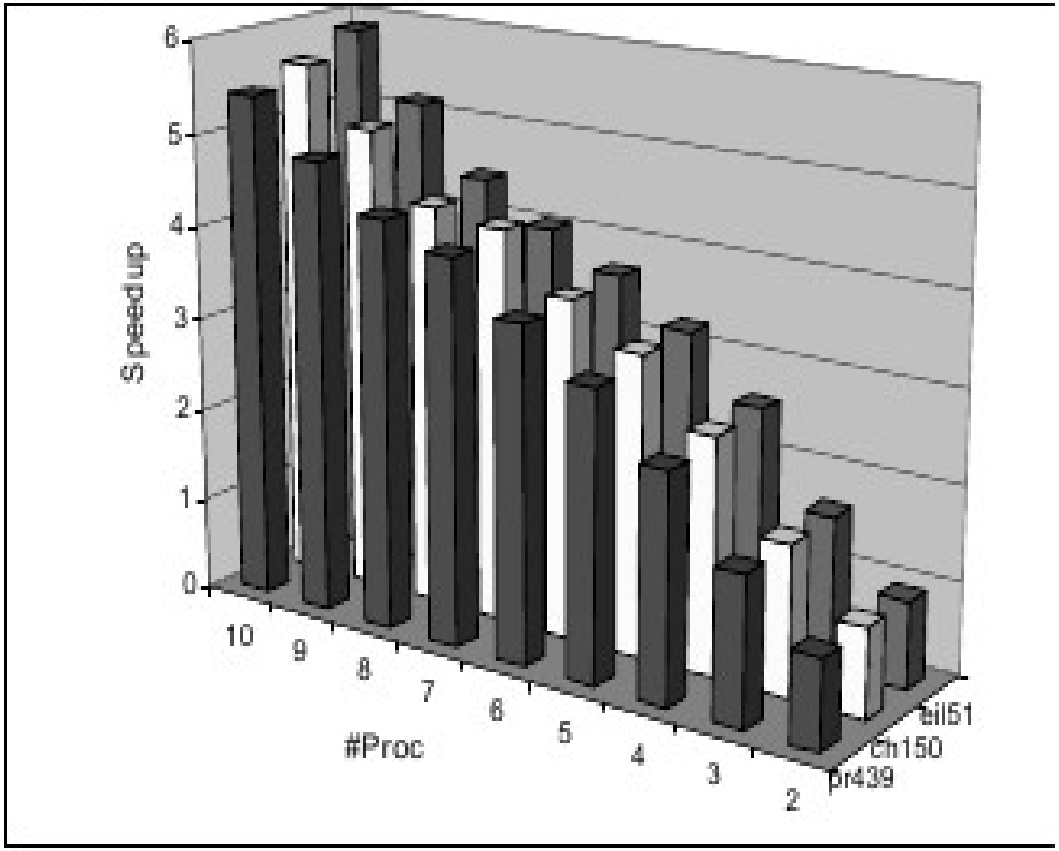


Fig.6. Speedup of parallel GA

Algorithmes évolutionaires
en résumé