Algorithmique avancée et parallélisme

Laurent Lemarchand
LISyC/UBO
Laurent.Lemarchand@univ-brest.fr

Partie 2
Techniques d'optimisation combinatoire

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 ...

Optimisation combinatoire
introduction

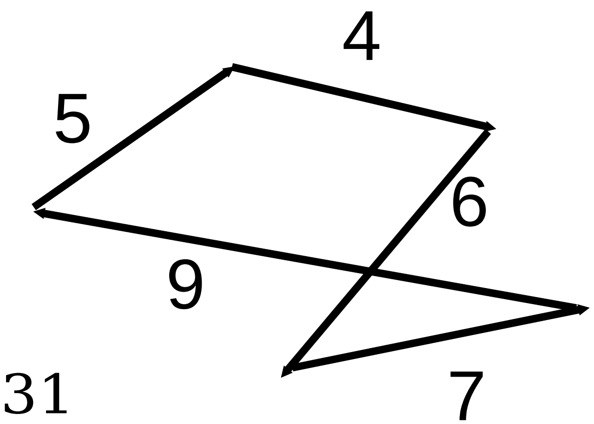
- Problème d'optimisation : *trouver la meilleure solution à un problème donné parmi un ensemble de solutions possibles (réalisables)*

  - *S :* espace de solutions

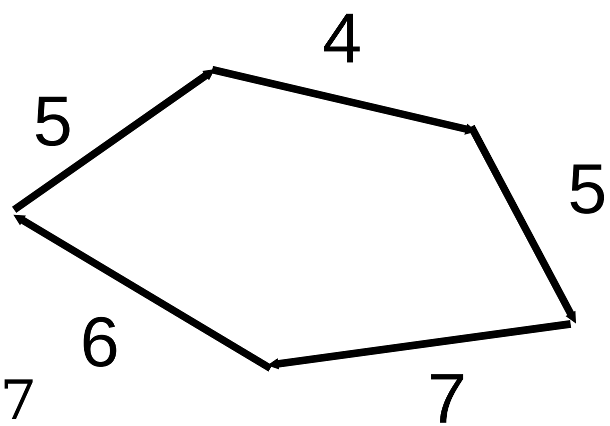
  - *f(S) :* fonction de coût permettant de juger la qualité d'une solution

Optimisation combinatoire
exemples

- Voyageur de commerce : *passer dans toutes les villes une et une seule fois et revenir à un point de départ*
  - *S* *:* toutes les tournées possibles
  - *f(S)* *:* longueur d'une tournée (à minimiser !)



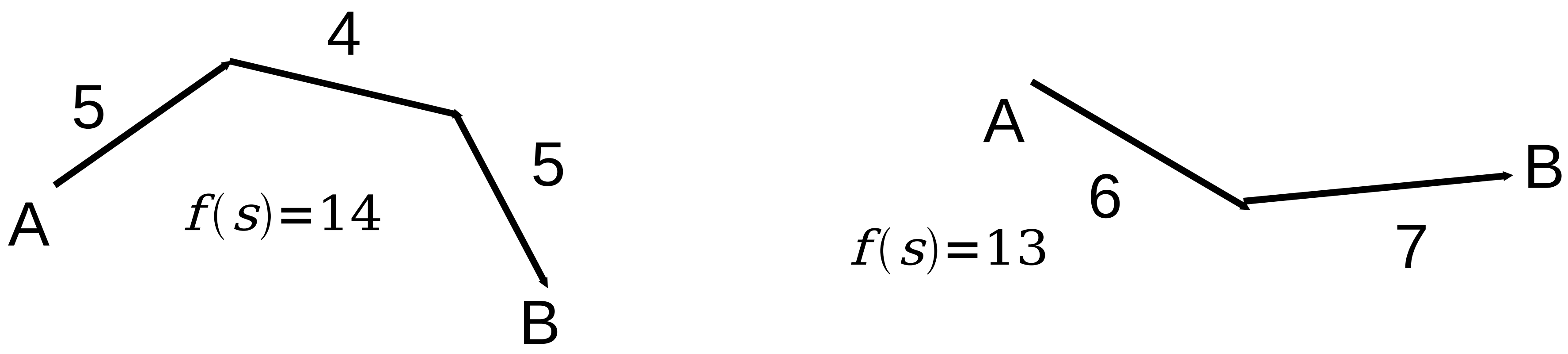
$f(s)=31$



$f(s)=27$

Optimisation combinatoire
exemples

- Calcul de chemins : *plus court chemin dans un graphe entre 2 sommets*
  - *S :* toutes les chemins possibles
  - *f(S) :* longueur d'un chemin (à minimiser !)



- Optimisation dans les graphes ...

Optimisation combinatoire
exemples

- Un meunuisier peut fabriquer au plus 6 chaises et 3 tables par journée de 8 heures maximum
  - Une table lui rapporte 90 euros (1h15 de travail)
  - Une chaise, 50 euros (45mn de travail)
- Maximiser son profit

$$f(s) = 90t + 50c$$
$$75t + 45c \leqslant 480$$
$$0 \leqslant t \leqslant 3$$
$$0 \leqslant c \leqslant 6$$

- *Programmation linéaire :* simplexe en $O(2^n)$

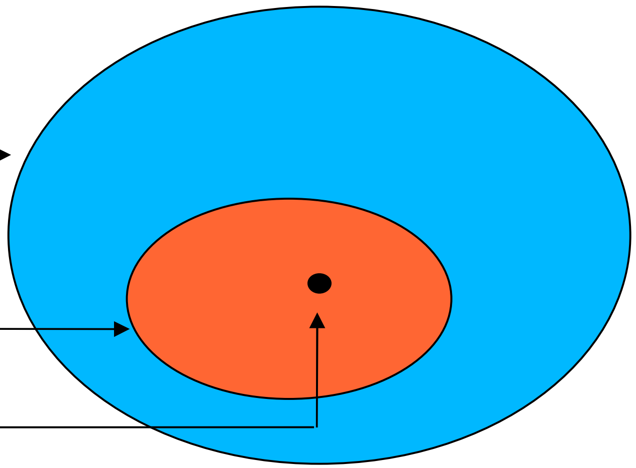
Optimisation combinatoire
cadre général

- Espace de solutions $S \subseteq X$

- Fonction objective $f : X \rightarrow \mathbb{R}$

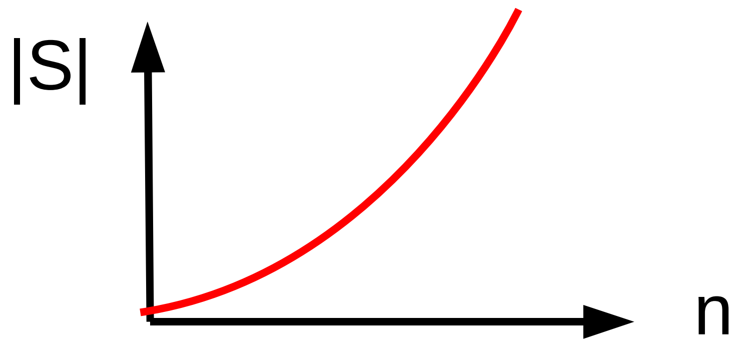
- Trouver $s^* \in S$ t.q $\forall s \in S \; f(s^*) \leq f(s)$

- X , espace des solutions
- S , espace des solutions **réalisables**
- s* , solution optimale



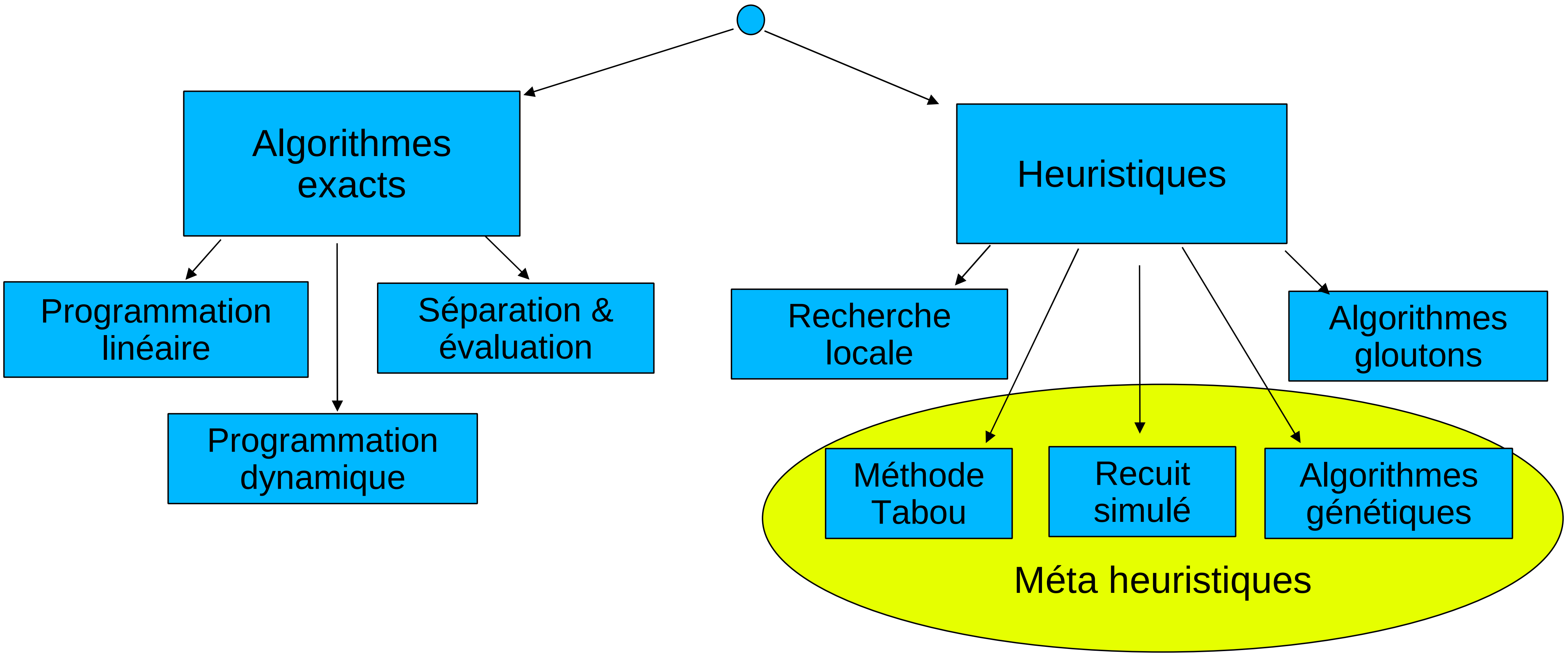
Optimisation combinatoire
explosion combinatoire

- Taille de S par rapport à la taille du problème
  - Voyageur de commerce (n-1)! / 2
  - Bi partitionnement 2^n
  - Programme en variable bivalentes 2^n



- La taille de S est exponentielle

Optimisation combinatoire
algorithmes de recherche



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 :

  $V^* = f(v_1, v_2, \ldots, v_n)$

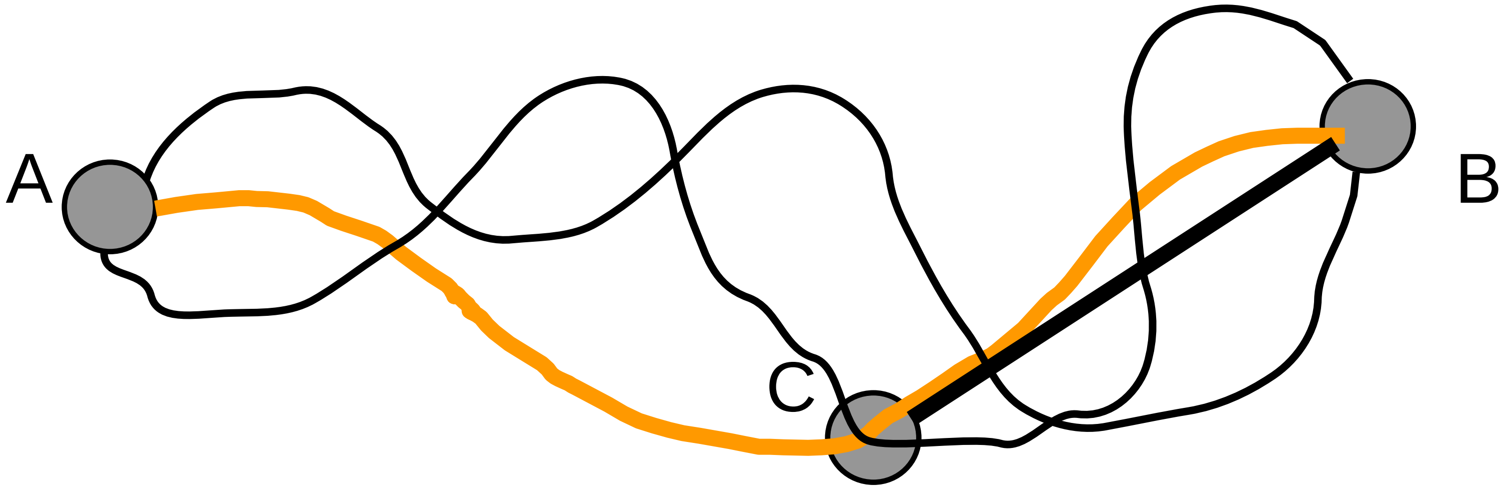
- Mécanisme récursif

Programmation dynamique
conditions d'utilisation

- Principe d'optimalité

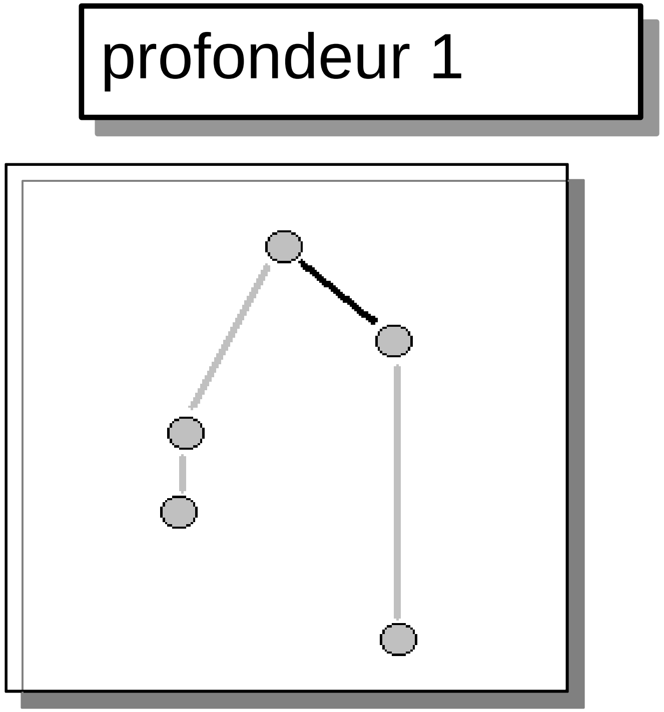
Une solution est optimale ssi.
ses sous-solutions sont optimales

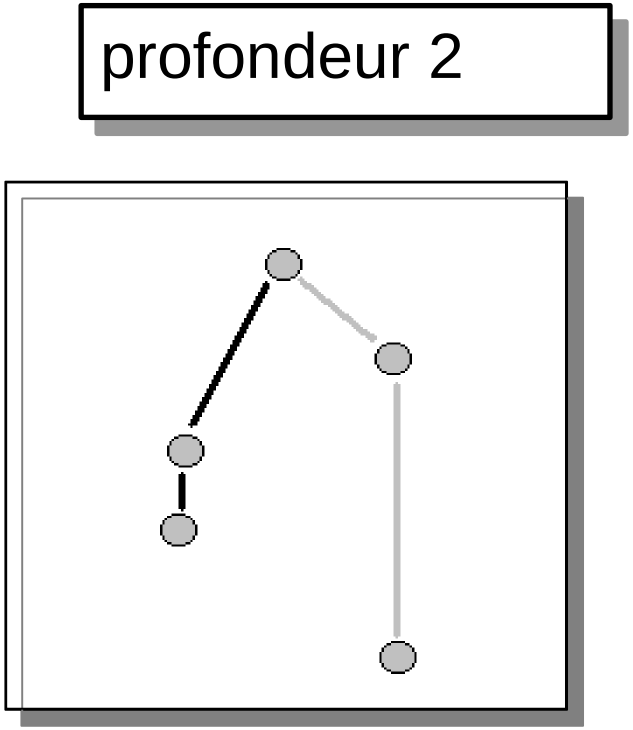
- Exemple : plus court chemin : (A,B) est optimal ssi (A,C) et (B,C) sont optimaux.



Principe d'optimalité
contre exemple

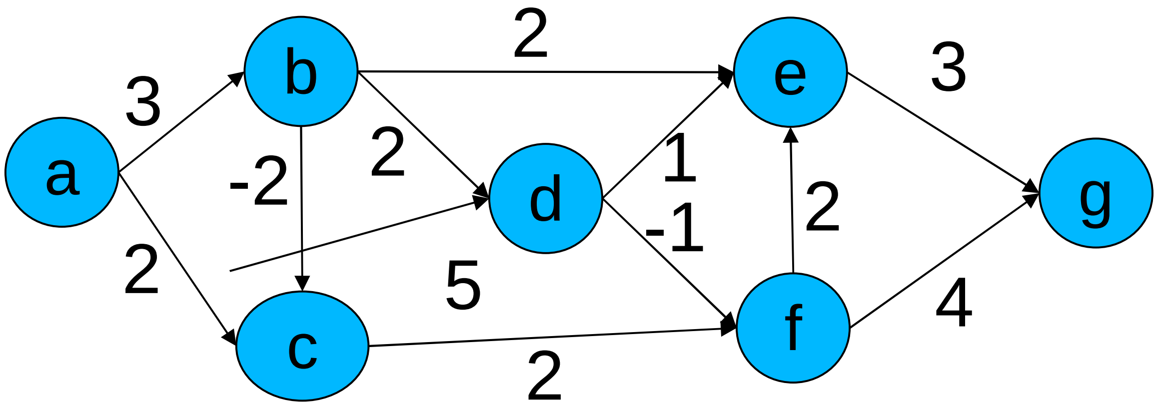
- Recherche de plus court chemin de profondeur donnée dans un arbre





•Il faut examiner toutes les solutions partielles

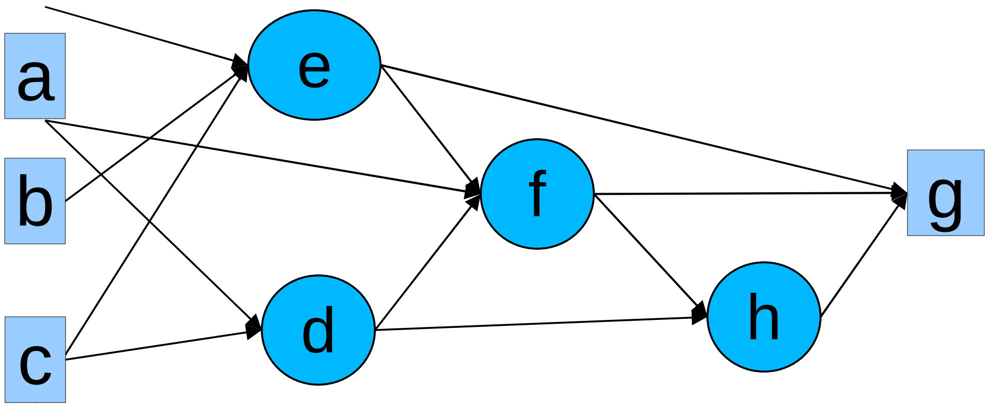
Programmation dynamique
plus court chemin : Ford-Bellman

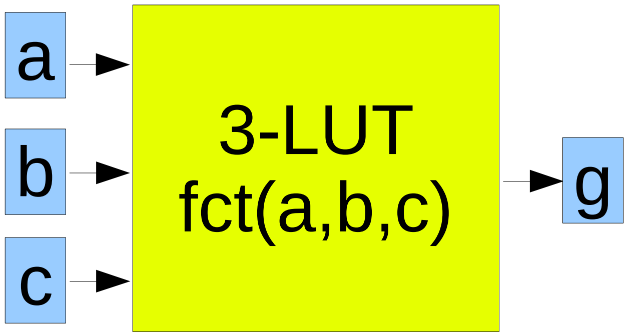


- Calcul de chemins : plus court chemin de **a** à l'un de ses successeurs
- Récursivement : $\Pi(s) = \min_{s' \in preds(s)} \Pi(s') + w_{s' \rightarrow s}$
- Autres exemples : *Chortle-crf,* allocation de ressources, gestion de stock ...

Programmation dynamique
Exemple : Chortle-crf

- Implantation de circuits logiques sur FPGA

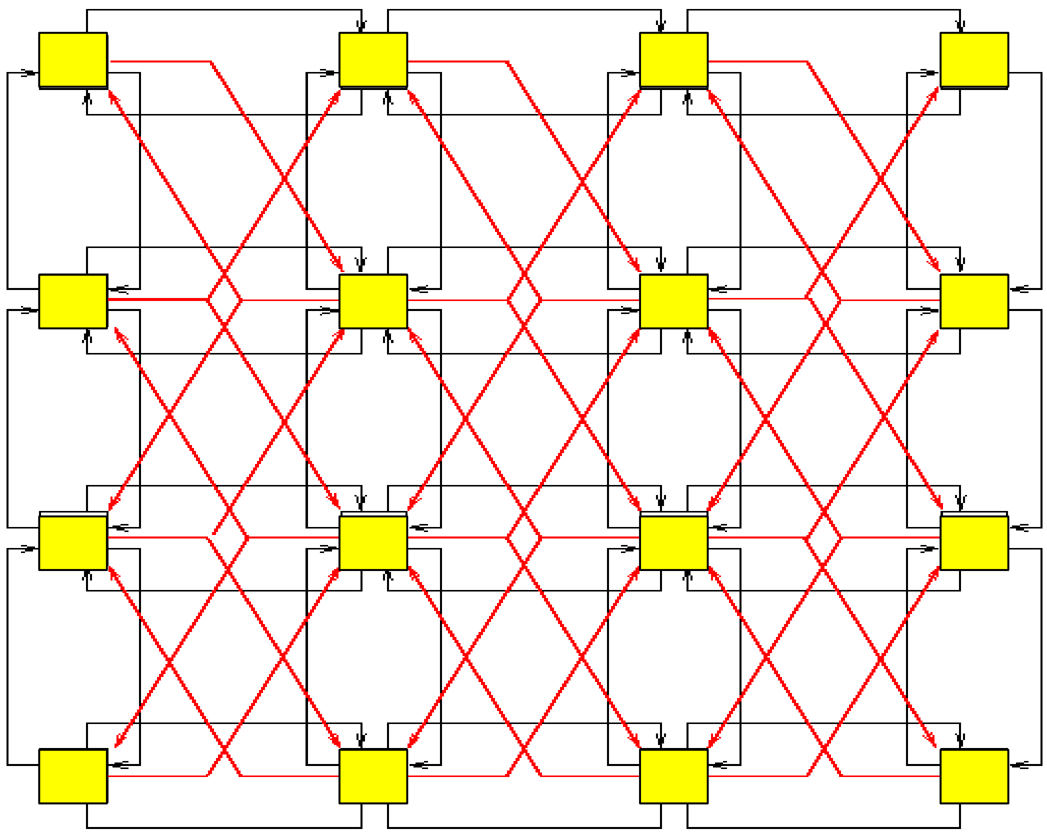




- Grouper les noeuds logiques en k-LUTs : une LUT peut implanter n'importe quelle fonction logique d'au plus k entrées (table de vérité)

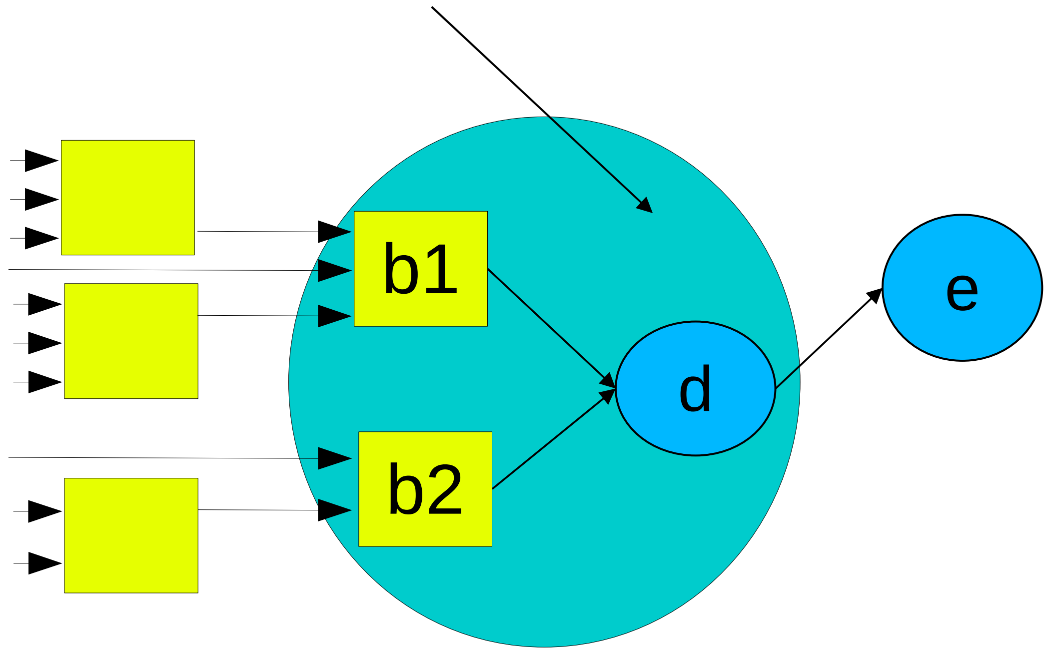
Chortle-crf
FPGA (Field Programmable Gate Array)

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



Programmation dynamique
Exemple : Chortle-crf

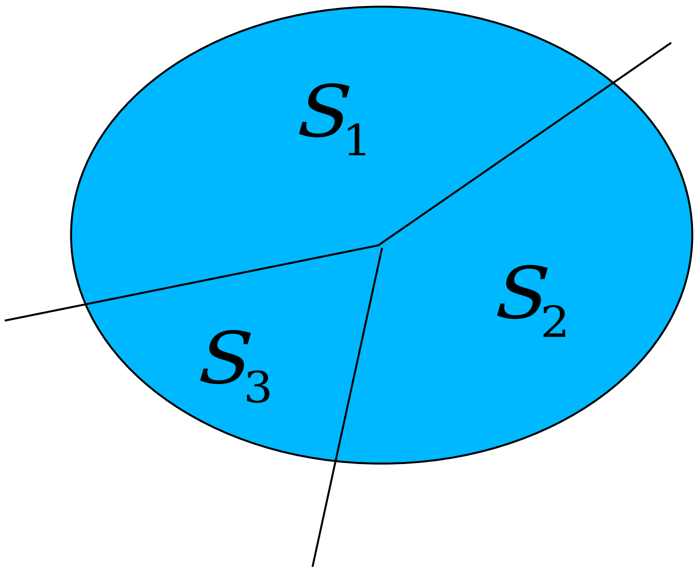
- Progression des entrées vers les sorties

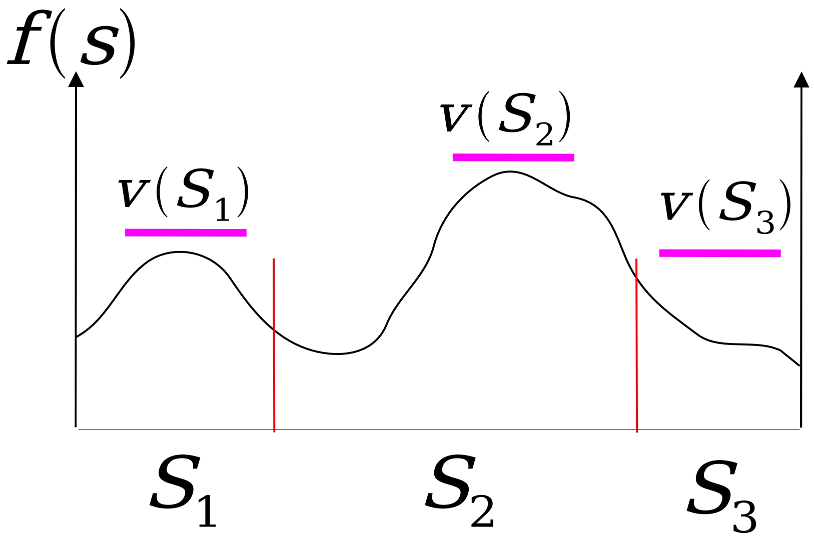


- La solution pour d+b1+b2 doit
  - Minimiser le nombre de LUT
  - Minimiser le fanin de la LUT de tête

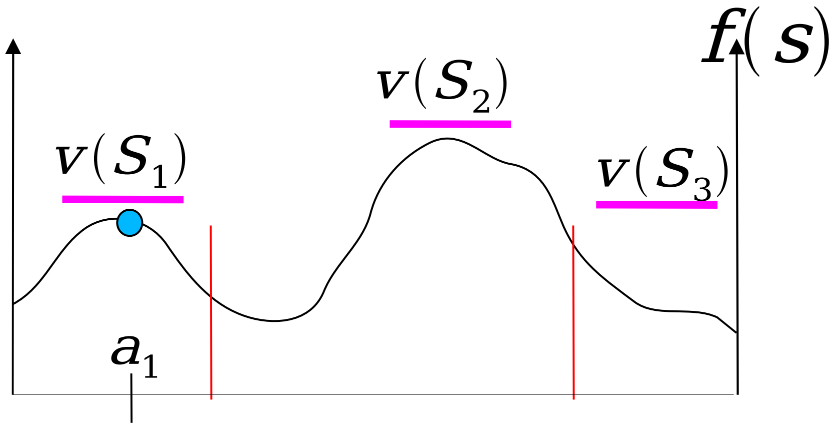
Méthodes par séparation et évaluation
principe B&B

- 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*)

branch

  - Une fonction de partitionnement d'un espace S en sous espaces

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

bound

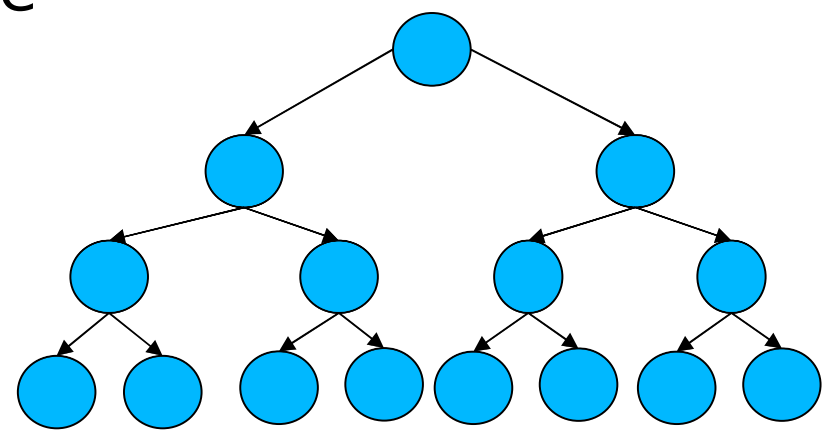
  - Une fonction d'évaluation de qualité pour un espace donnant une borne supérieure sur S :

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

  - 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 $L \neq \emptyset$ **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

      $S_{best} = S_i$

      $U = v(S_i)$ $(= f(S_i))$

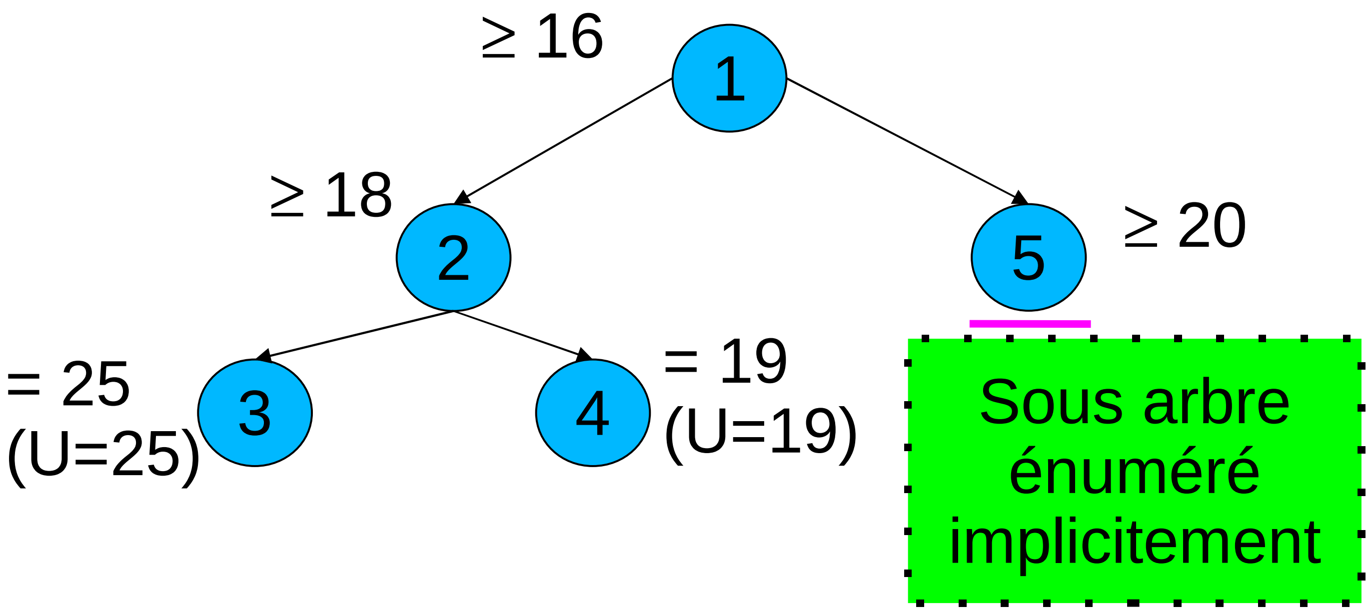
    sinon

      $L = L \cup \{S_i\}$

fin

Méthodes par séparation et évaluation
coupes

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



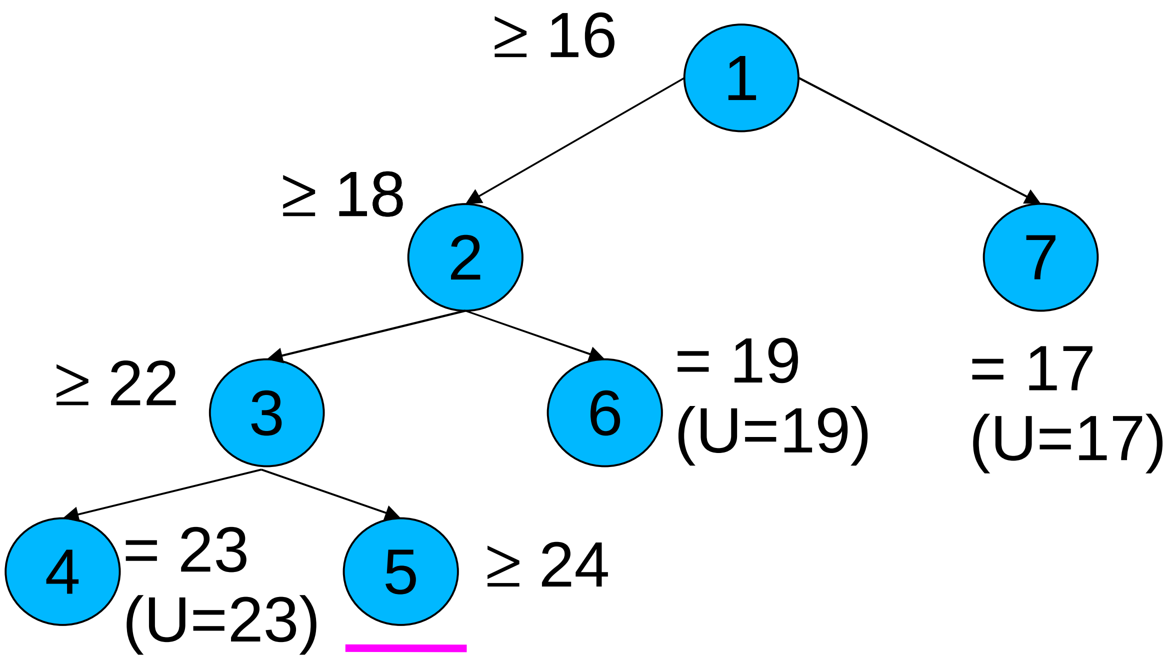
Méthodes par séparation et évaluation
parallélisation

- 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

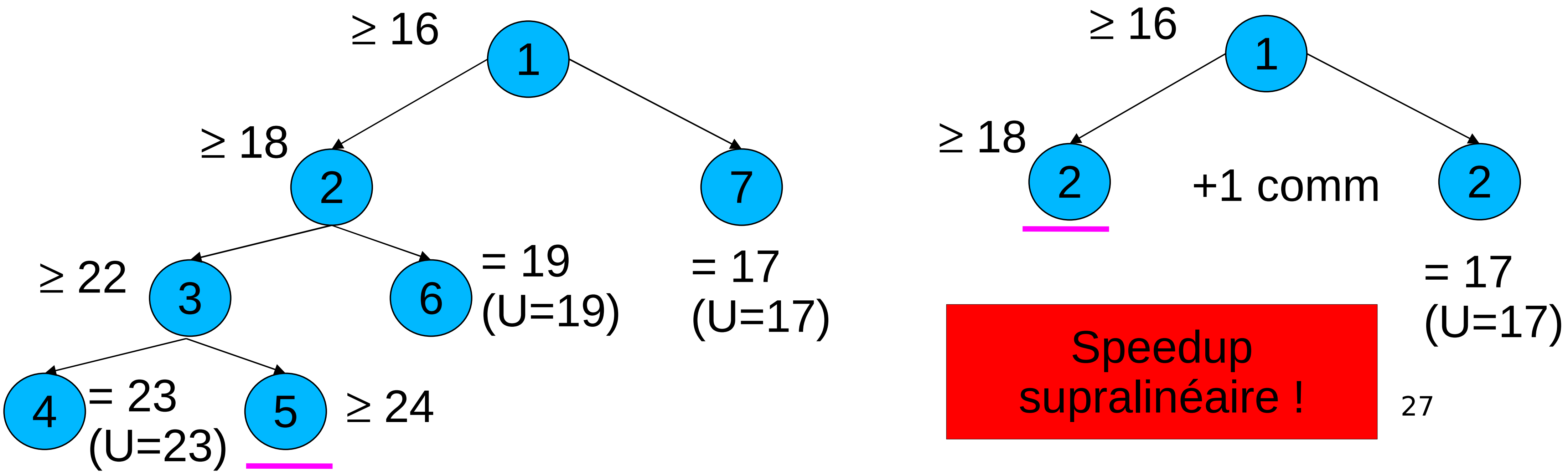
Méthodes par séparation et évaluation accélération et parallélisation

- 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

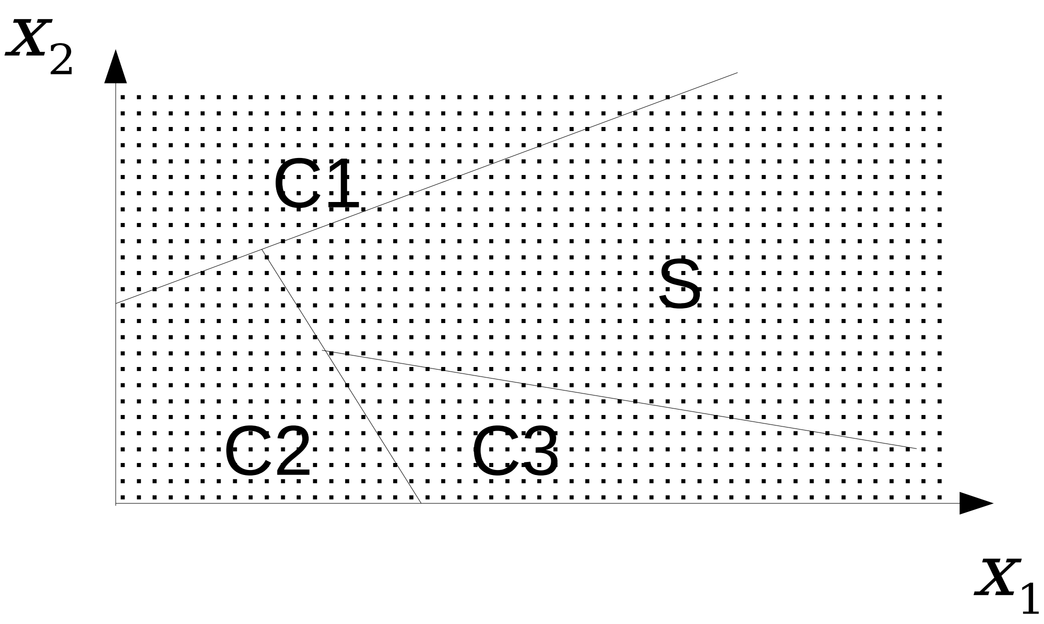
- Exemple avec mise en oeuvre centralisée
  - 1 maitre, 2 esclaves, comms 1 unité de temps en // ...
  - Parcours profondeur d'abord de l'arbre de recherche
    7 unités de temps en séquentiel
    2 unités de temps en // sur 2 processeurs



Speedup supralinéaire !

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

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

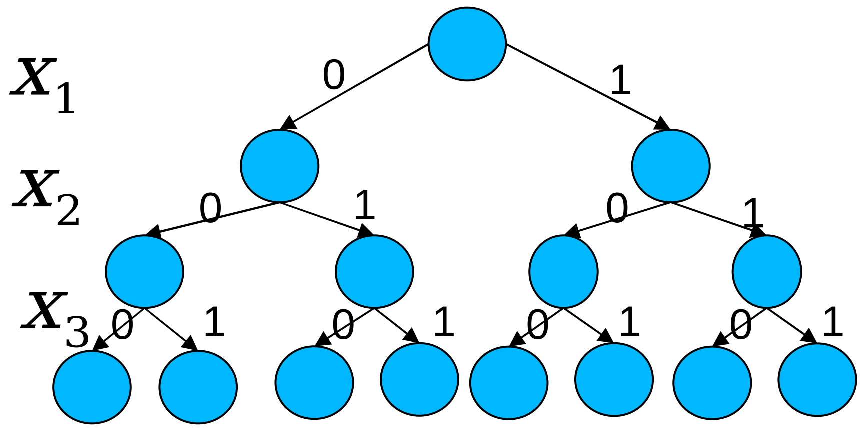


$$x_1 + 3x_2 \leqslant 15$$
$$-2x_1 - x_2 \leqslant -12$$
$$-3x_1 - 11x_2 \leqslant -66$$
$$min \quad 2x_1 + 4x_2$$
$$x_1, x_2 \in \mathbb{R}$$

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

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*)



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	∞

Exemples d'utilisation B&B
évaluation

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

	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	∞

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

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

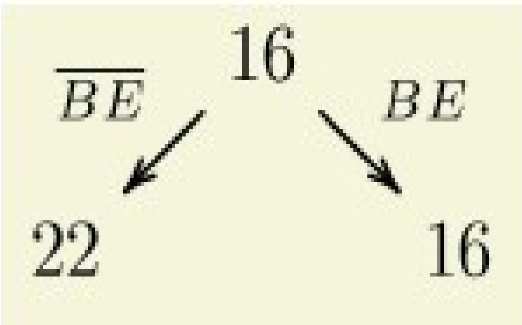
    $d = 16 + 3$

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 … choix du plus coûteux en espérant le couper ensuite

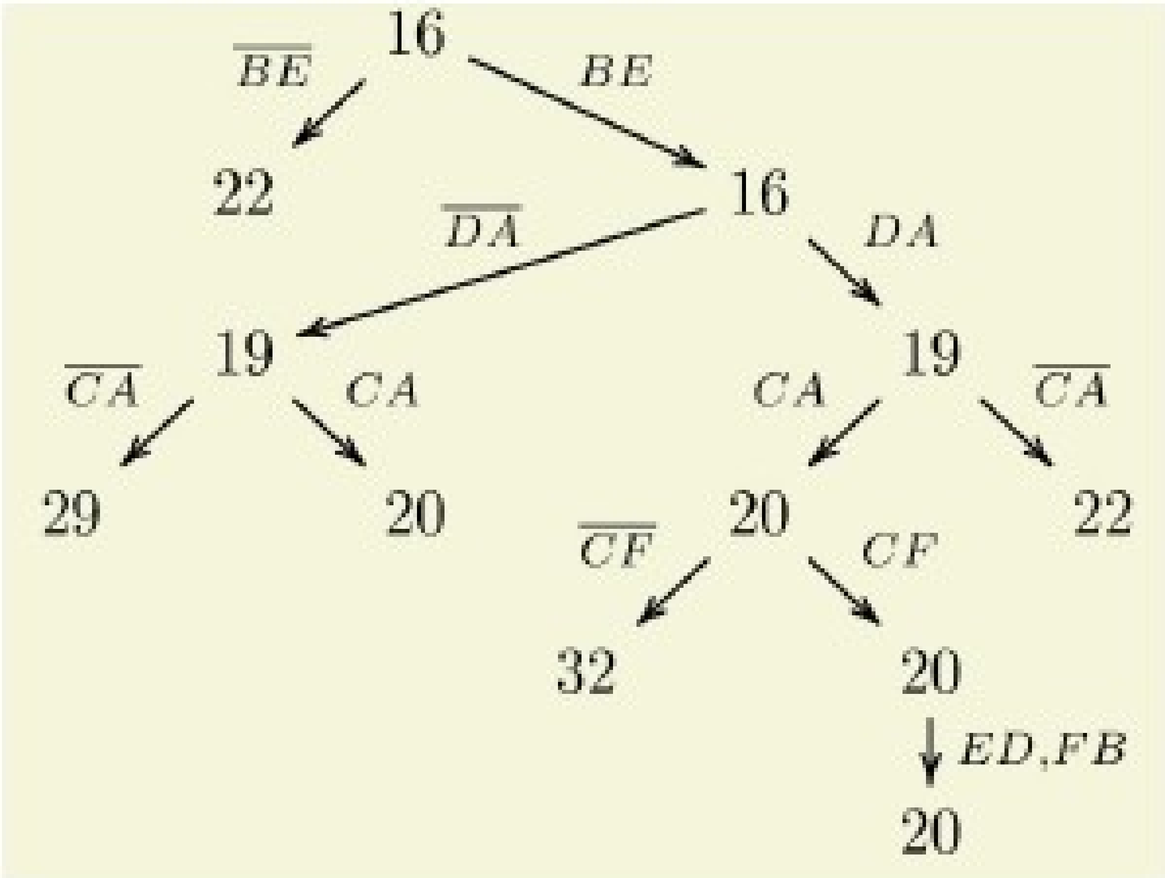


	A	B	C	D	E	F
A	∞	0(2)	3	2	13	1
B	2	∞	2	8	0(6)	23
C	3	11	∞	0(0)	4	0(1)
D	0(2)	1	0(2)	∞	7	9
E	13	5	6	0(2)	∞	2
F	16	1	6	0(1)	14	∞

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 $G' = (V', E)$ $V' = V \setminus \{V_0\}$
  - 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

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
    $a_1x_1 + a_2x_2 \; ... \; + a_nx_n \;\leq\; b$
    $\textit{max} \;\; c_1x_1 + c_2x_2 \; ... \; + c_nx_n$
    $x_j \leq \beta_j \qquad X_j \in \mathbb{N}$

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

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 $c_1x_1 + c_2x_2 + \ldots + c_mx_m$

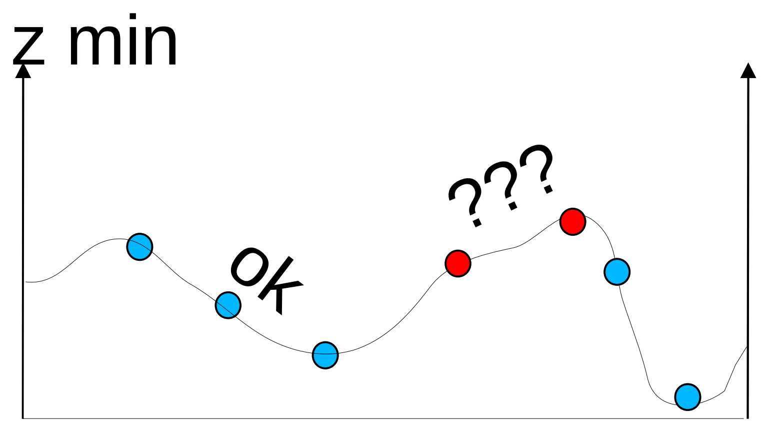
  $x_j \leq 1$ et $x_j \in \mathbb{N}$
- Trouver k t.q $c_k/a_j = \mathit{min}_{\, j \in 1..n}\ c_j/a_j$ avec $a_j = \sum_{i=1..m} A_i^j$
- Eliminer la colonne k et toutes les lignes tq $A_i^k = 1$
- Recommencer sur le problème réduit

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

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 $V : S \longrightarrow S^n$
  - 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 $s \in S$

générer une solution initiale s_0

$s = s_0$

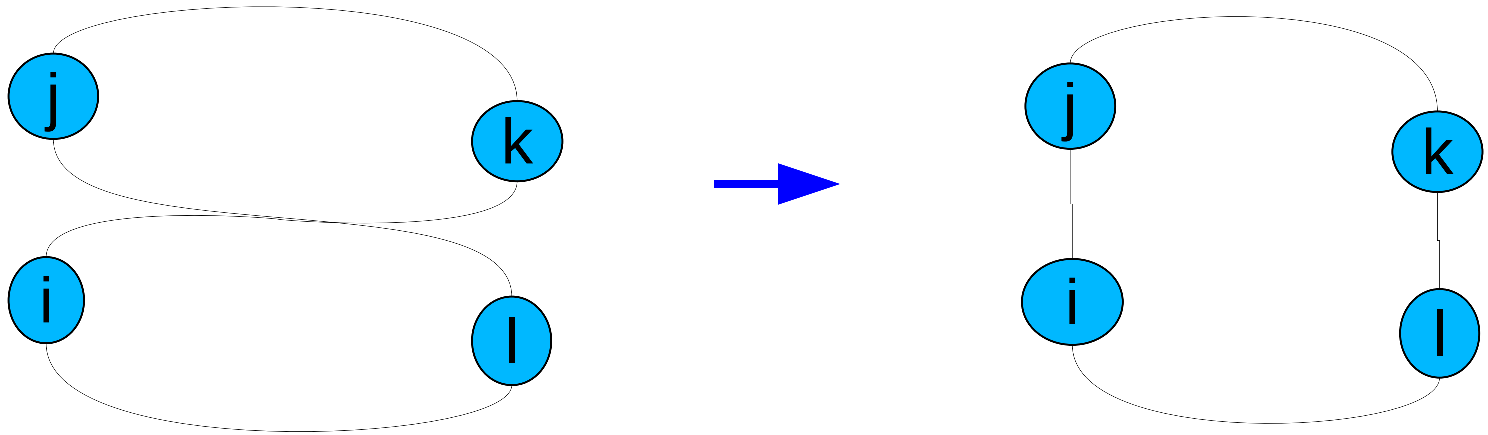
Tant que non fin

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

Exploration locale
problème du voyageur de commerce

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

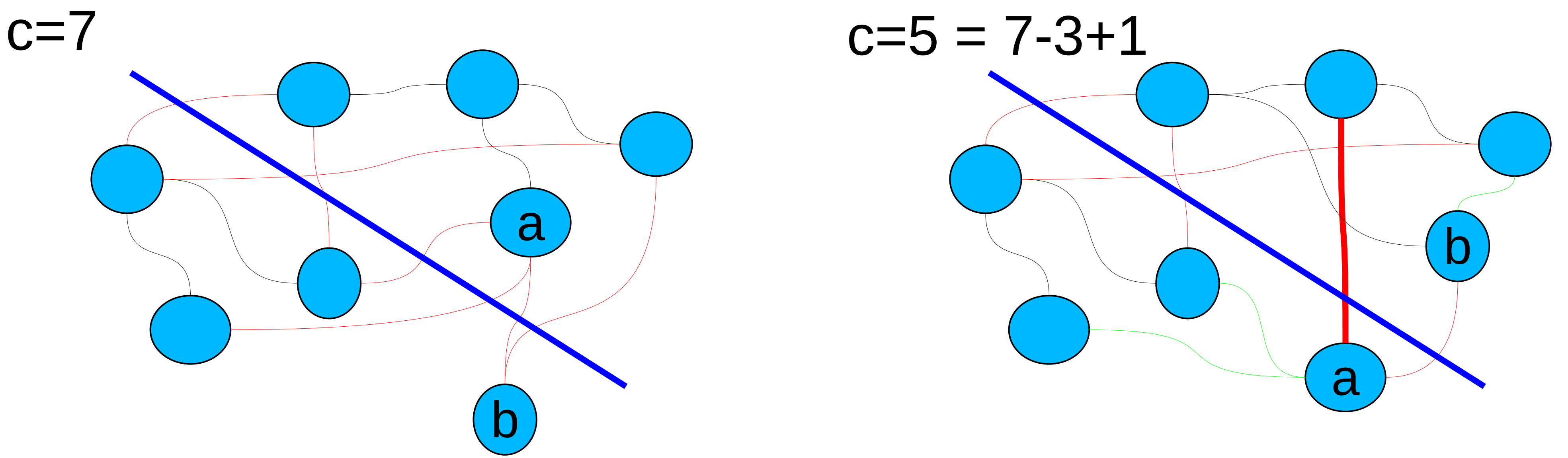
    T' = T U { i→k, j→l } \ { i→j, k→l }



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

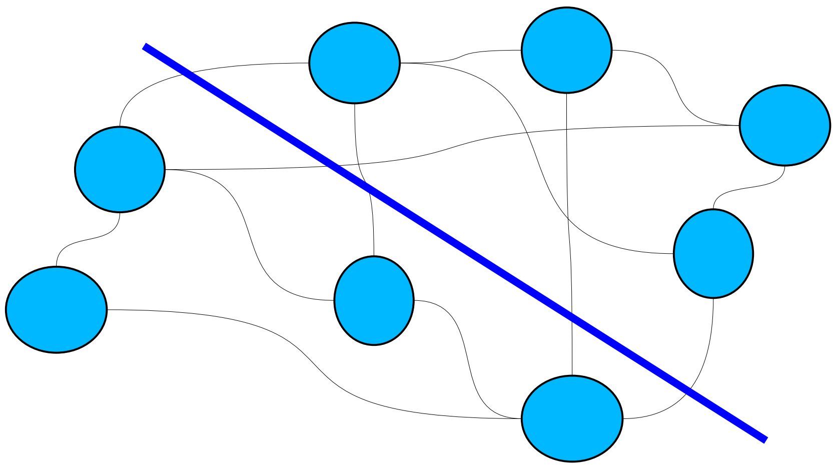
Exploration locale
problème du 2-partitionnement

- 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



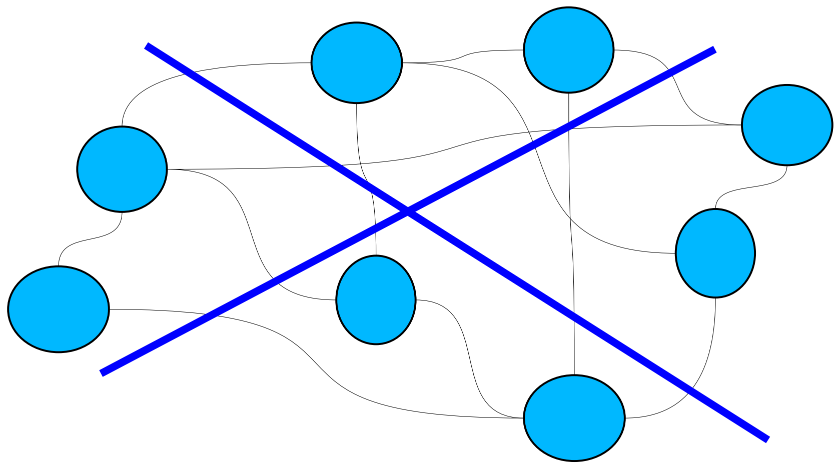
2-partitionnement
heuristique de Kernighan-Lin

- 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
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

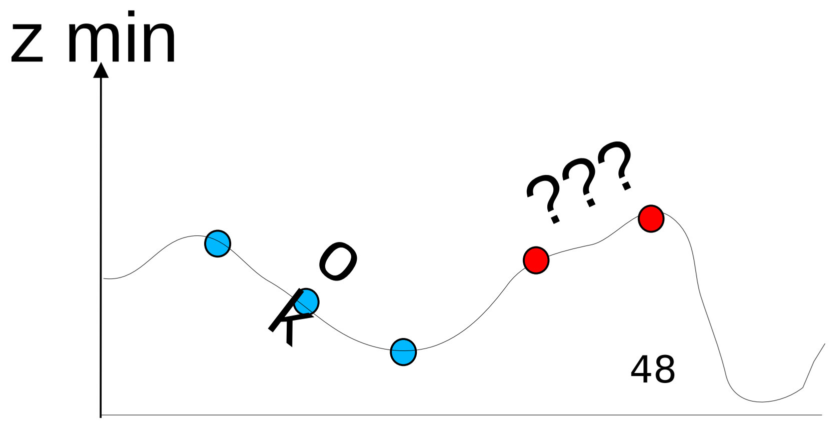
Amélioration des méthodes de descente
problématique

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

Exploitation ⟷ Exploration

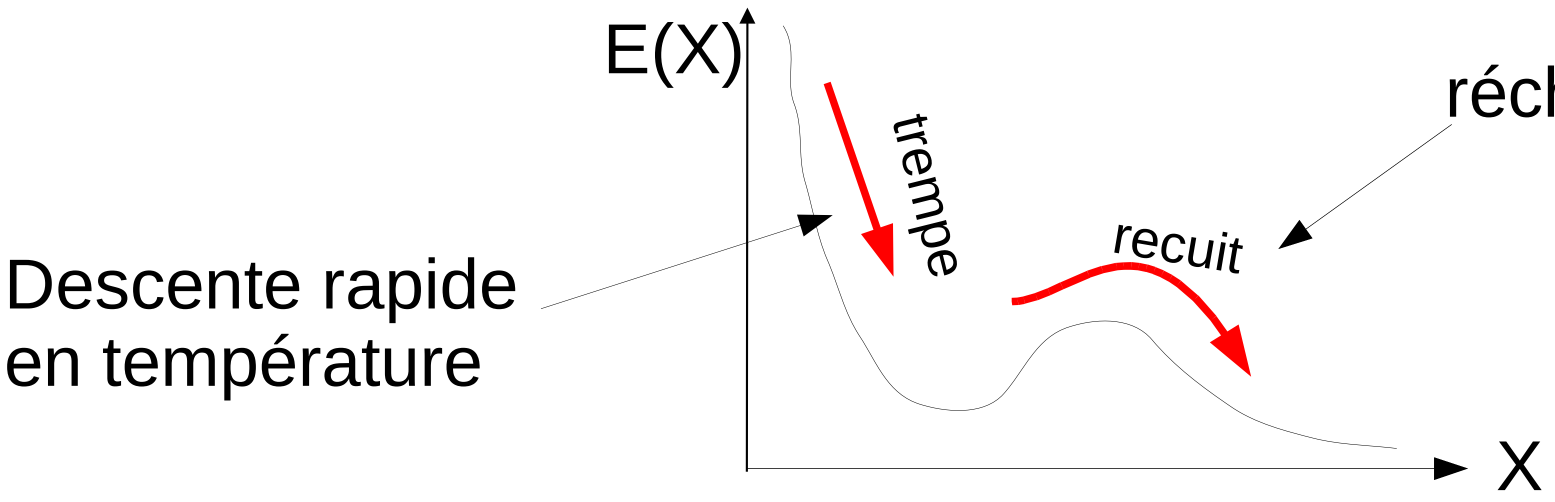
- Toujours le problème des extrema locaux

- Politique de compromis



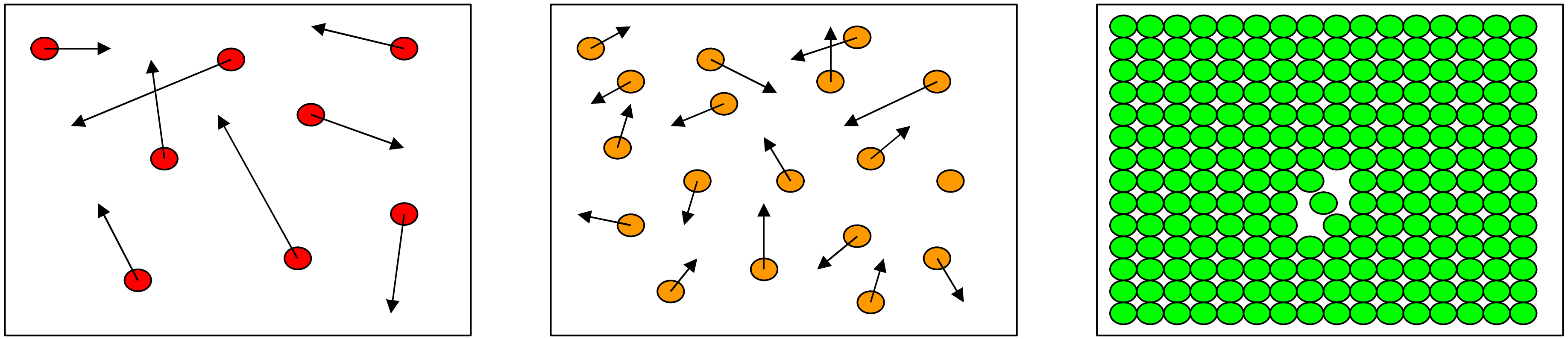
Recuit simulé
simulated annealing (SA)

- 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

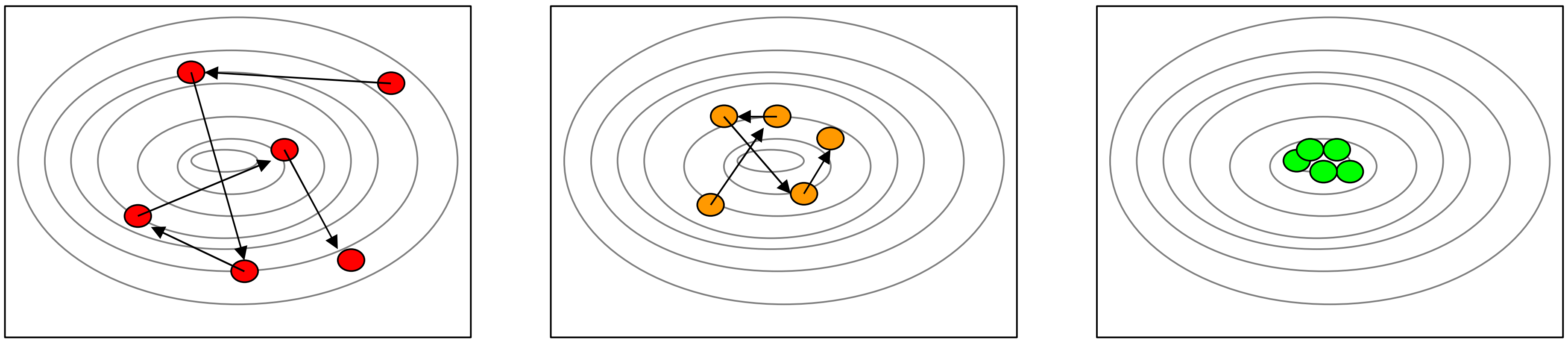


Recuit simulé
analogie physique/optimisation

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

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

≈ *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

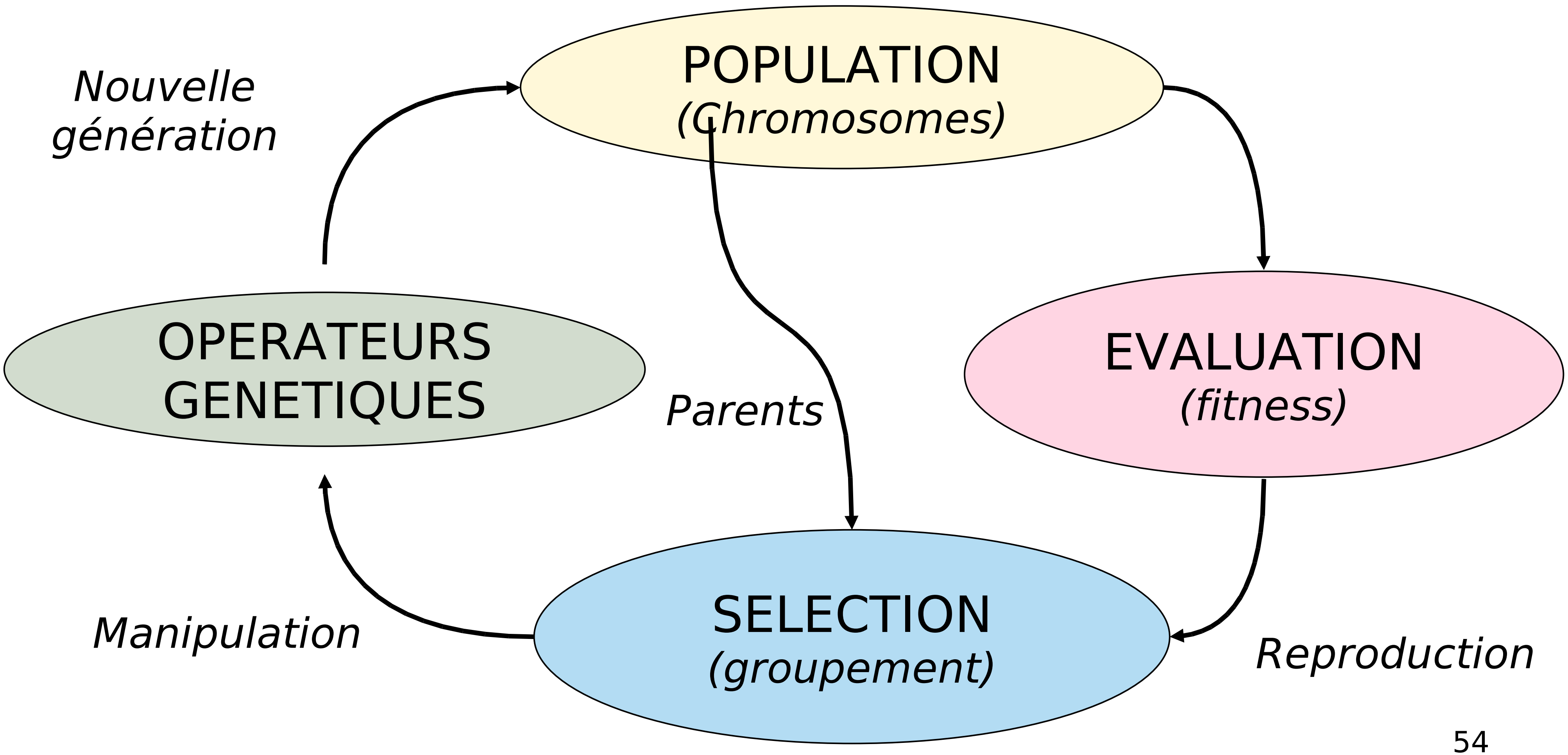
$s = s_{best}$ = glouton()

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

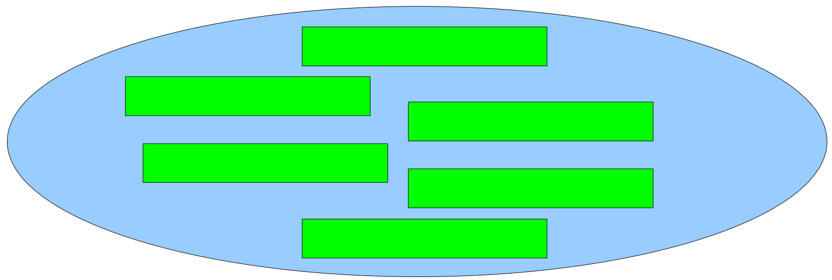
T = température(T)
fin tant que
fin

Algorithmes génétiques
principe de base

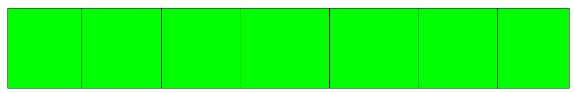


Algorithmes génétiques
données

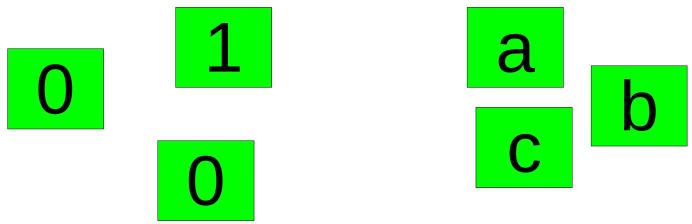
- Population
  - Ensemble d'individus



- Individu
  - Code une solution
  - Représentation par chromosomes

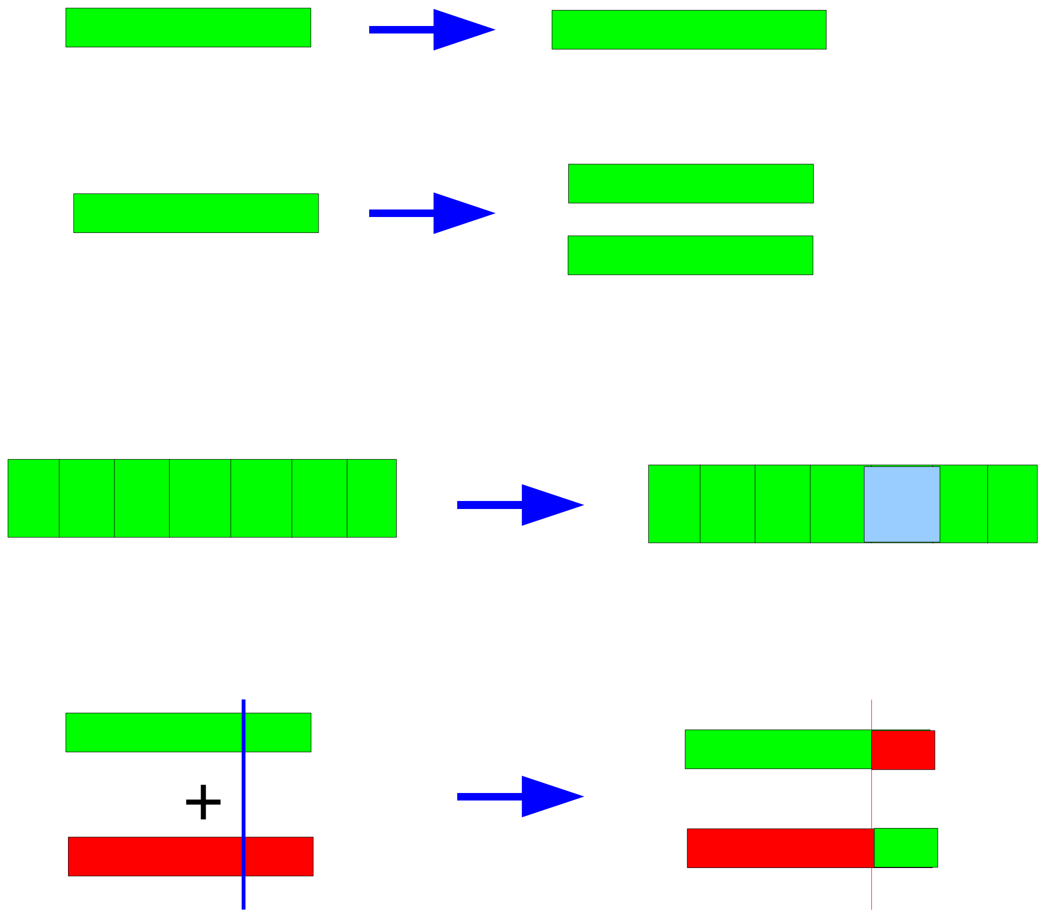


- Chromosomes
  - Découpés en allèles



Opérations génétiques
basées sur les chromosomes

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



Algorithmes génétiques
Algorithme

Espace S,

Évaluation f(s)(*min*),

crossover(P)

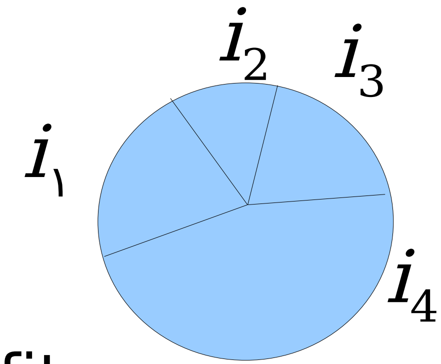
mutation(P)

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
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

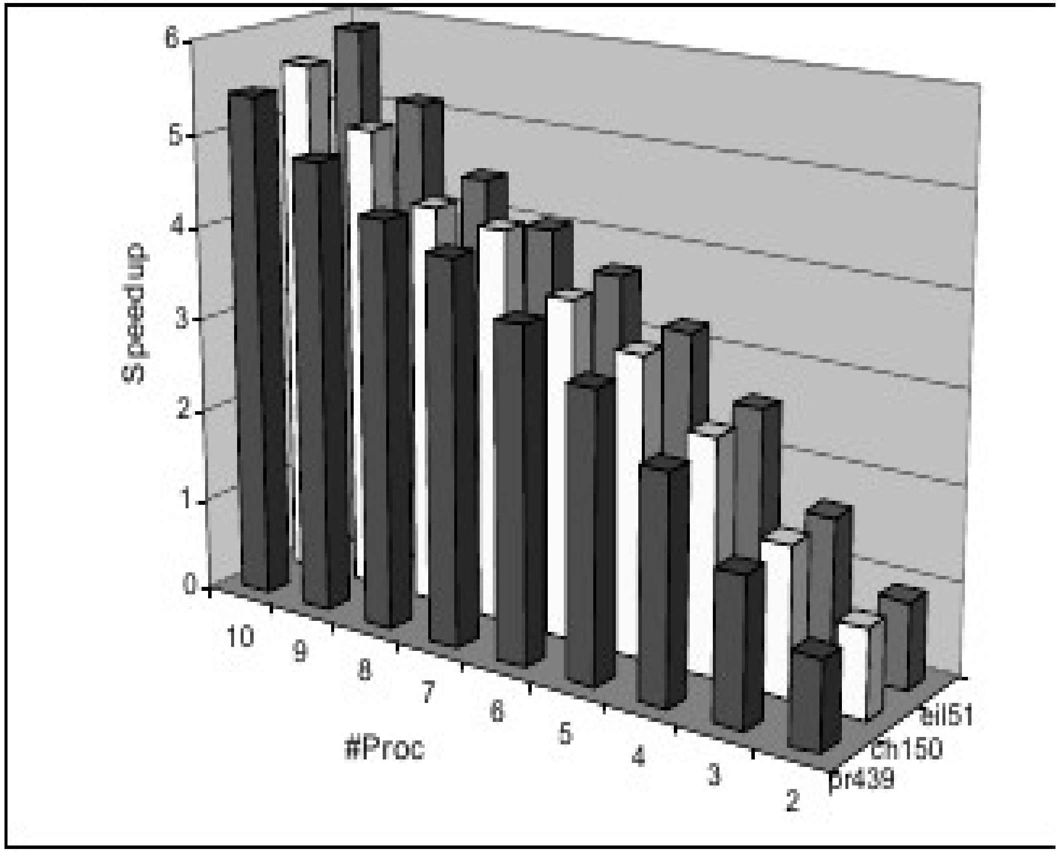


Fig.6. Speedup of parallel GA