# Introduction à l'optimisation combinatoire

Philippe Morignot

# Sommaire

- Introduction
- Techniques
- Outils du marché
- Références

# Introduction

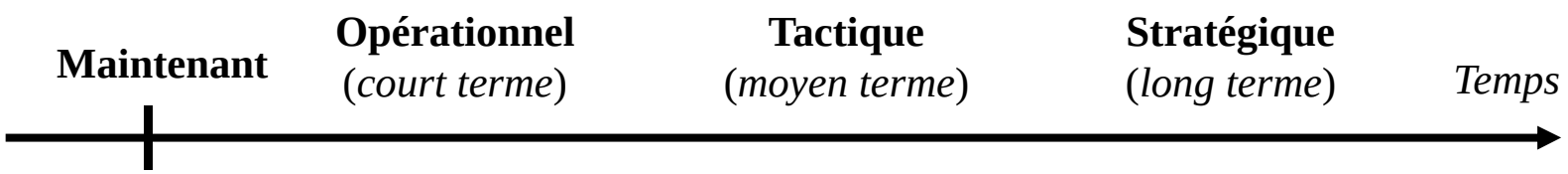
- Intuitivement :
  - “*Optimiser*” = améliorer un processus existant.
  - “*Combinatoire*” = dérivé de “combinaison” : il y a plusieurs façons d’améliorer cet existant.
- Optimisation combinatoire = recombiner un processus existant de façon à l’améliorer.
  - “*amélioration*” mesurée par la définition d’un coût ou de la qualité d’une combinaison de l’existant.
- Formellement : chercher le minimum d’une hyper-surface par déplacements continus ou discrets.

# Introduction - solution naïve

```
while(...) {
   combinaison = nextCombinaison(existant);
   if (qualite(combinaison) > bestQualite) {
      bestQualite = qualite(combinaison);
      bestCombinaison = combinaison;
   }
}
```

**BEAUCOUP TROP LONG !!!!**
**(même sur des petits problèmes)**

# Introduction - applications




* Attribuer des salles et des ressources à des élèves.
* Attribuer des places de parking à des avions.
* Se réorganiser suite à un aléa de dernière minute.

* Dimensionner et situer des entrepôts en fonction de la clientèle.
* Dimensionner une flotte d'avions de ligne.

# Techniques

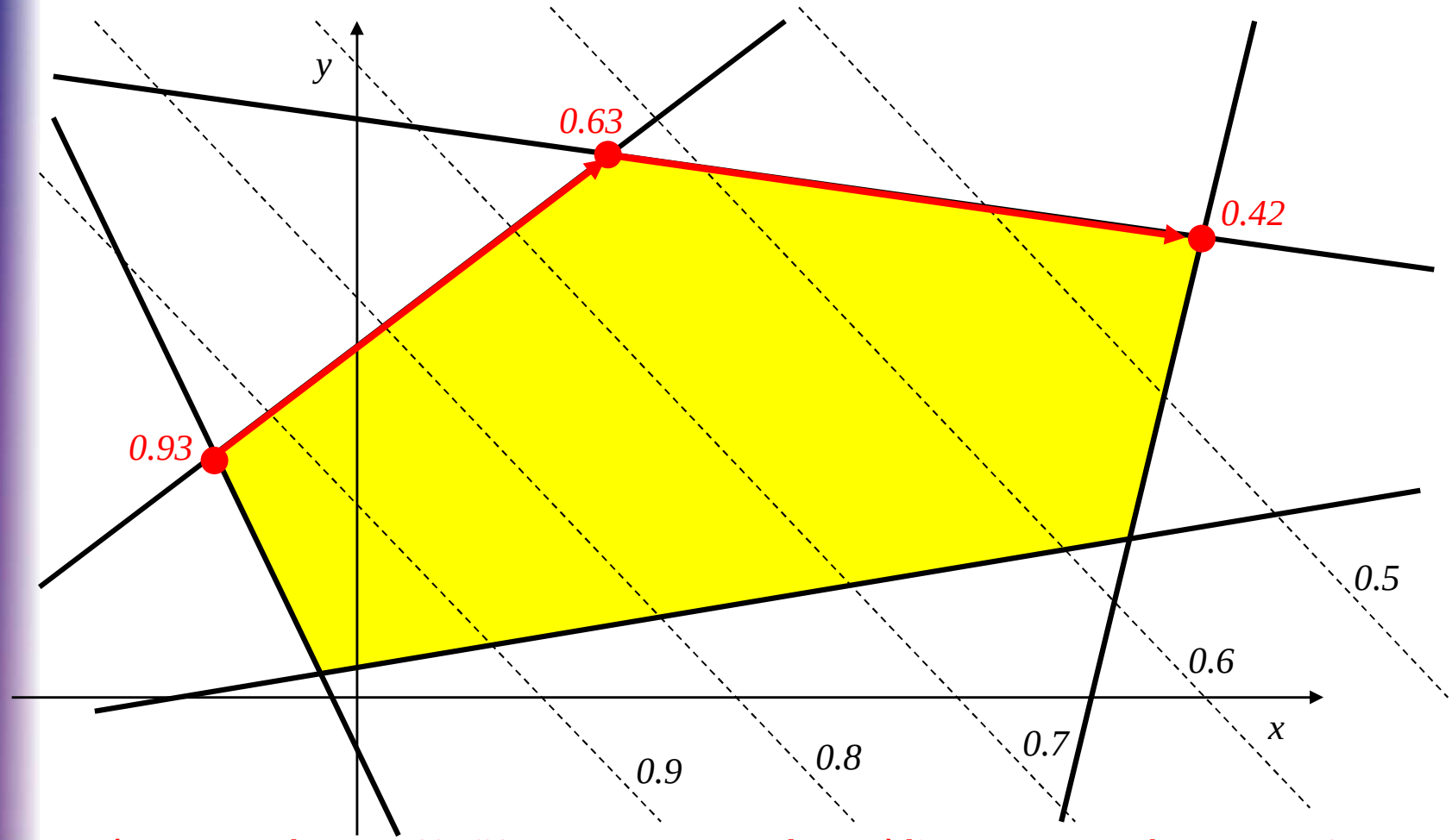
# Programmation linéaire

- Problème décrit sous forme d'inéquations linéaires et d'une fonction de coût linéaire, portant sur des variables continues.
- Formellement : $\forall j \in [1, L], \sum_{i=1}^{C} a_{i,j} x_i \leq b_j$

$$\begin{cases} a_{i,j} & \text{constantes} \\ b_j & \text{constantes} \\ x_i & \text{variables} \end{cases}$$

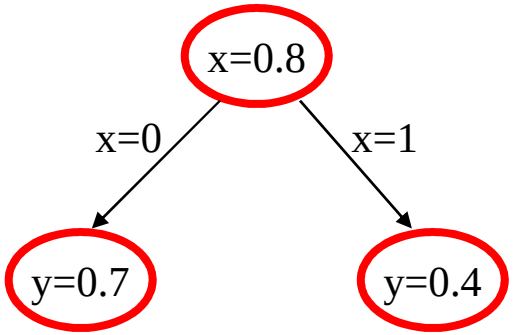
  - Ex. : $3x + 4y + 5z < 6$
- Algorithme du simplexe.
- Solution optimale (i.e., minimum absolu de la fonction de coût).

# PL - simplexe



# PL en nombres entiers

- Problème décrit comme précédemment, mais certaines variables sont entières (e.g., booléennes).
- Le problème devient NP-complet.
- Méthode : approximer par petites touches.
  - Résoudre le problème en variables continues (solution relâchée).
  - Séparation / évaluation (*branch & bound*, *branch & cut*).
    - Quelle valeur donner à *x* si *x* vaut *0.8* dans la solution relâchée ? *0* ou *1* ? Et si *x* vaut *0.5* ?



# PLNE - Exemple


$x_{i,j}$ vaut 1 ssi la tâche *i* est sur la vacation *j*, et 0 sinon.

$$\forall i, \sum_{j \in [1,N]} x_{i,j} \leq 1$$

$$\forall (i_1, i_2)\, se\, recouvrant, \forall j \in [1, N],\ x_{i_1,j} + x_{i_2,j} < 1$$

$$\min(-\sum_{i,j} x_{i,j})$$

# PLNE - Exemple


$y_{i,j}$ vaut 1 ssi la tâche mobile de la vacation *i* commence à la date indicée *j*, et 0 sinon.

$$\forall i, \sum_j y_{i,j} = 1$$

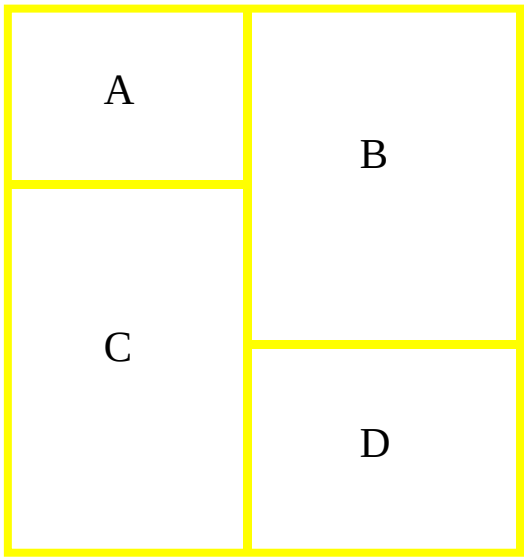
$$\forall (i,k) \; se \; recouvrant, \forall j, \; x_{i,j} + y_{j,k} < 1$$

# Programmation par contraintes

- Problème représenté par des variables à domaine fini et des contraintes (quelconques) reliant ces variables.
- Une solution est l'affectation de chaque variable à une valeur de son domaine, de façon à satisfaire toutes les contraintes.
- Importance des heuristiques (sur les variables, sur les valeurs) pour obtenir de bonnes performances.

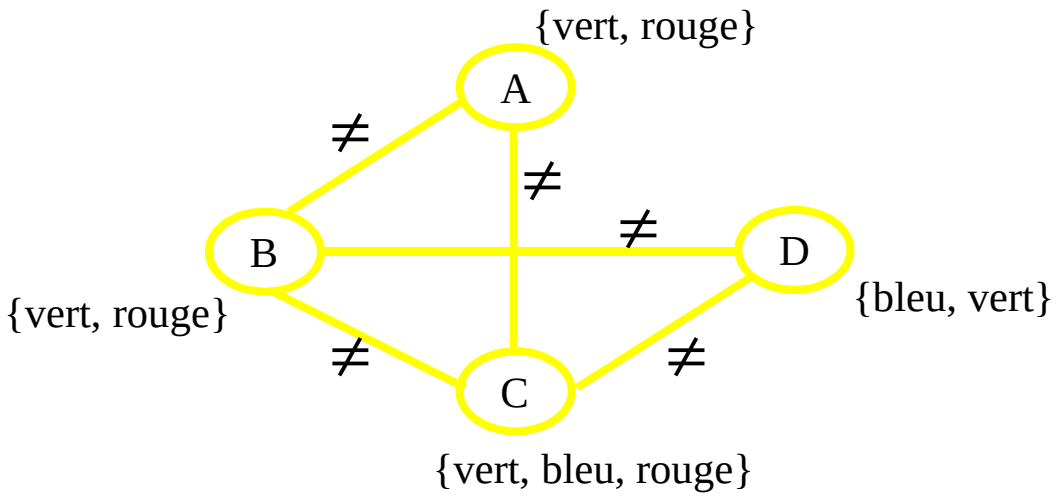
# PPC - Exemple : coloration de graphe

Colorier A,B,C et D (ci-dessous) sans que deux couleurs ne se touchent, avec les couleurs possibles pour chaque case :




A et B sont verts ou rouges
C est vert, bleu ou rouge
D est bleu ou vert

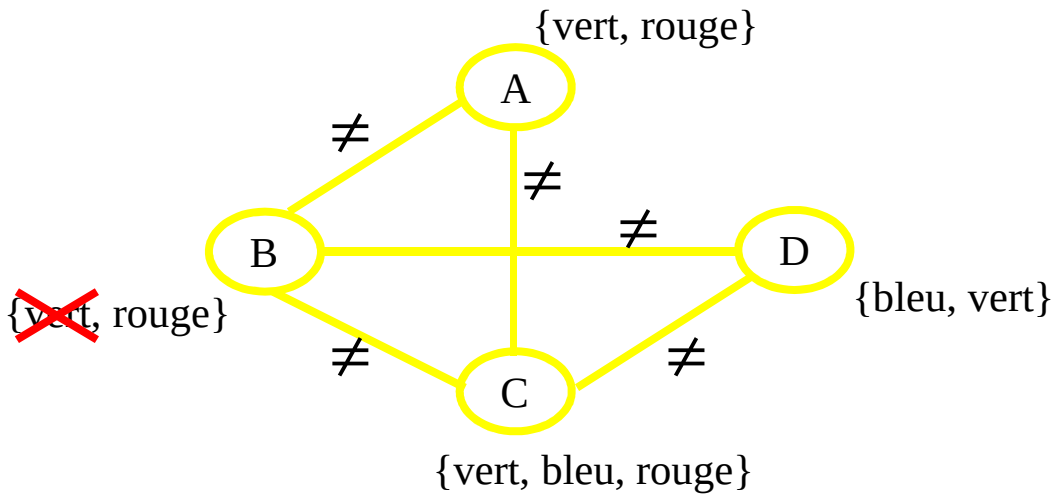
# PPC - Exemple : modélisation




- Chaque couple de variables est relié par une contrainte symétrique de différence.
- Le domaine de chaque variable apparaît entre accolades {...}.

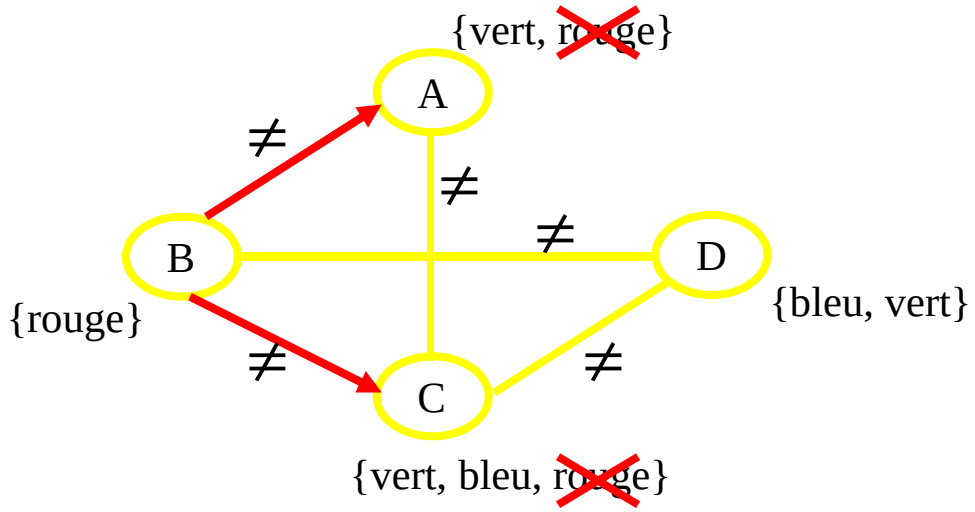
# PPC - Exemple : résolution

- Supposons que l'on fixe arbitrairement B à rouge.



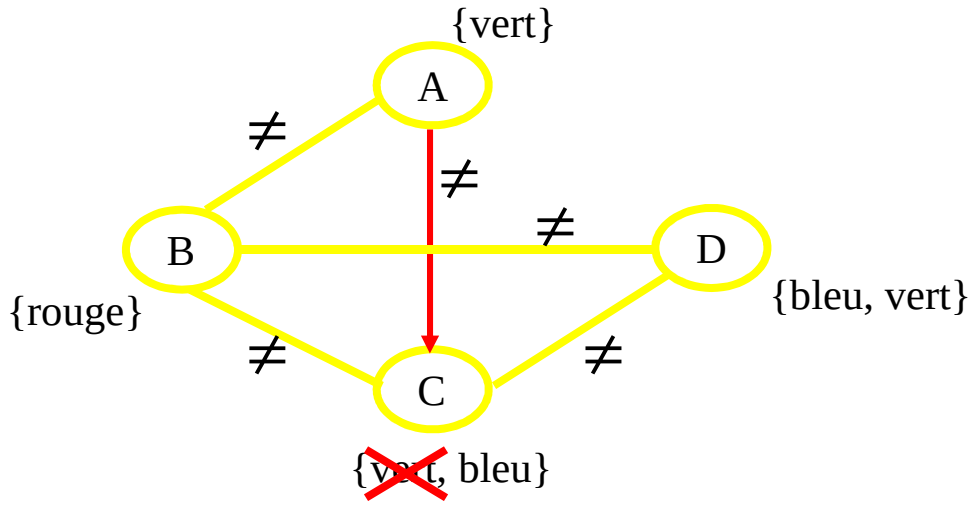
# PPC - Exemple : résolution

- Propagation : instantiation de A à vert.



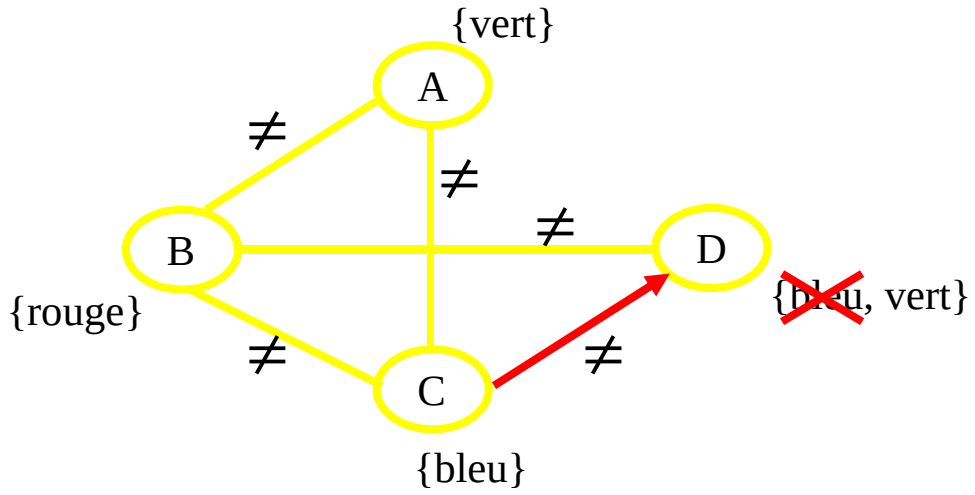
# PPC - Exemple : résolution

- Propagation : instantiation de C à bleu.



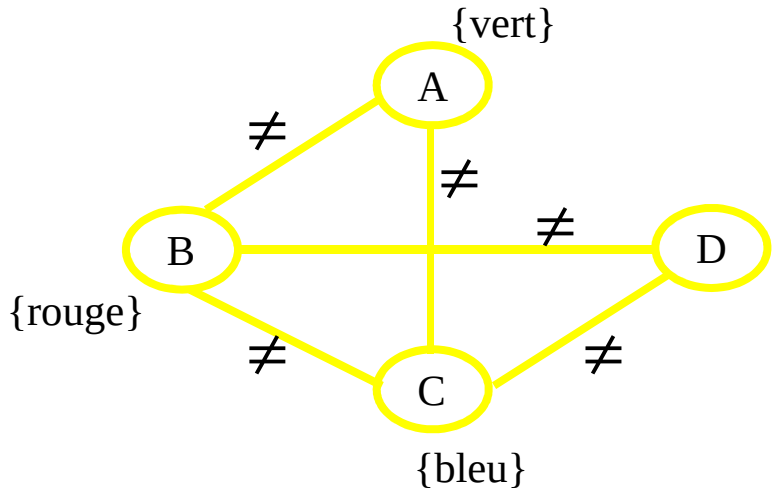
# PPC - Exemple : résolution

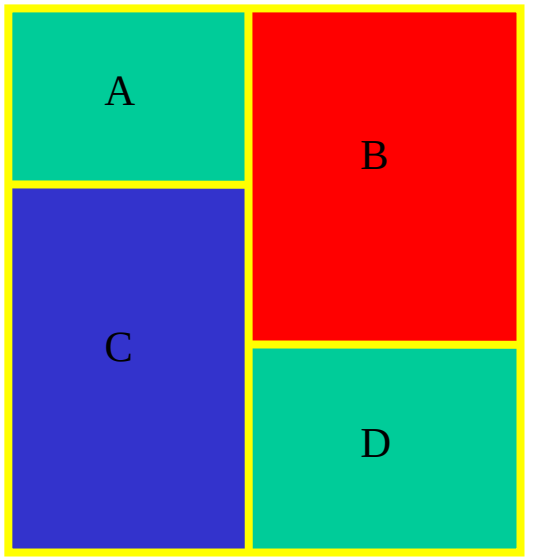
- Propagation : instantiation de D à vert.



# PPC - Exemple : résolution

- Propagation : solution !

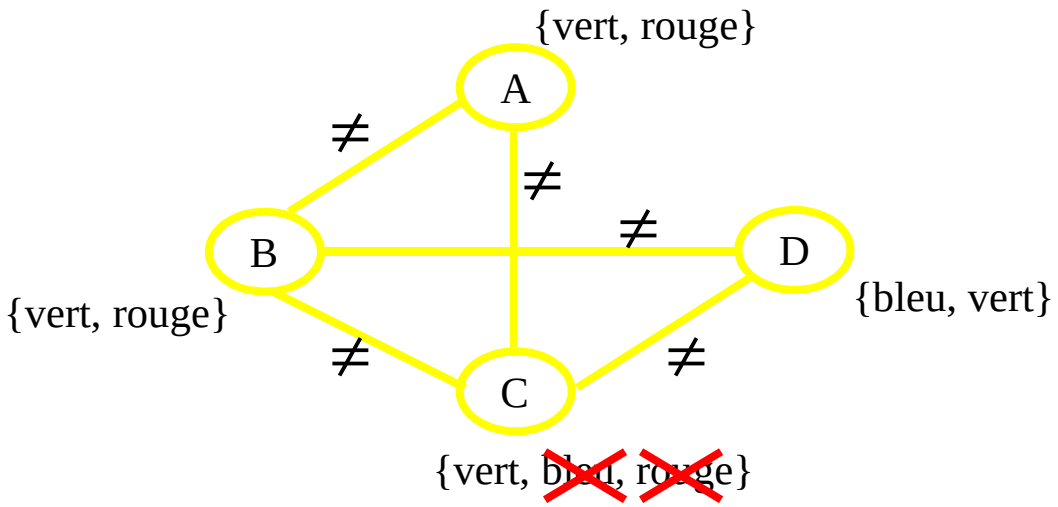



{vert}

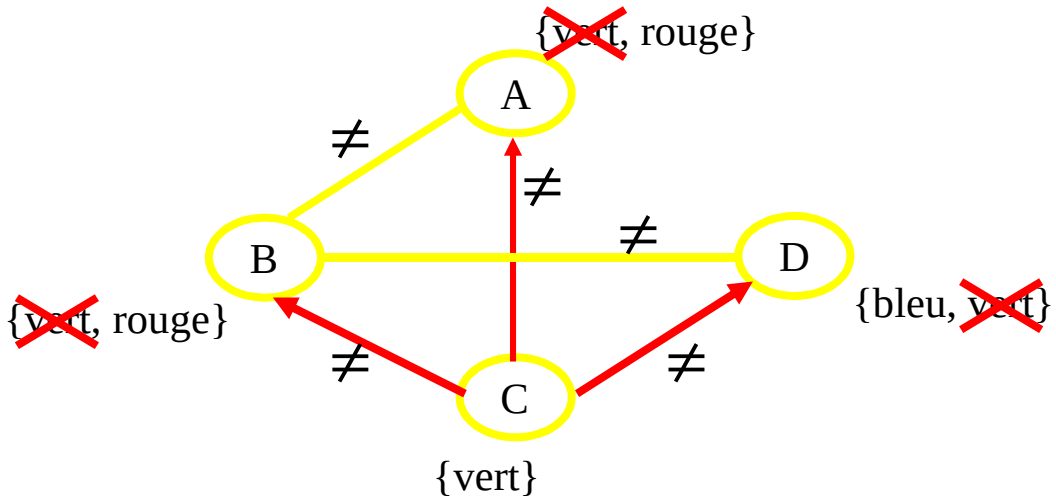
# PPC - Exemple : résolution (2)

- Supposons que l'on fixe arbitrairement C à vert.



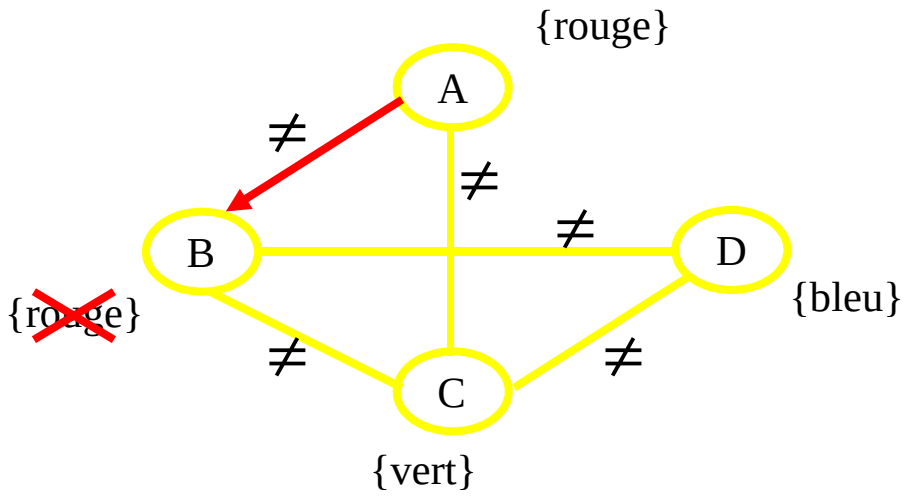
# PPC - Exemple : résolution (2)

- Propagation : instantiation de D à bleu, de A et B à rouge.



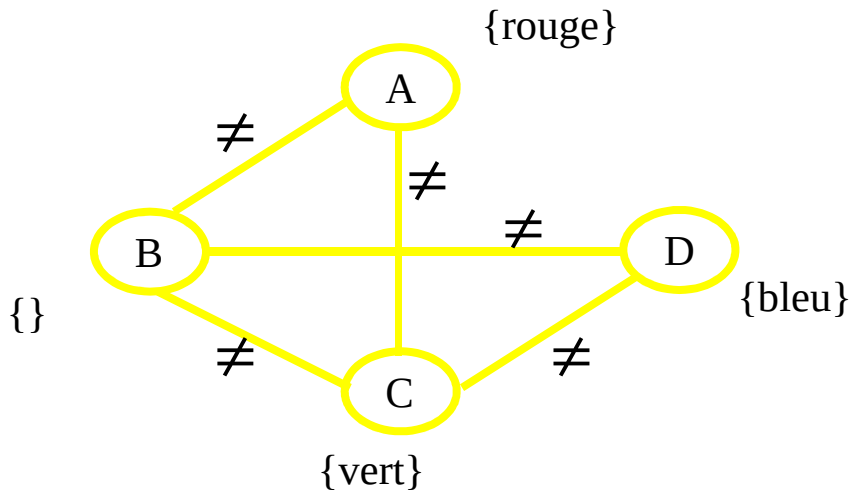
# PPC - Exemple : résolution (2)

- Propagation : enlèvement de rouge du domaine de B.



# PPC - Exemple : résolution (2)

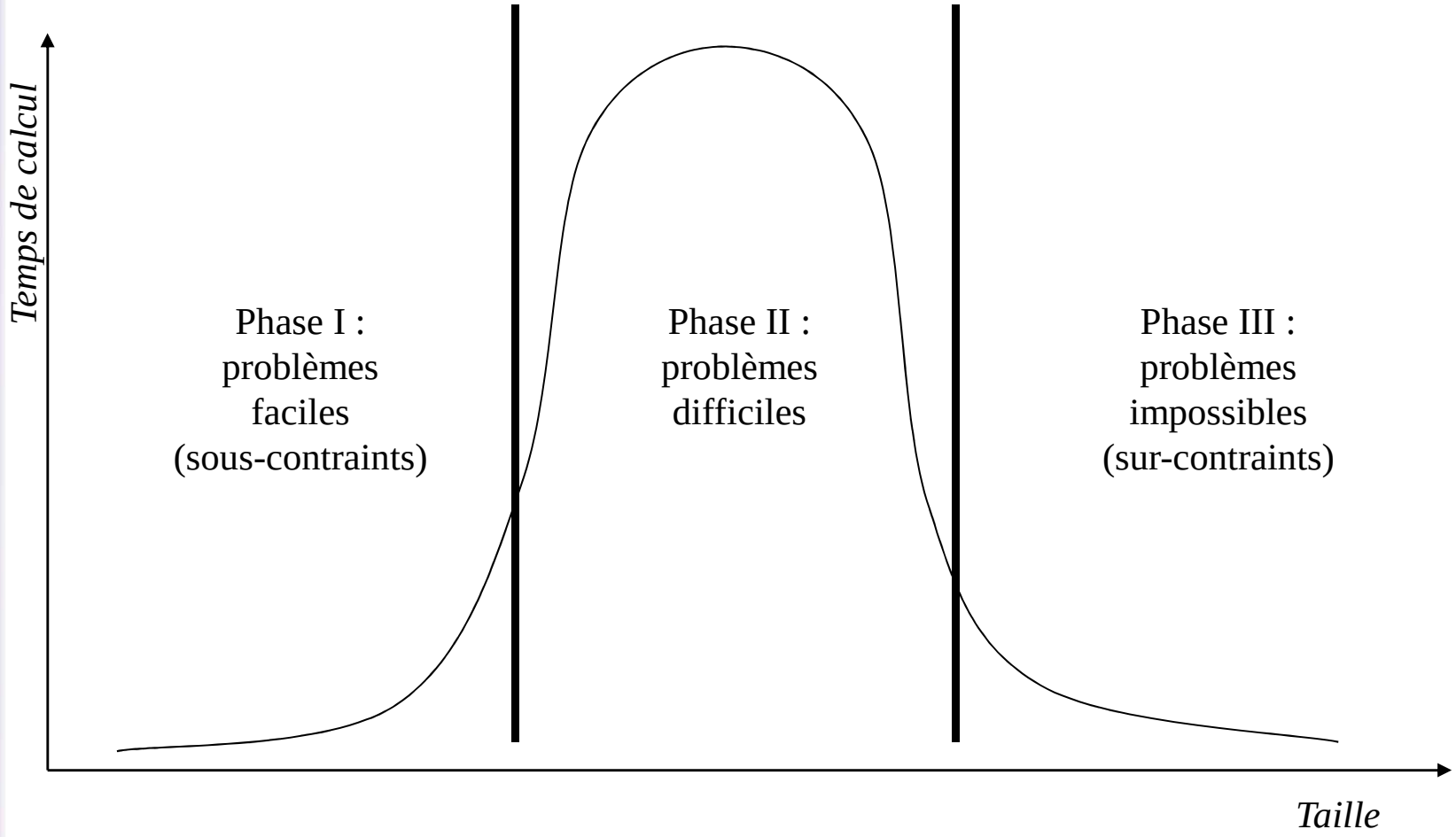
- Propagation : échec sur B ! Retour-arrière ...



# PPC - Algorithme

- 1) Choisir une variable
- 2) Choisir une valeur de son domaine
- 3) Instantier cette variable sur cette valeur
- 4) Propager les conséquences sur les autres variables via les contraintes (NP-complet).
- 5) Si échec, revenir sur un point de choix précédent et choisir une autre valeur/variable.
- 6) S'il reste une variable non instanciée, revenir en 1.
- 7) Solution.

# PPC - Transition de phase



# PPC - Exemple


Pour chaque tâche $i$, $x_i$ vaut l'indice de la vacation sur laquelle elle est affectée, ou 0.

$$\forall (i_1, i_2) \; se\ recouvrant, \; x_{i_1} \neq x_{i_2}$$

Pour chaque vacation $j$, $y_j$ vaut la date de début de la tâche mobile.

$$\forall i, (x_i \neq 0) \Rightarrow x_i(date\_fin) < y_i \vee y_i(date\_fin) < x_i(date\_debut)$$

$$\min[-\sum_i (x_i \neq 0)]$$

# Optimisation locale - Algorithmes

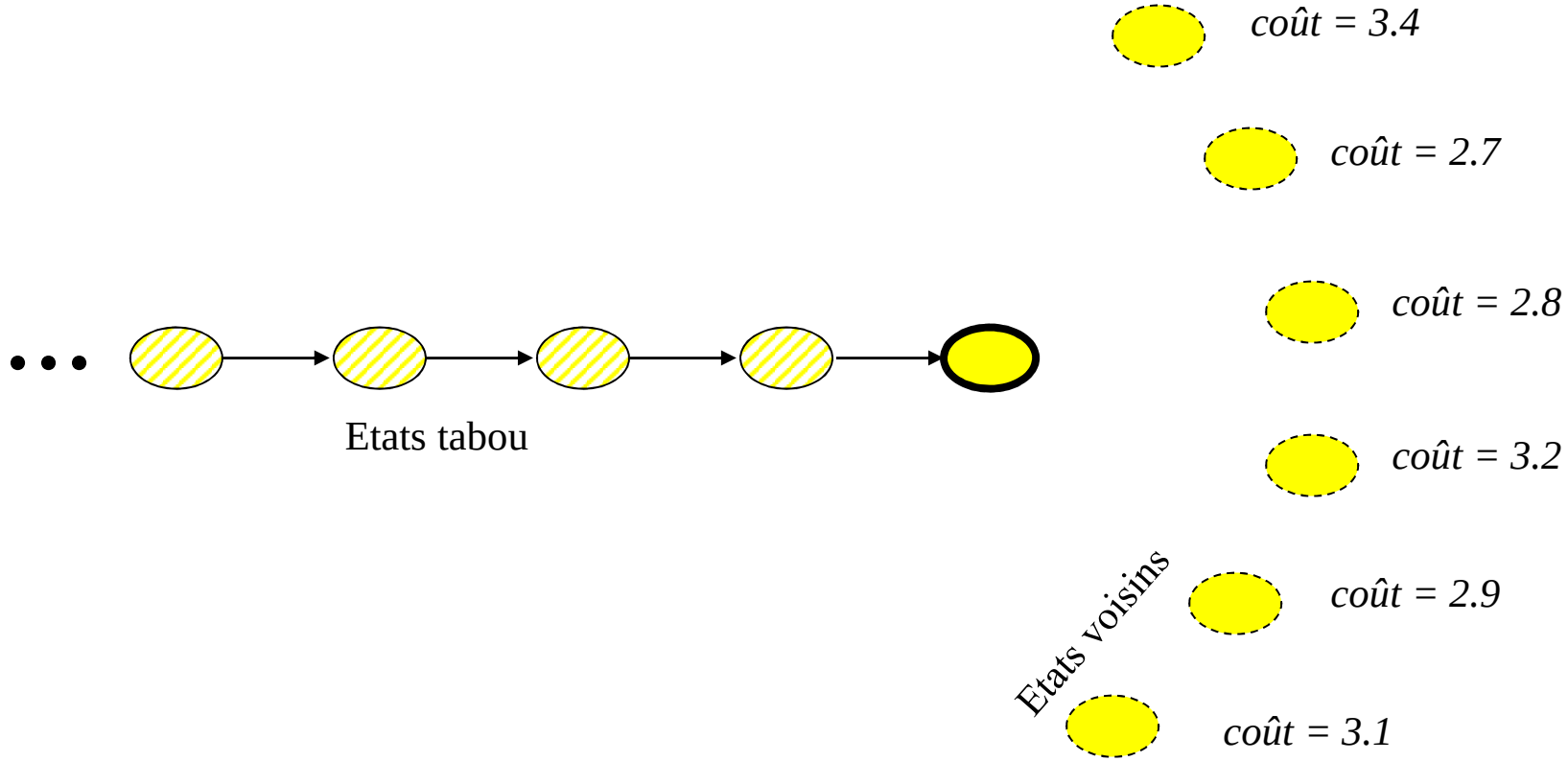
- Algorithme “tabou”
- Recuit simulé
- Algorithmes génétiques
- Algorithmes de fourmis
- Essaims particulaires

*Chaque auteur trouve que sa méthode est la meilleure ...*

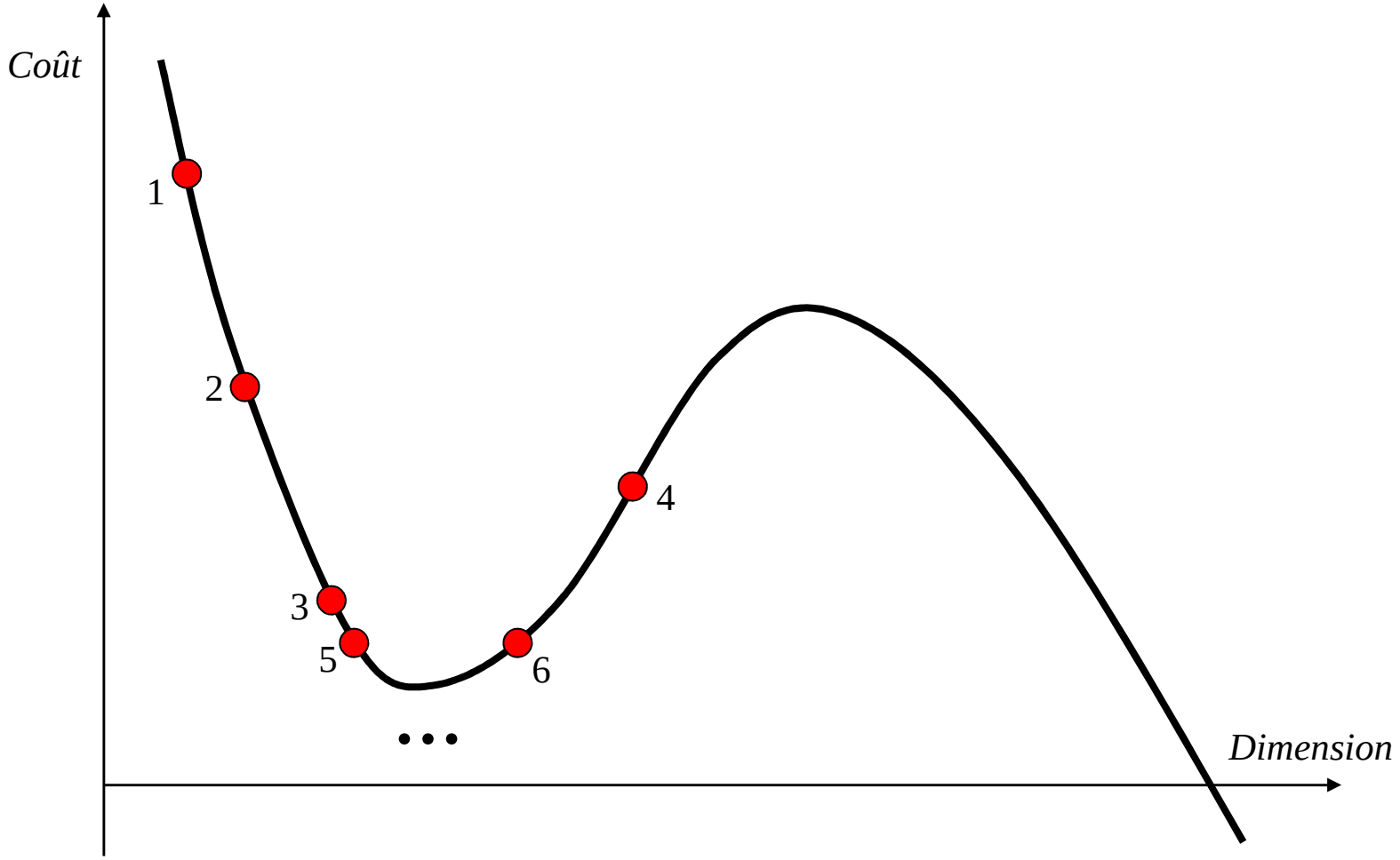
# Optim. locale

- Problème formalisé sous forme d'*états*, de *mouvements* entre états et d'une *fonction de coût*, pour rechercher un état-solution.
- Principe : à partir d'un état initial, générer itérativement les états *voisins* de l'état courant (i.e., états accessibles par un seul mouvement) pour diminuer progressivement le coût de l'état courant.
- Résolution par l'algorithme "*tabou*" (et ses nombreux dérivés) ou l'algorithme du *recuit simulé*.
- Recherche non complète (=> locale autour de l'état initial). Peut être piégée dans des minima locaux.

# Optimisation locale (2)



# Optim. locale - minima locaux



# Optim. locale - Exemple


Algorithme tabou avec :

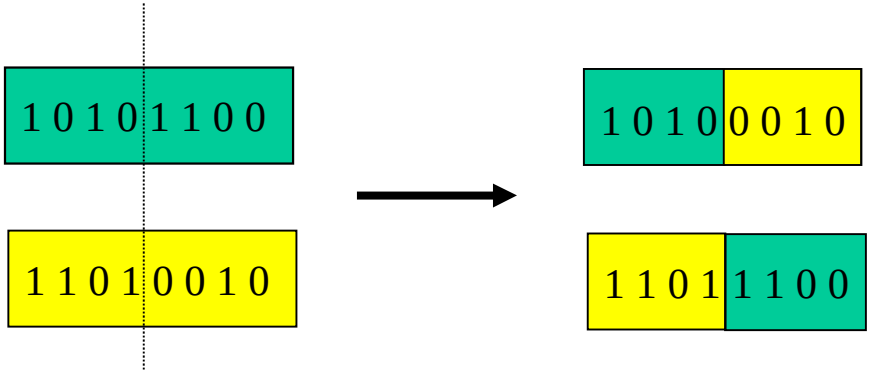
Un **état** = un planning
Un **mouvement élémentaire** = la permutation de (groupes de) tâches entre deux vacations.
**Fonction de coût** = qualité du planning, définie quantitativement par plusieurs critères.

# Algo. génétiques

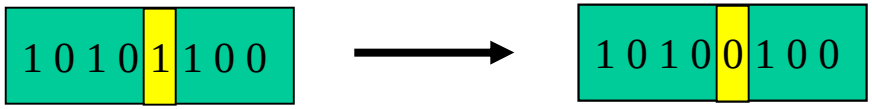
- Principe : dans une population d'individus disparates, simuler l'évolution naturelle, par croisement de gènes de parents pour obtenir des enfants mieux adaptés à leur environnement.
- Représentation :
  - Un codage du problème en un individu (son code génétique).
  - Un opérateur de **croisement**.
  - Un opérateur de **mutation**.
  - Une fonction de coût (adaptation à l'environnement).

# Algo. Génétiques - opérateurs

Croisement :




Mutation :

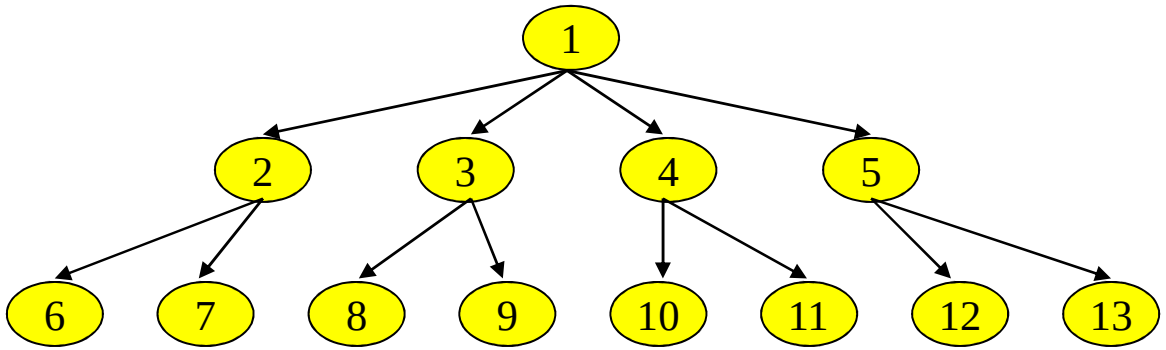



Codage du problème sur un individu (1) par position ou (2) par valeur.
Le croisement parcourt localement l'espace de recherche.
La mutation parcourt globalement l'espace de recherche.

# Algorithmes de recherche dans un espace d'états

- Problème défini sous forme d'états, d'une fonction de succession et d'une fonction solution.
- Principe : à partir d'un état initial (racine), parcourir l'arbre des états selon la relation de succession, à la recherche d'un état solution.
- Plusieurs algorithmes de parcours (heuristiques ou aveugles).
- NP-complet. Algorithme complet ou non suivant le parcours.

# Algorithmes de recherche dans un espace d'états (2)




Profondeur d'abord : 1, 2, 6, …
Largeur d'abord : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, …
Meilleur d'abord : 1, 2, 3, 4, 5, 8, 9, … (selon l'heuristique)
A* : 1, 2, 3, 4, 5, 10, 11, … (selon l'heuristique)

# OUTILS

# Programmation linéaire

- ILOG CPLEX
  - rapide
  - inéquations modélisées directement, pas de matrice à rentrer !
  - Fonction de coût quadratique
- DASH XPRESS-MP
  - rapide (benchmarks pour le tester contre Cplex ?)
  - inéquations modélisées directement , pas de matrice à rentrer !
  - Fonction de coût quadratique
- OZ/MOZART-LP
- IBM LP-SOLVE
- Packages sur le web

# Programmation par contraintes

- ILOG SOLVER
  - complet, mais prix de l'ordre de 15 k€.
  - sous-classement des contraintes.
- COSYTEC CHIP
- INOVIA AQL
- SICSTUS PROLOG
- PROLOG IV
- Packages sur le web

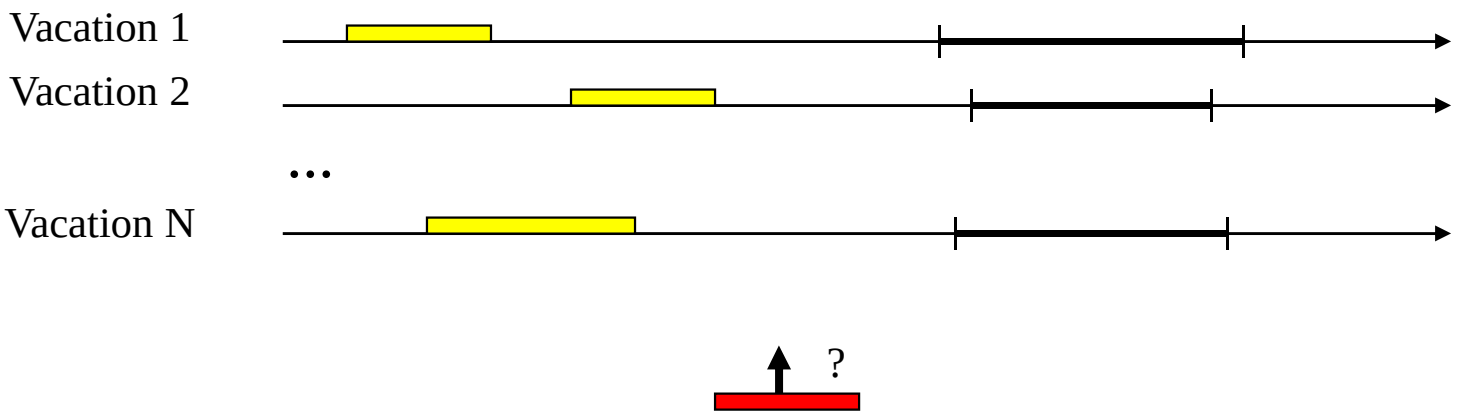
# Optimisation locale

- ILOG DISPATCHER (tournées de véhicules)
- algorithmes suffisamment simples pour les reprogrammer soi-même en 2 heures.

# Algorithme de recherche dans un espace d'états

- Packages sur le web …
- Algorithmes suffisamment simples pour les reprogrammer soi-même en 2 jours (pour A*) ou quelques heures (pour les autres).

# AREE - Exemple




Algorithme “largeur d’abord” avec :

Un **état** = un planning
Un **successeur** = la(les) tâche(s) libre(s) entre dans une vacation, ce qui fait sortir des tâches.
Une **solution** = pas de tâches libres.
(**Fonction de coût** = nombre de permutation par rapport à l’état initial).

# Conclusion

# Un problème d'ingéniérie

- Dans le problème, s'agit-il de variables discrètes ou continues ?
- Quelle est le dimensionnement du problème ?
  - Nombre d'objets manipulés ?
  - Nombre de variables ?
  - Nombre de contraintes/inéquations/relations ?
  - => calculer la taille du modèle
- Quel doit être l'ordre de grandeur du temps de réponse du futur système ? (quelques secondes ? une nuit ?)
- En cas de doute sur la méthode, prototypage concurrentiel.

# Synthèse

| Technique | Avantages | Inconvénients |
|---|---|---|
| *Programmation linéaire* | Optimum global<br>Beaucoup de vars. | Linéarité |
| *PLNE* | Variable de décision | Optimum local |
| *PPC* | Flexible | Bonnes heuristiques |
| *Optim. Locale* | Robuste<br>Souvent applicable | Optimum local |
| *AREE* | Peut etre complet<br>Souvent applicable | Peut etre incomplet |

# Conclusion

- Des problèmes combinatoires récurrents dans l'industrie. Un vaste marché.
- Surtout ne pas être technique en avant-vente !
- Développer éventuellement un modèle et une technique dans la proposition technique.
- Un vaste panorama de techniques, enseignées dans les écoles et universités (ISIMA, EPITA, options de 3e année, DEA, DESS à Avignon, …).

  $\Rightarrow$ Il ne reste plus qu'à trouver des contrats !!!

# Références


- M. Gondran. M. Minoux. Graphes ety algorithmes. *Programmation linéaire, entre autres*. ***
- O. Hudry, I. Charon, A. Germa. *Un peu tout*. ***
- F. Glover, Laguna. *Méthode tabou*. ***
- Nemhauser, Wolsey. *Programmation linéaire*. ***