Les Méthodes heuristiques

Chapitre 3

Khadija Assafra

Introduction

- Nous avons vu dans le chapitre 1 qu'une méthode d'optimisation pouvait être déterministe ou stochastique.
- Nous retrouvons également une opposition dans les termes "exacte" et "heuristique". En effet, l'utilisation de méthodes exactes n'est pas toujours possible pour un problème donné, par exemple à cause de temps de calcul trop important ou bien d'une séparation du problème impossible.
- Dans ces cas, nous utiliserons des méthodes approchées, appelées heuristiques. Il convient néanmoins de souligner qu'une méthode heuristique peut être déterministe ou stochastique.

Introduction

- Le mot heuristique vient du grec ancien eurisko ("je trouve") et qualie tout ce qui sert à la découverte, à l'invention et à la recherche. Ces méthodes exploitent au mieux la structure du problème considéré dans le but de trouver une solution approchée, de qualité "raisonnable", en un temps aussi faible que possible. Typiquement, elles trouvent une solution approchée à un problème NP en temps polynomial.
- On peut citer des heuristiques très simples comme les algorithmes gloutons [Cormen 90] [DeVore 96] ou les approches par amélioration itérative [Basili 75].

Introduction

- **Heuristiques**

Toute méthode approchée basée sur les propriétés structurelles ou sur les charactéristiques des solutions des problèmes, avec complexité inférieure à celles des algorithmes exactes et donnant, en général, des solutions réalisables de bonne qualité (sans garantie de haute qualité)

méthodes constructives
recherche locale
metaheuristiques

Méthodes constructives

- **Construction d'une solution:**
  Sélectionner séquentiellement des élements de E, éventuellement en préjudice d'autres déjà sélectionnés antérieurement, de façon à ce que à la fin on obtienne une solution réalisable, i.e. appartenant à F.

Méthodes constructives

- **Algorithmes gloutons**:
  Lors de la construction d'une solution, ce type d'algorithme choisi séquentiellement toujours l'élément de E qui minimise l'augmentation du coût de la solution partielle courante, de façon à ce que à la fin on obtienne une solution réalisable.

- L'augmentation dans le coût de la solution partielle est la fonction gloutonne.

Méthodes constructives

- **Problème de l'arbre minimum**:

  Etant donné un graphe connexe $G=(V,E)$ et des poids c_e associés aux arêtes $e \in E$, déterminer un arbre générateur $T \subseteq E$ dont le poids $c(T) = \Sigma_{e \in T} c_e$ soit minimum.

  E: ensemble d'arêtes
  F: sous-ensembles de E qui forment des arbres générateurs

Méthodes constructives

- Algorithme glouton pour la construction d'un arbre de poids minimum (Kruskal)

Etape 0:

Trier les arêtes de E de façon à ce que $c_1 \leq c_2 \leq \ldots \leq c_n$

$T \leftarrow \varnothing$

Etape 1:

Pour i **de** 1 **à** n **faire**

Si $T \cup \{e_i\} \in F$

alors $T \leftarrow T \cup \{e_i\}$

Note:

La solution T est une solution optimale

Fonction gloulonne: c_e (poids de l'arête)

Méthodes constructives

- Algorithme glouton randomisé (probabiliste)
  - Algorithme glouton obtient toujours la même solution pour un problème donné
  - Randomisation permet d'avoir de la diversité dans les solutions obtenues (*Le Petit Robert*: "randomisation" = "hasardisation")
  - Créer une liste de candidats L pour forcer un choix différent (aspect probabiliste) à chaque itération

- La qualité de la meilleure solution dépend de la qualité des éléments dans la liste de candidats

- La diversité des solutions obtenues dépend de la cardinalité de la liste L

- Cas extrêmes: - algorithme glouton pur
  - solution totalement aléatoire

Méthodes constructives

- Algorithme glouton randomisé: (minimisation)

Etape 0:

Trier les éléments de E de façon à ce que

$c_1 \leq c_2 \leq \ldots \leq c_n$

$S \leftarrow \emptyset$

Etape 1:

Pour i **de** 1 **à** n **faire**

Créér une liste $L \subseteq \{1, 2, \ldots, n\} \setminus S$

telle que $S \cup \{e\} \in F, \forall e \in L$

Choisir au hasard un élément $e \in L$

Si $S \cup \{e\} \in F$

alors $S \leftarrow S \cup \{e\}$

Représentation de solutions

- Ensemble F de solutions (réalisables) est un sous-ensemble de E (ensemble de support ou de base) des éléments qui satisfont certaines conditions (contraintes).

- Représentation d’une solution: indiquer les éléments de E qui appartiennent à la solution et ceux qui n’y appartiennent pas.

- Problème du sac-à-dos: n objets, vecteur 0-1 de n éléments, $x_j = 1$ si l’objet j appartient à la solution, $x_j = 0$ sinon

Représentation de solutions

- Problème du voyageur de commerce

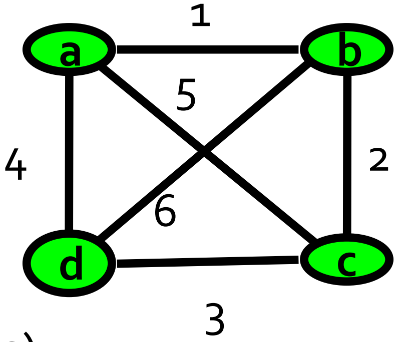
  E: ensemble d'arêtes

  F: sous-ensembles de E qui forment un circuit hamiltonien (CH)

  Une solution est un vecteur de n = |E| éléments:

  $v_e = 1$, si l'arête e appartient au CH

  $v_e = 0$, sinon.



Solutions réalisables (parmi 64 possibilités):

(1,1,1,1,0,0), (1,0,1,0,1,1), (0,1,0,1,1,1)

Représentation de solutions

- Autre représentation: pour les solutions du problème du voyageur de commerce: représenter chaque solution par l'ordre dans laquelle les sommets sont visités, c.à.d., comme une permutation circulaire des n sommets (puisque le premier sommet peut être choisi arbitrairement)

(abcd)	**(abdc)**
(acbd)	**(acdb)**
(adbc)	**(adcb)**

Représentation de solutions

- Indicateurs 0-1 d'appartenance:
  - Problème du sac-à-dos
  - Problème de Steiner dans les graphes
  - Problèmes de recouvrement et de partitionement
- Indicateurs généraux d'appartenance:
  - Partitionement de graphes
  - Coloration de graphes
  - *Clustering*
- Permutations:
  - Problèmes d'ordonnancement
    - *Job/Flow/Open Shop Scheduling*
  - Problème du voyageur de commerce

Voisinages

- Problème combinatoire:
  $f(s^*) = \text{minimum } \{f(s) : s \in S\}$
  S est un ensemble fini de solutions

- **Voisinage**: élément qui introduit la notion de proximité entre les solutions de S.

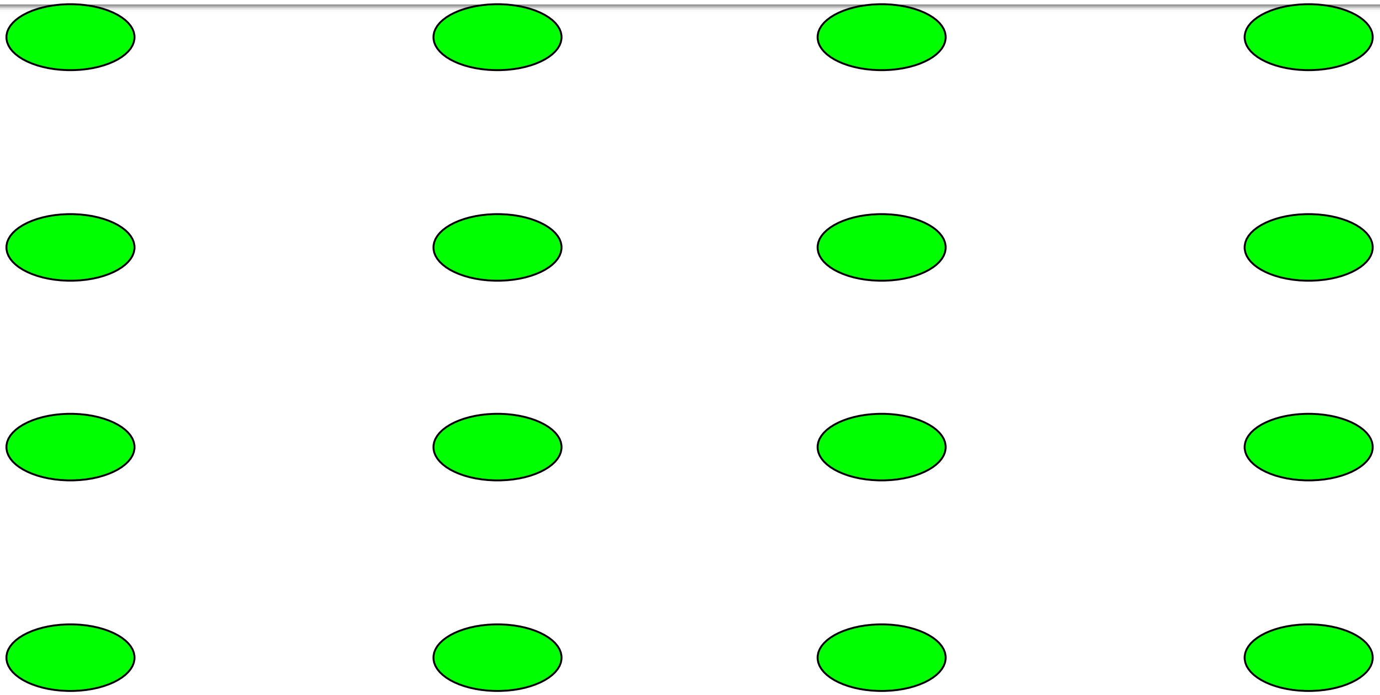
- Le voisinage d'une solution $s \in S$ est un sous-ensemble de S:
  $N : S \to 2^S$
  $N(s) = \{s_1, s_2, \ldots, s_k\}$ **solutions voisines** de s

- Les bons voisinages permettent de représenter de forme efficace et compacte l'ensemble des solutions voisines à toute solution s .

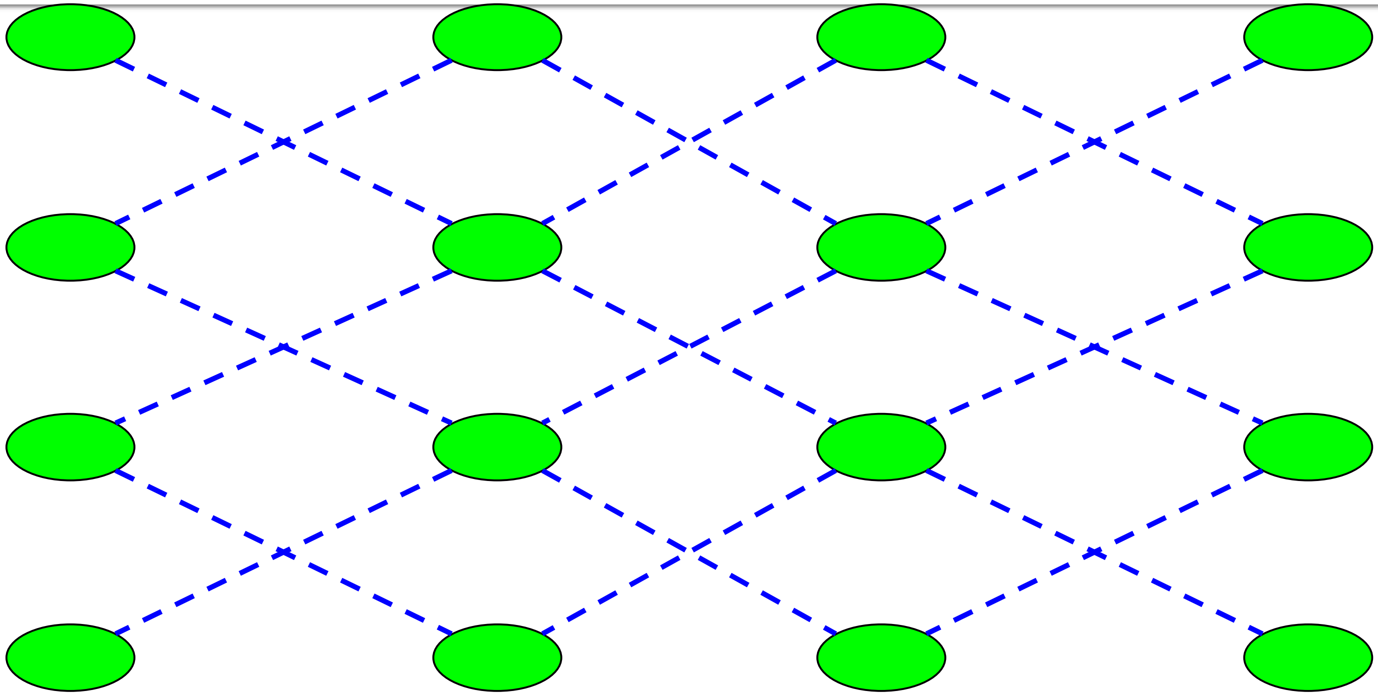
Voisinages

- Voisinages dans l'espace des permutations:
- Solution $\pi=(\pi_1,\ldots,\pi_{i-1},\pi_i,\pi_{i+1},\ldots,\pi_j,\ldots,\pi_n)$
- $N1(\pi)=\{(\pi_1,\ldots,\pi_{i+1},\pi_i,\ldots,\pi_n): i=1,\ldots, n-1\}$
  Voisins de (1,2,3,4) = {(2,1,3,4),(1,3,2,4), (1,2,4,3)}
- $N2(\pi)=\{(\pi_1,\ldots,\pi_j,\ldots,\pi_i,\ldots,\pi_n): \quad i=1,\ldots,n-1; j=i+1,\ldots,n\}$
  Voisins de (1,2,3,4)= {(2,1,3,4),(1,3,2,4),
  (1,2,4,3),(3,2,1,4),(1,4,3,2),(4,2,3,1)}
- $N3(\pi)=\{(\pi_1,\ldots, \pi_{i-1},\pi_{i+1},\ldots,\pi_j,\pi_i,\ldots,\pi_n): i=1,\ldots,n-1; j=i+1,\ldots,n\}$
  Voisins de (1,2,3,4)= {(2,1,3,4),(2,3,1,4),
  (2,3,4,1),(1,3,2,4),(1,3,4,2),(1,2,4,3)}

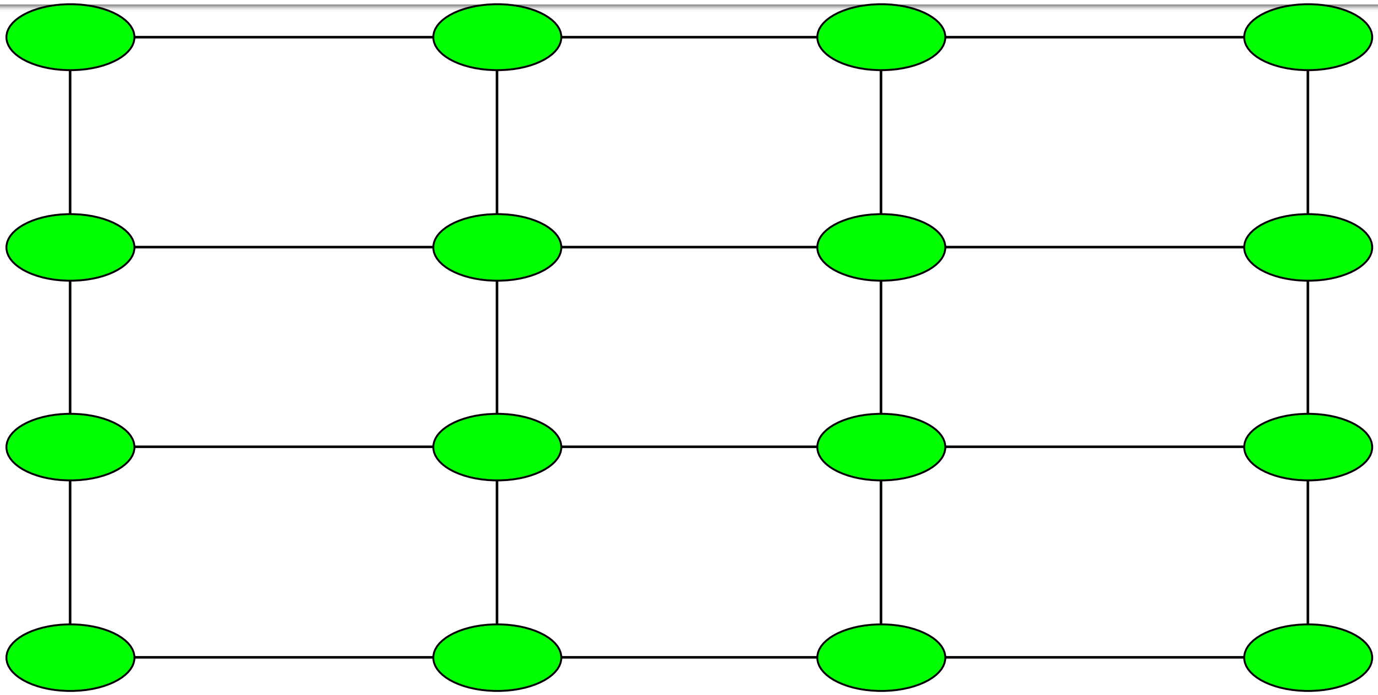
Voisinages



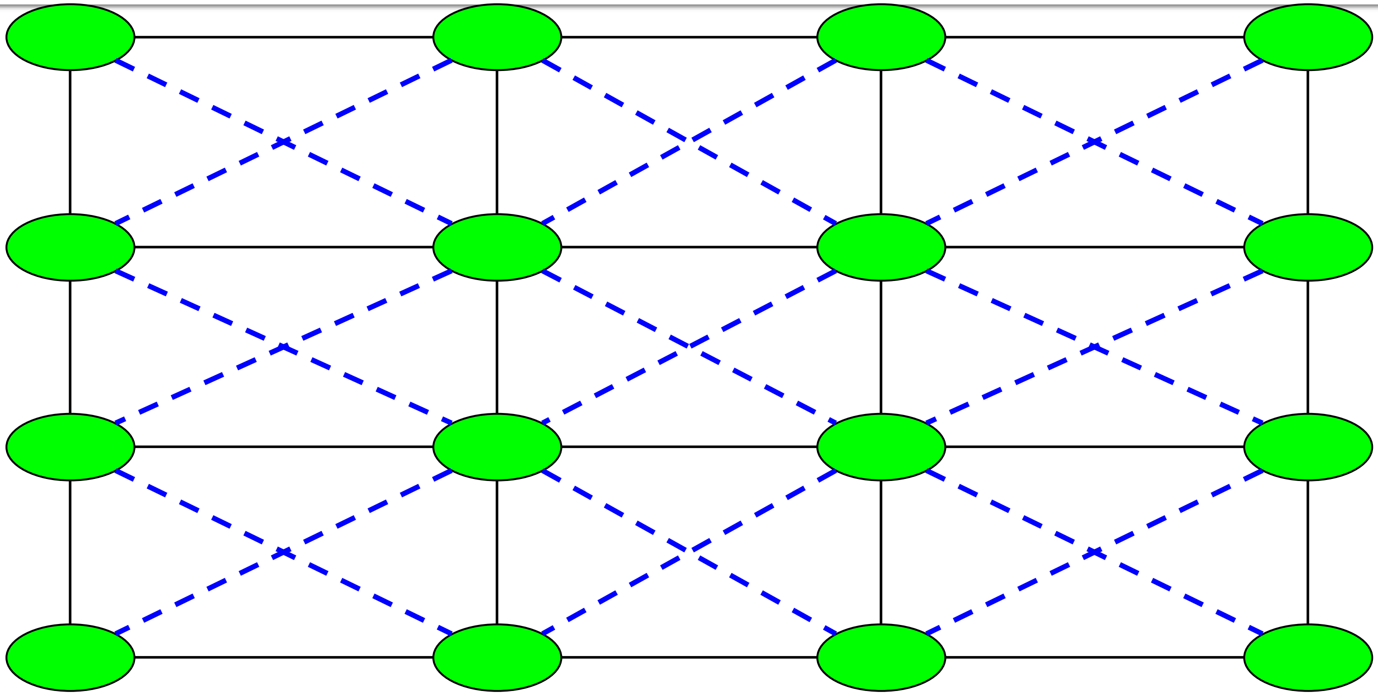
Voisinages



Voisinages



Voisinages



Voisinages

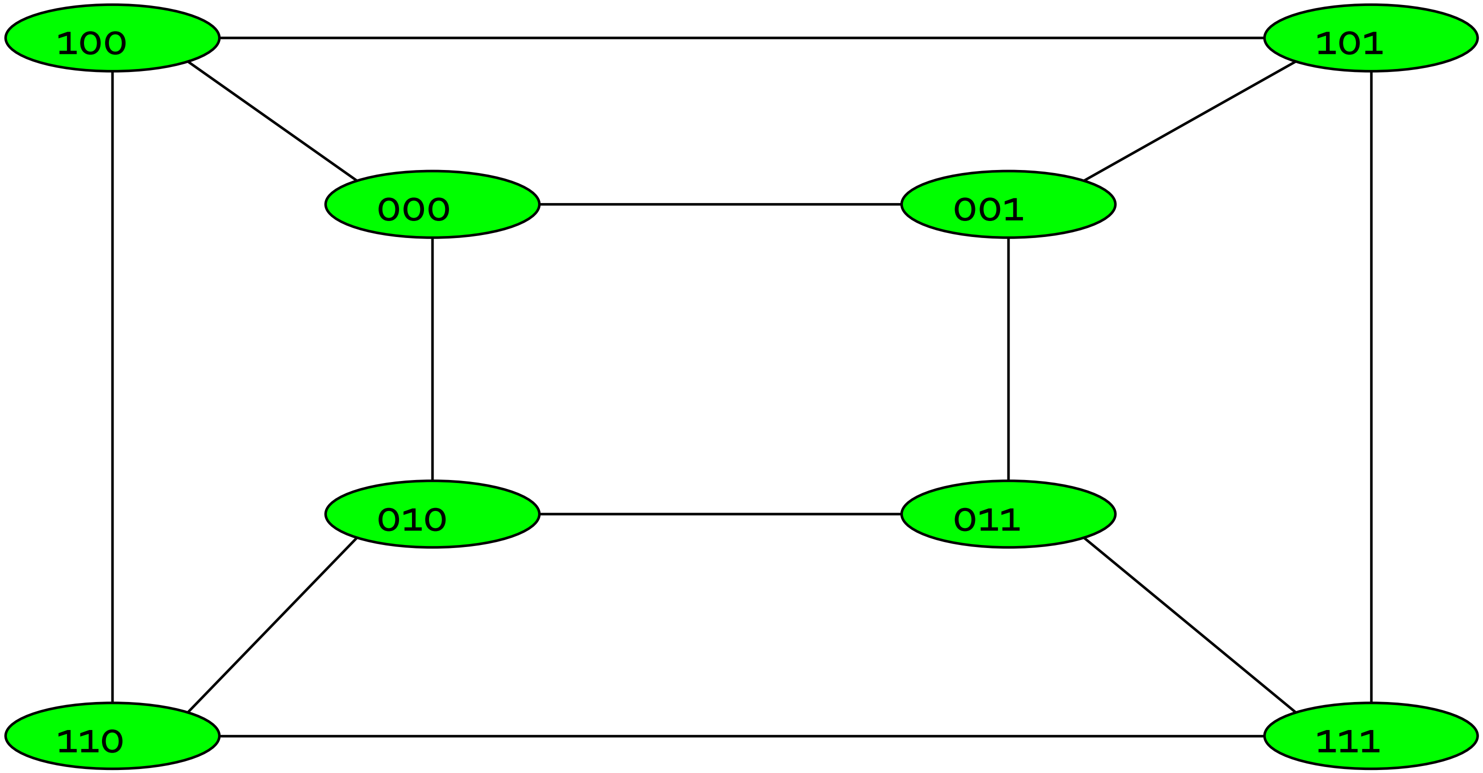
- Espace de recherche: défini par l'ensemble de solutions S et par un voisinage N

- Exemple 1: vecteurs d'appartenance 0-1

$v=(v_1,\ldots,v_i,\ldots,v_n)$

$N4(v)=\{(v_1,\ldots,1-v_i,\ldots,v_n): i=1,..,n\}$

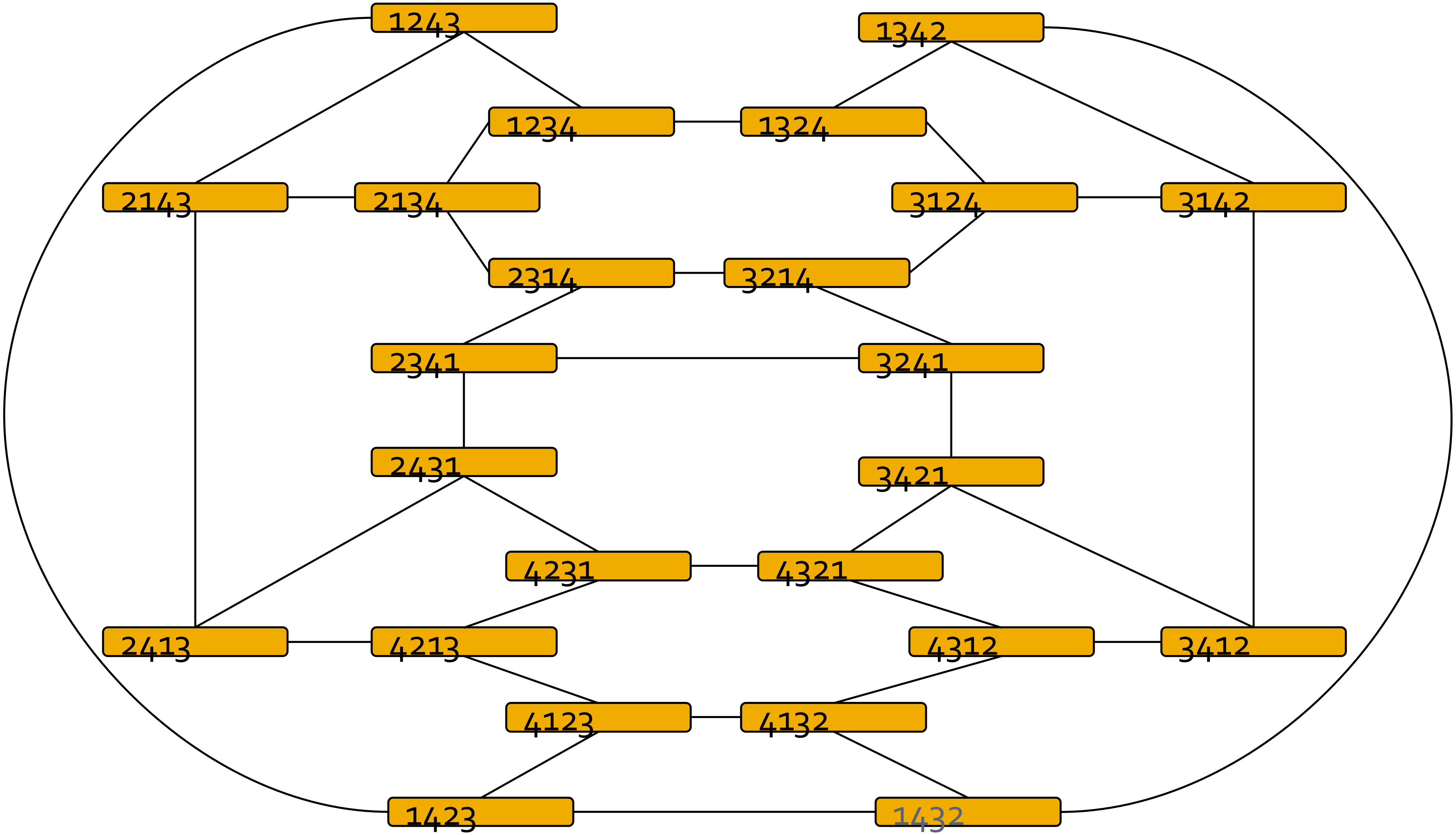
Voisins de (1,0,1,1)=
{(0,0,1,1),(1,1,1,1),(1,0,0,1),(1,0,1,0)}


100
101
000
001
010
011
110
111

Voisinages

- **Espace de recherche**: défini par l'ensemble de solutions S et par un voisinage N

- Exemple 2:permutations avec le voisinage N1


1243
1342
1234
1324
2143
2134
3124
3142
2314
3214
2341
3241
2431
3421
4231
4321
2413
4213
4312
3412
4123
4132
1423
1432

FIGL1AB

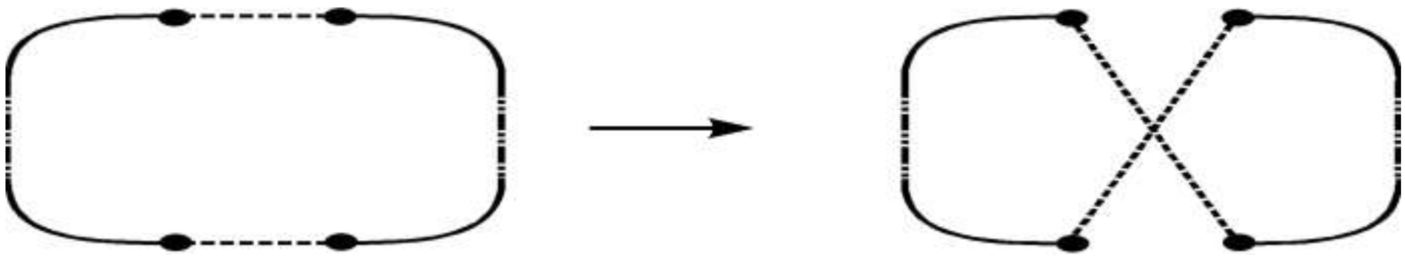
Voisinages

- Espace de recherche: défini par l'ensemble de solutions S et par un voisinage N

- Exemple 3:
  Voisinages 2-opt et 3-opt pour le voyageur de commerce

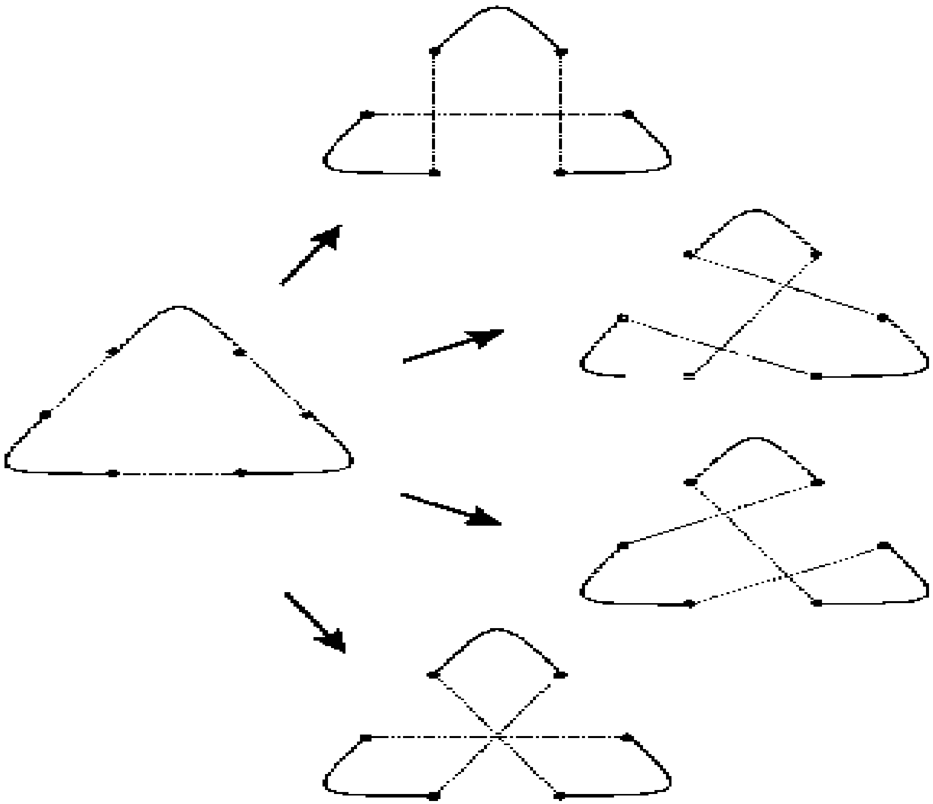
Voisinages

- Voisinage 2-opt pour le voyageur de commerce:



Voisinages

- Voisinage 3-opt pour le voyageur de commerce:



Voisinages

- L'**espace de recherche peut être vu comme un graphe** où les sommets représentent les solutions et les arêtes connectent les paires de solutions voisines.

- Cet espace peut être vu aussi comme une surface avec des cols et des sommets, définis par la valeur des solutions et par la proximité (les relations de voisinage) entre elles.

- Un **chemin** dans l'espace de recherche est une séquence de solutions, deux solutions consécutives étant toujours voisines.

Voisinages

- **Optimum local**: une solution aussi bonne que toutes les voisines

- Problème de minimisation:

$$s^{+} \text{ est un optimum local}$$
$$\Updownarrow$$
$$f(s^{+}) \leq f(s), \ \forall s \in N(s^{+})$$

- **Optimum global** (solution optimale) s^*:

$$f(s^*) \leq f(s), \quad \forall s \in S$$

Algorithmes de recherche locale

- Les algorithmes de recherche locale sont conçus comme une stratégie pour explorer l'espace de recherche.
- Départ: solution initiale obtenue par une méthode constructive
- Itération: amélioration de la solution courante par une recherche dans son voisinage
- Arrêt: premier optimum local trouvé (il n'y a pas de solution voisine meilleure que la solution courante)
- Heuristique subordonnée utilisée pour obtenir une solution meilleure dans le voisinage de la solution courante.

Algorithmes de recherche locale

- Questions fondamentales:
  - Définition du voisinage
  - Stratégie de recherche dans le voisinage
  - Complexité de chaque itération:
    - Dépend du nombre de solutions dans le voisinage
    - Dépend du calcul efficace du coût de chaque solution voisine

Algorithmes de recherche locale

- **Amélioration itérative**: à chaque itération, sélectionner dans le voisinage une solution meilleure que la solution courante

procedure Amélioration-Itérative(s_o)
  $s \leftarrow s_o$; amélioration $\leftarrow$ .vrai.
  while amélioration **do**
    amélioration $\leftarrow$ .faux.
    for-all $s' \in N(s)$ **e** amélioration =.faux. **do**
      if $f(s') < f(s)$ **then**
        $s \leftarrow s'$; amélioration $\leftarrow$ .vrai.
    end-if
    end-for-all
  end-while
  return s
end Amélioration-Itérative

Algorithmes de recherche locale

- Descente la plus rapide: à chaque itération, sélectionner la meilleure solution du voisinage

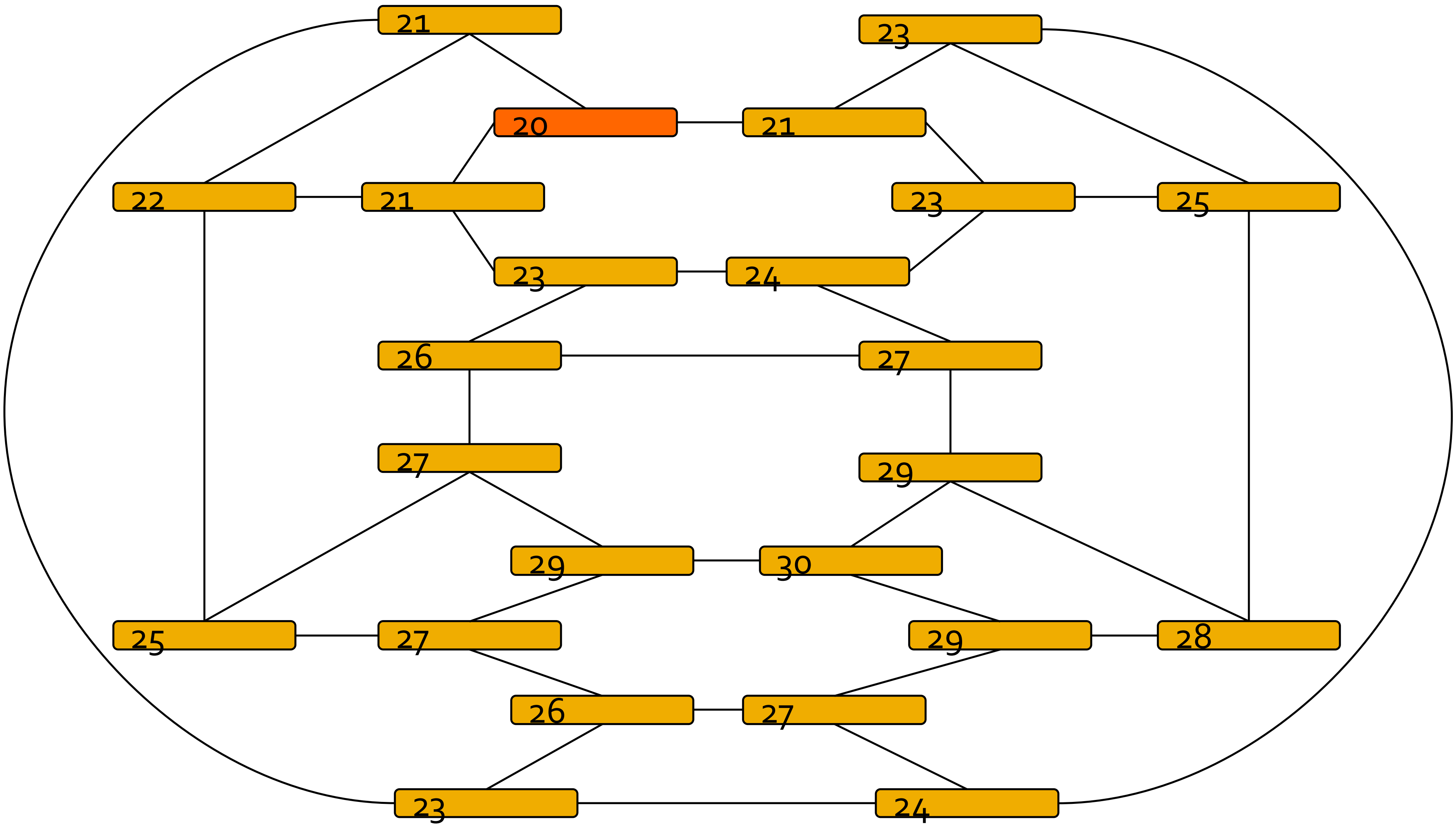
procedure Descente-Rapide(s_o)
  $s \leftarrow s_o$; amélioration $\leftarrow$ .vrai.
  while amélioration **do**
    amélioration $\leftarrow$ .faux.; $f_{min} \leftarrow +\infty$
    for-all $s' \in N(s)$ **do**
      if $f(s') < f_{min}$ **then**
        $s_{min} \leftarrow s'$; $f_{min} \leftarrow f(s')$
      end-if
    end-for-all
    if $f_{min} < f(s)$ **then**
      $s \leftarrow s_{min}$; amélioration $\leftarrow$ .vrai.
    end-if
  end-while
  return s
end Descente-Rapide

Algorithmes de recherche locale

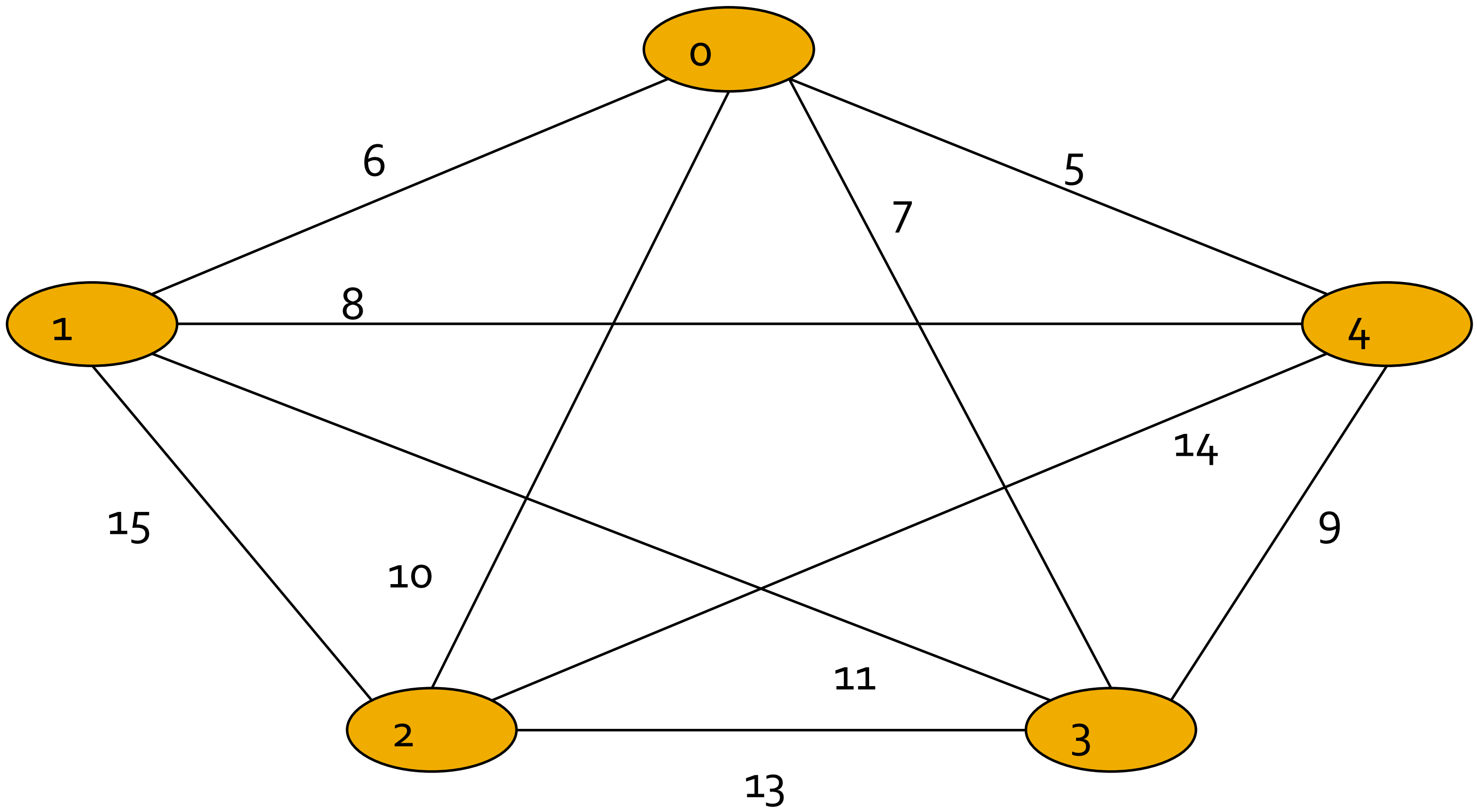
- Exemple: algorithme de descente la plus rapide appliqué à un problème d'ordonnancement
- Espace de recherche: permutations des n éléments
- Solution $\pi=(\pi_1,\ldots,\pi_{i-1},\pi_i,\pi_{i+1},\ldots,\pi_j,\ldots,\pi_n)$
- Voisinage: $N1(\pi)=\{(\pi_1,\ldots,\pi_{i+1},\pi_i,\ldots,\pi_n): i=1,\ldots, n-1\}$
- Coût d'une permutation: $f(\pi) = \sum_{i=1,\ldots,n} i.\pi_i$

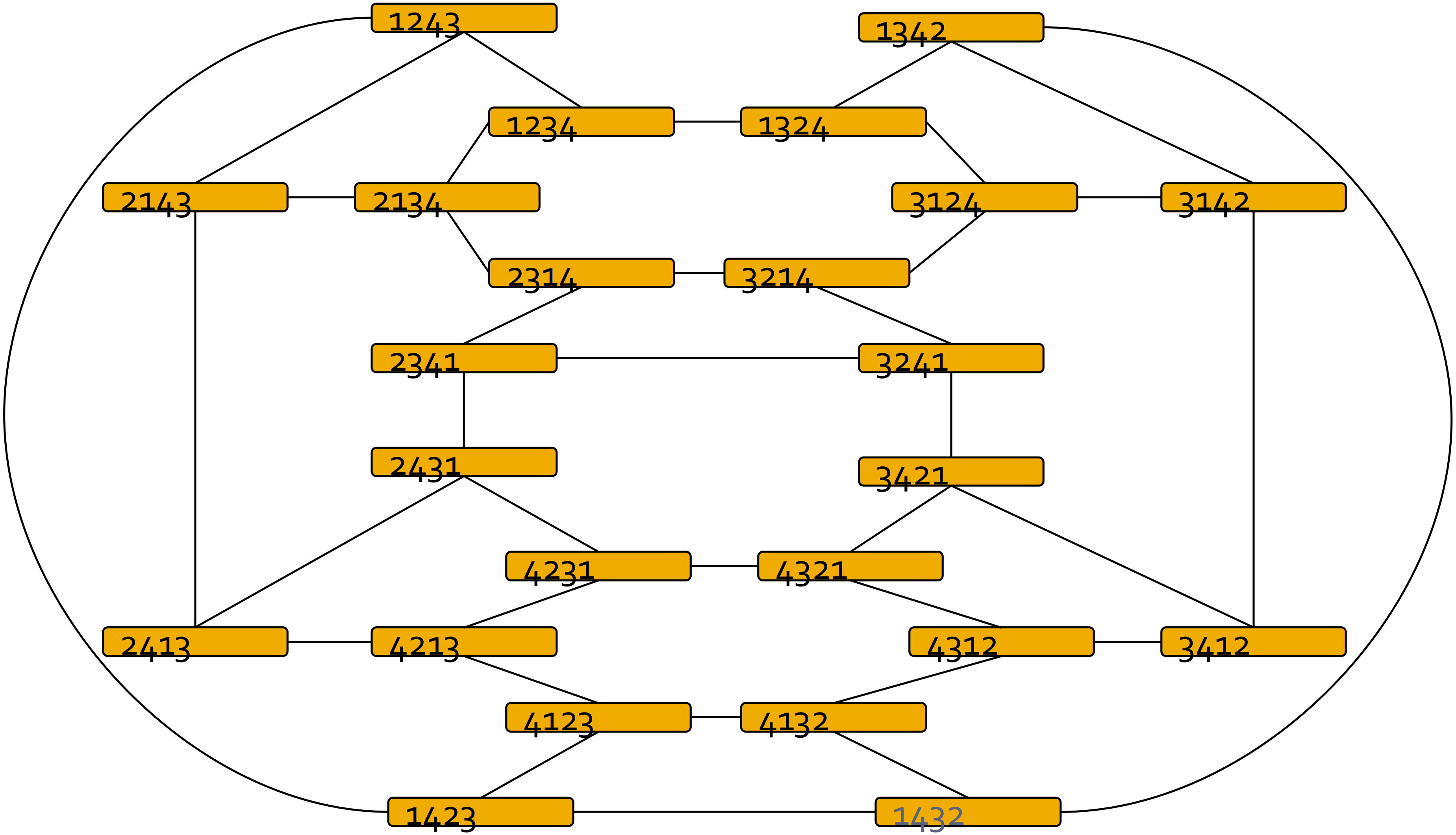
Algorithmes de recherche locale

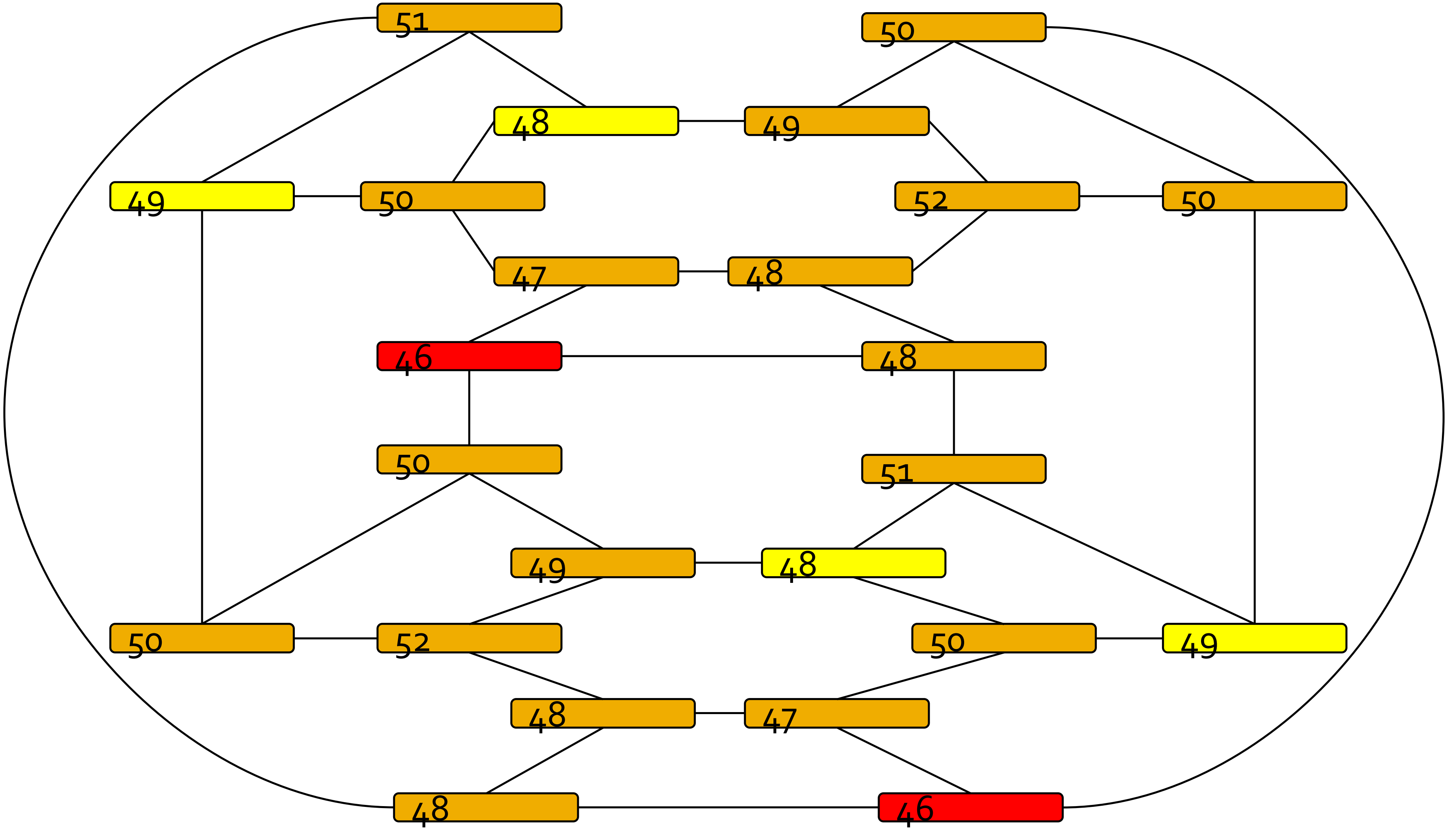
```
procedure RL-Perm-N1(π₀)
    π ← π₀; amélioration ← .vrai.
    while amélioration do
        amélioration ← .faux.; f_min ← +∞
        for i = 1 to n-1 do
            π′ ← π;  π′_i ← π_{i+1}; π′_{i+1} ← π_i;
            if  f(π′) < f_min then
                π_min ← π′ ; f_min ← f(π′)
            end-if
        end-for
        if  f_min < f(π)  then
            π ← π_min  ; amélioration ← .vrai.
        end-if
    end-while
    π⁺ ← π
    return  π⁺
end RL-Perm-N1
```


21
23
20
21
22
21
23
25
23
24
26
27
27
29
29
30
25
27
29
28
26
27
23
24

Voyageur de commerce symétrique

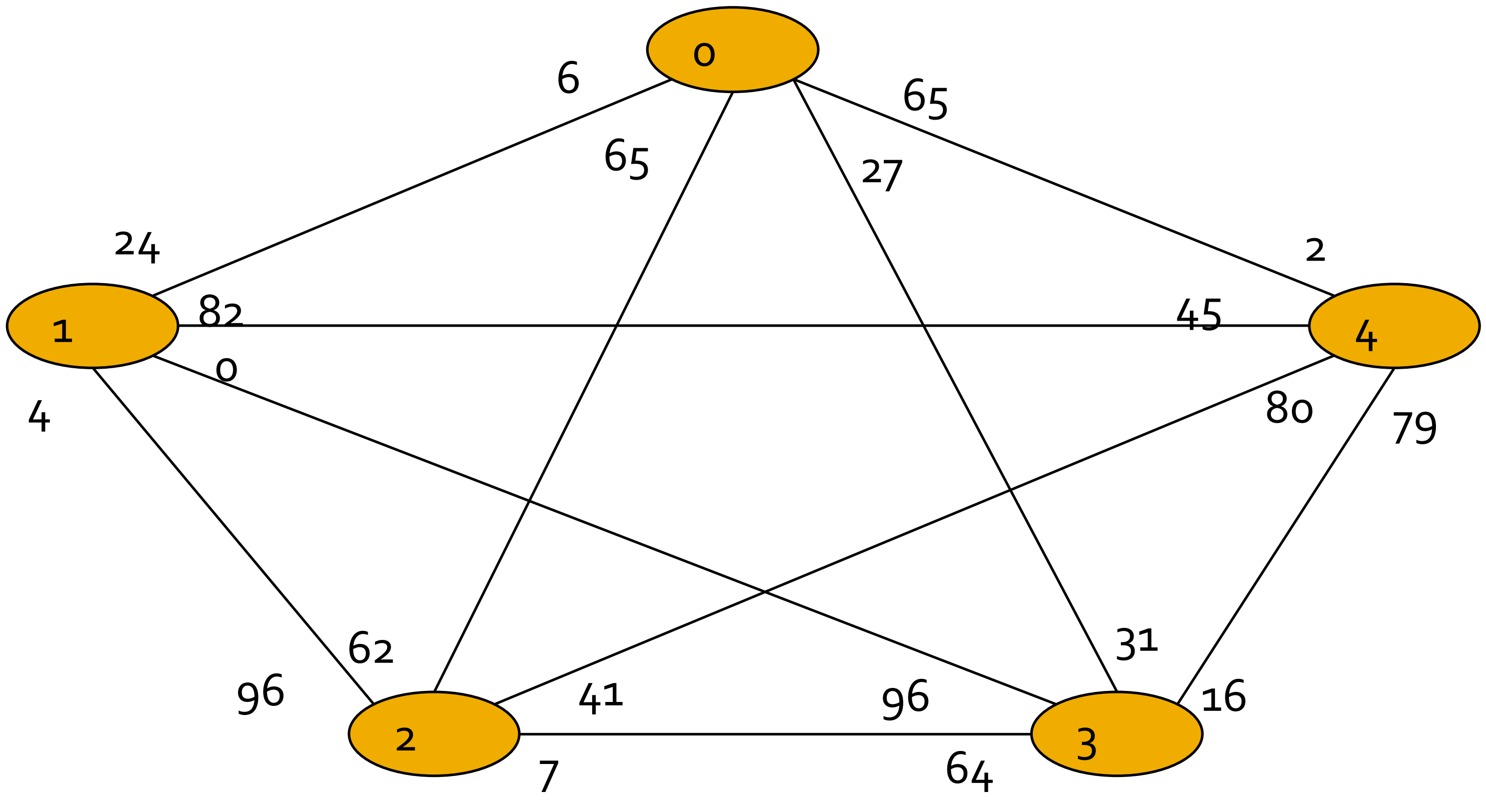


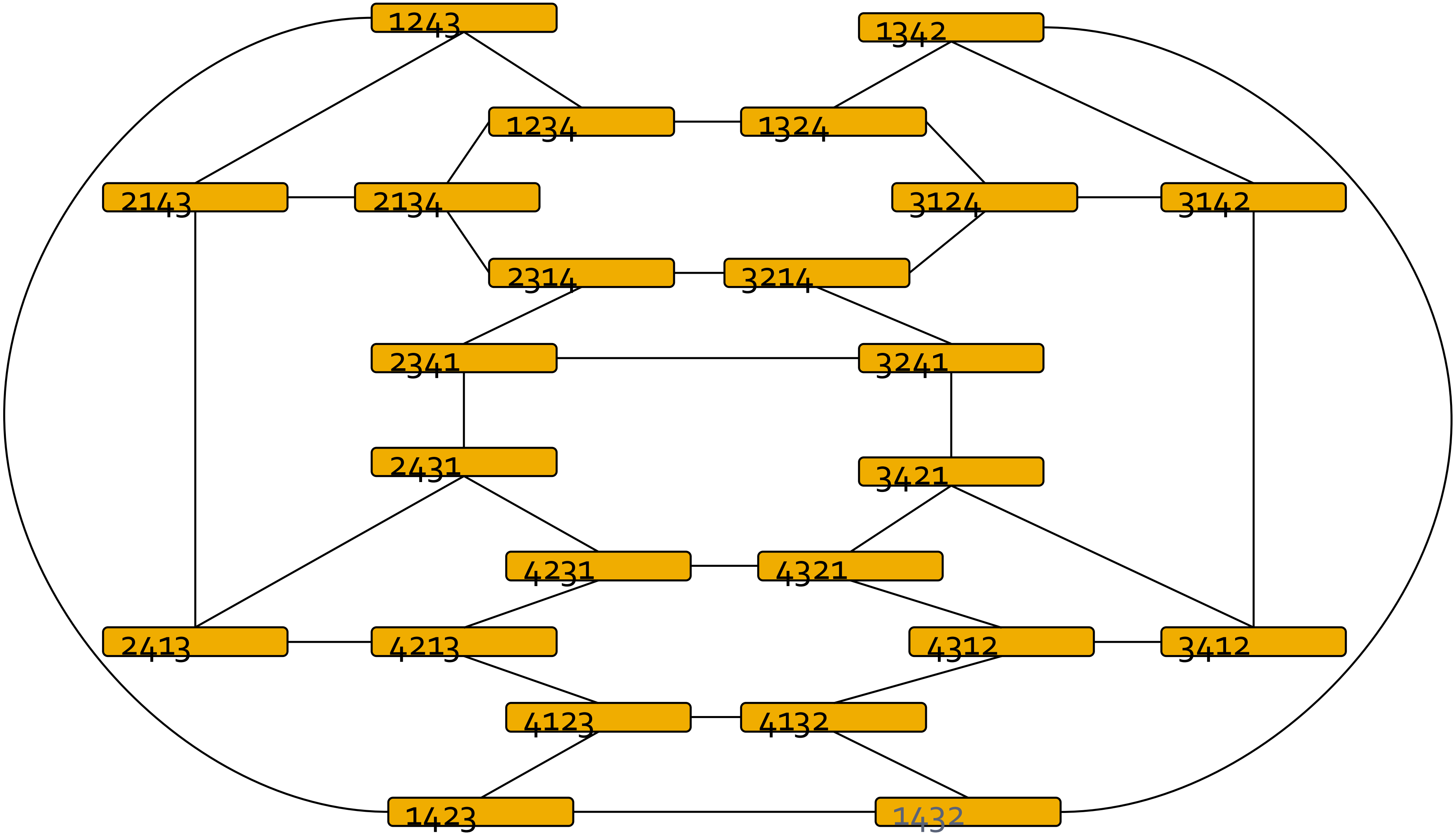

1243
1342
1234
1324
2143
2134
3124
3142
2314
3214
2341
3241
2431
3421
4231
4321
2413
4213
4312
3412
4123
4132
1423
1432

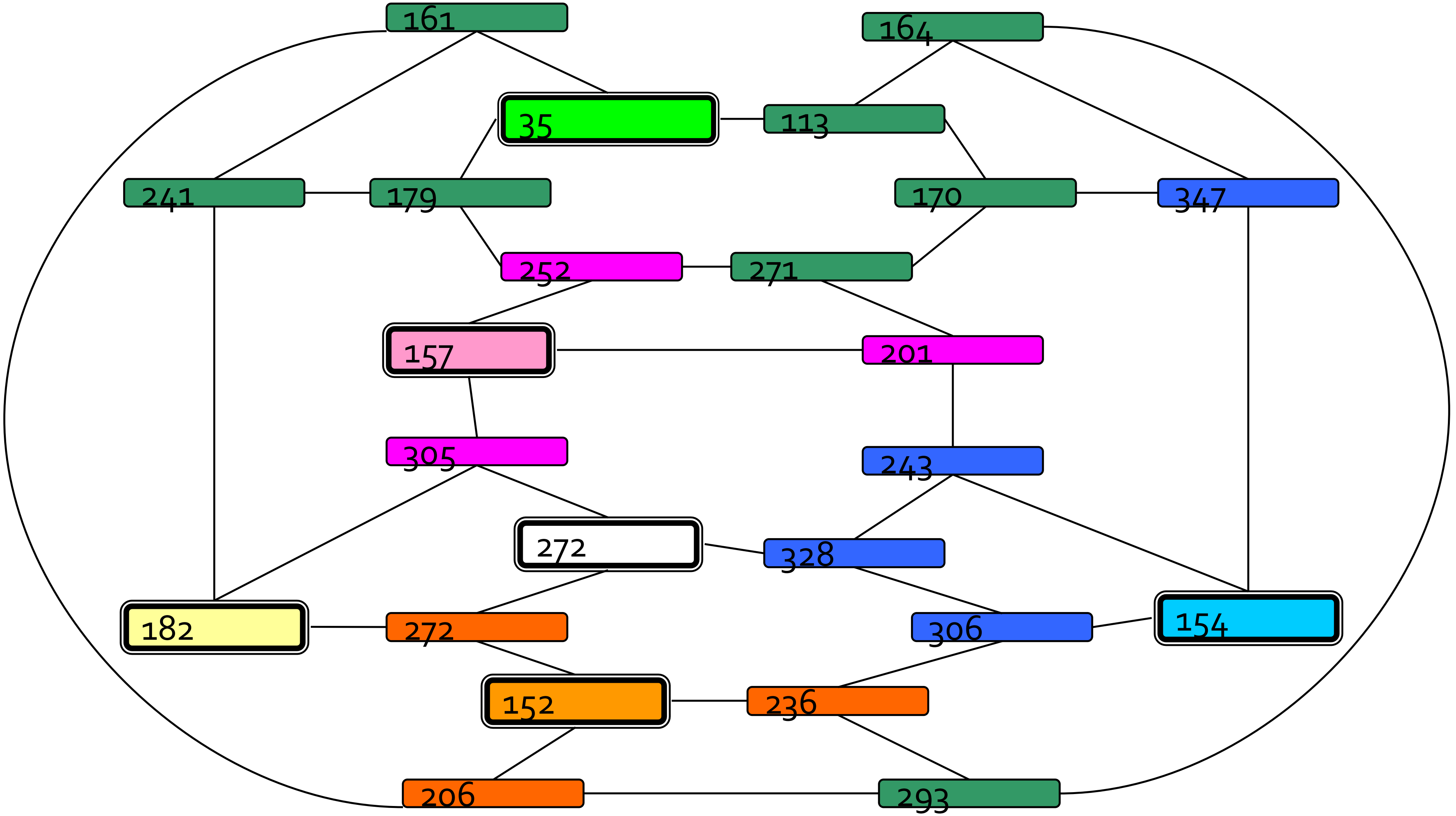


FIGL1AB

Voyageur de commerce asymétrique




1243
1342
1234
1324
2143
2134
3124
3142
2314
3214
2341
3241
2431
3421
4231
4321
2413
4213
4312
3412
4123
4132
1423
1432


161
164
35
113
241
179
170
347
252
271
157
201
305
243
272
328
182
272
306
154
152
236
206
293

FIGL1AB

Algorithmes de recherche locale

- Différents aspects de l'espace de recherche peuvent influencer la performance d'un algorithme de recherche locale
- Connexité: il doit y avoir un chemin entre chaque paire de solutions de l'espace de recherche
- Distance entre deux solutions: nombre de sommets visités sur le chemin le plus court entre elles.
- Diamètre: distance entre les deux solutions les plus eloignées (diamètres réduits!)

Algorithmes de recherche locale

- Dificultés:
  - Arrêt prématuré sur le premier optimum local
  - Sensibles à la solution de départ
  - Sensibles au voisinage choisi
  - Sensible à la stratégie de recherche
  - Nombre d'itérations

Algorithmes de recherche locale

- Extensions pour réduire les difficultés des algorithmes de recherche locale:
  - Réduction des voisinages: considérer un sous-ensemble du voisinage (e.g. par randomisation)
  - Accepter parfois des solutions qui n'améliorent pas la solution courante, de façon à pouvoir sortir d'un optimum local.
  - Multi-départ: appliquer l'algorithme de recherche locale à partir de plusieurs solutions de départ
  - Multi-voisinages dynamiques: changer de définition de voisinage après avoir obtenu un optimum local (ex.: 2-opt, après 3-opt)

→ **Metaheuristiques**