Résolution de Problèmes Combinatoires

Corentin BAZAN, Louis GRAS, Simon ROSARD

Sommaire

1. Problème
2. Modele MILP
   2.1. Présentation
   2.2. Modelisation
   2.3. Features
   2.4. Résultats
3. Modele CP
   3.1. Présentation
   3.2. Modelisation
   3.3. Résultats
4. Conclusion

Problème

Modele MILP

Modele MILP, Présentation

- Mixed-Integer Linear Programming

- Programmation linéaire sur nombres entiers

- Différentes méthodes:
  - Simplexe
  - Branch & Bound / Branch & cut
  - …

Modele MILP, Modelisation

- Variables :
  - **x,** une liste de booléens représentant si l’on prend une pièce ou non
  - **pos,** une liste de tuples de taille 3 représentant les positions des pièces
  - **orientation**, une liste de taille 6 représentant les rotations des pièces

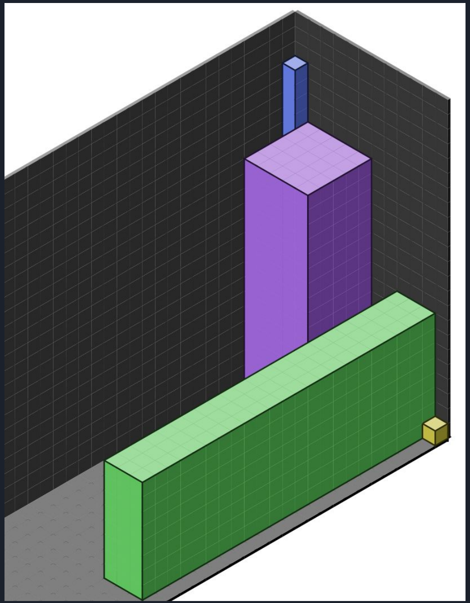
- Fonction objectif :
  - f (x, pos) = alpha * total_items + beta * total_volume

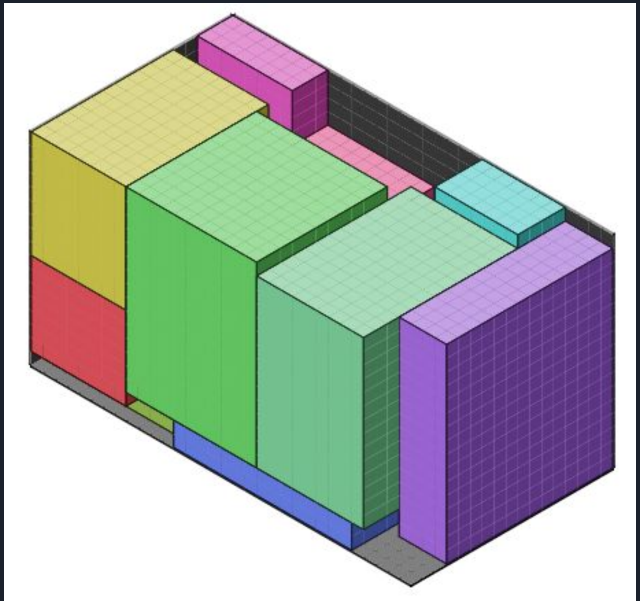
- Contraintes :
  - (p1, p2) dans le camion pour chaque item
  - (p1, p2) et (p3, p4) ne se chevauchent pas

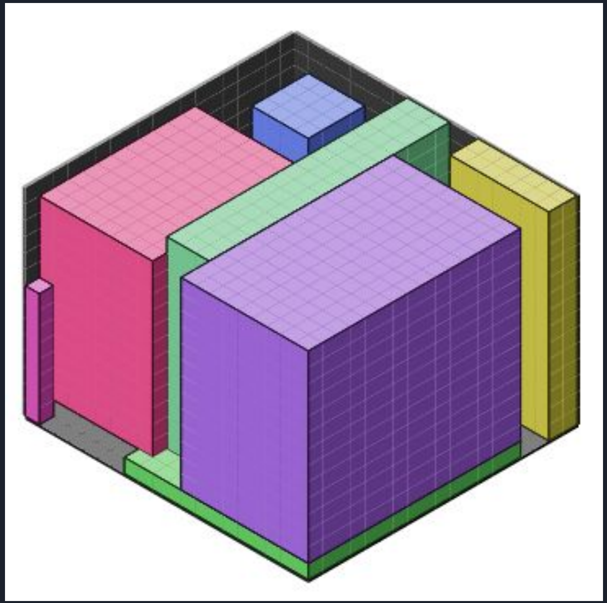
Modele MILP, Features

Model specifications	
✅ Can	❌ Can't
• Packages Position • Parallelepiped Rectangle Packages • Overlap • Truck Size • Maximize Packages amount (weight tunable) • Maximize Packages size (weight tunable) • Packages Rotation	• Maximize large empty space • Levitating packages • Multiple Trucks • Delivery order • Real physics with weight (small package can carry a huge package • Fragile packages • Not Parallelepiped Rectangle Packages

Modele MILP, Résultats







Modele CP

Modele CP, Présentation

- Constraint Programming

- Basé sur les relations et contraintes parmis les variables

- Différents type de recherche de solution:
    - Recherche arborescente
    - Propagation de contraintes

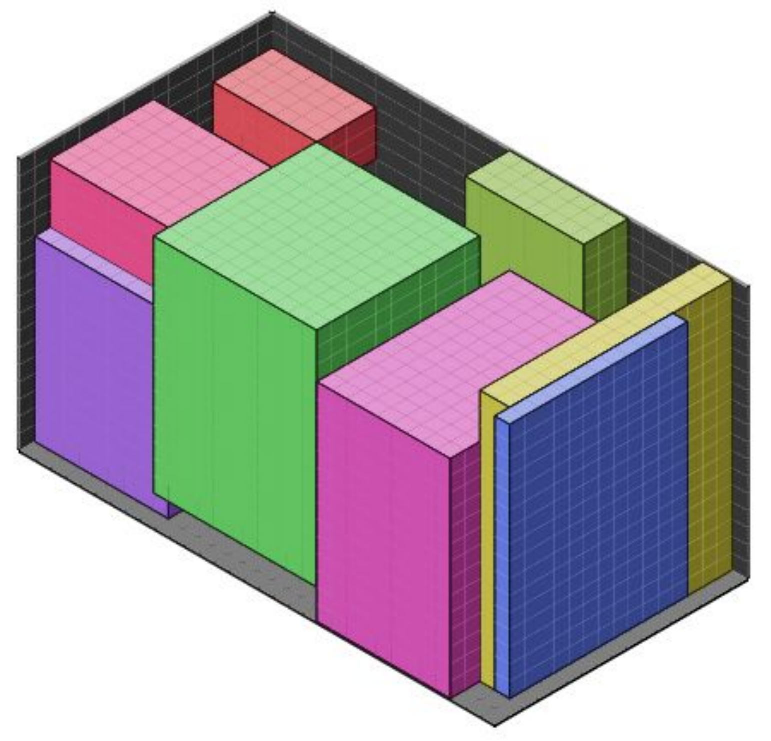
Modele CP, Modelisation

- Variables :
  - **x,** une liste de booléens représentant si l'on prend une pièce ou non
  - **pos,** une liste de tuples de taille 3 représentant les positions des pièces

- Fonction objectif :
  - f (x, pos) = total_items

- Contraintes :
  - (p1, p2) dans le camion pour chaque item
  - (p1, p2) et (p3, p4) ne se chevauchent pas

Modele CP, Résultats



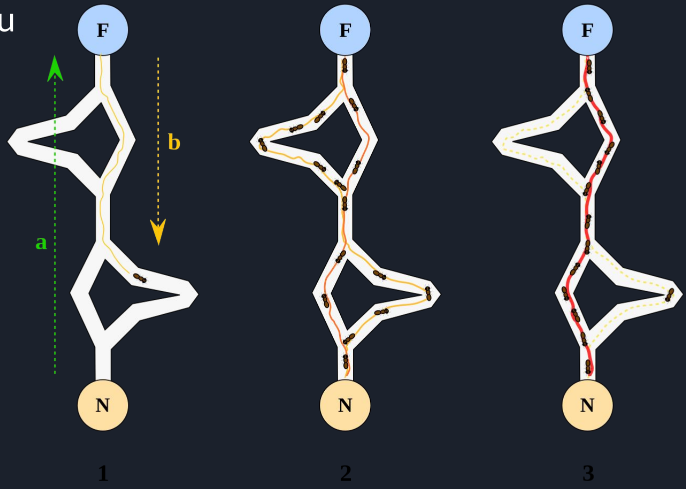
Conclusion

COLONIE DE FOURMIS

Simon ROSARD

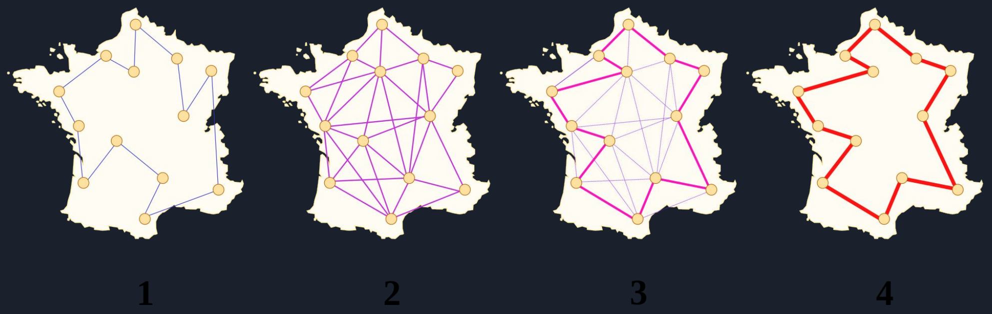
Principe

- une fourmi (appelée « éclaireuse ») parcourt plus ou moins au hasard l’environnement autour de la colonie ;
- si celle-ci découvre une source de nourriture, elle rentre plus ou moins directement au nid, en laissant sur son chemin une piste de phéromones
- ces phéromones étant attractives, les fourmis passant à proximité vont avoir tendance à suivre, de façon plus ou moins directe, cette piste
- en revenant au nid, ces mêmes fourmis vont renforcer la piste
- si deux pistes sont possibles pour atteindre la même source de nourriture, celle étant la plus courte sera, dans le même temps, parcourue par plus de fourmis que la longue piste
- la piste courte sera donc de plus en plus renforcée, et donc de plus en plus attractive
- la longue piste, elle, finira par disparaître, les phéromones étant volatiles
- à terme, l’ensemble des fourmis a donc déterminé et « choisi » la piste la plus courte.



DESCRIPTION GENERALE POUR LE PROBLEME TSP

- Une fourmi ne peut visiter qu’une fois chaque ville
- Plus une ville est loin, moins elle a de chance d’être choisie (c’est la « visibilité »)
- Plus l'intensité de la piste de phéromone déposée sur l’arête entre deux villes est grande, plus le trajet aura de chance d’être choisi
- Une fois son trajet terminé, la fourmi dépose, sur l’ensemble des arêtes parcourues, plus de phéromones si le trajet est court ;
- Les pistes de phéromones s’évaporent à chaque itération.



DESCRIPTION FORMELLE

La règle de déplacement, appelée « règle aléatoire de transition proportionnelle », est écrite mathématiquement sous la forme suivante :

(1) $$p_{ij}^{k}(t) = \begin{cases} \dfrac{\gamma + \tau_{ij}(t)^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in J_i^k} \left(\gamma + \tau_{il}(t)^{\alpha} \cdot \eta_{il}^{\beta} \right)} & \text{si } j \in J_i^k \\ 0 & \text{si } j \notin J_i^k \end{cases}$$

- J est la liste des déplacements possibles pour une fourmi k lorsqu'elle se trouve sur une ville i

- η_{ij} la visibilité, qui est égale à l'inverse de la distance de deux villes i et j ($1/d_{ij}$)

- $\tau_{ij}(t)$ l'intensité de la piste à une itération donnée t.

- α et β, qui contrôlent l'importance relative de l'intensité et de la visibilité d'une arête.

- une probabilité non nulle d'exploration de ces villes « inconnues », contrôlée par le paramètre γ.

DESCRIPTION FORMELLE

Une fois la tournée des villes effectuée, une fourmi k dépose une quantité de phéromone sur chaque arête de son parcours :

(2) $$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{Q}{L^k(t)} & \text{si } (i,j) \in T^k(t) \\ 0 & \text{si } (i,j) \notin T^k(t) \end{cases}$$

- Tk(t) est la tournée faite par la fourmi k à l'itération t
- Lk(t) la longueur du trajet
- Q un paramètre de réglage.

À la fin de chaque itération de l'algorithme, les phéromones déposées aux itérations précédentes par les fourmis s'évaporent. Ala fin de l'itération, on a la somme des phéromones qui ne se sont pas évaporées et de celles qui viennent d'être déposées :

(3) $$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t)$$

- m est le nombre de fourmis utilisées pour l'itération t
- ρ un paramètre de réglage.

ALGORITHME

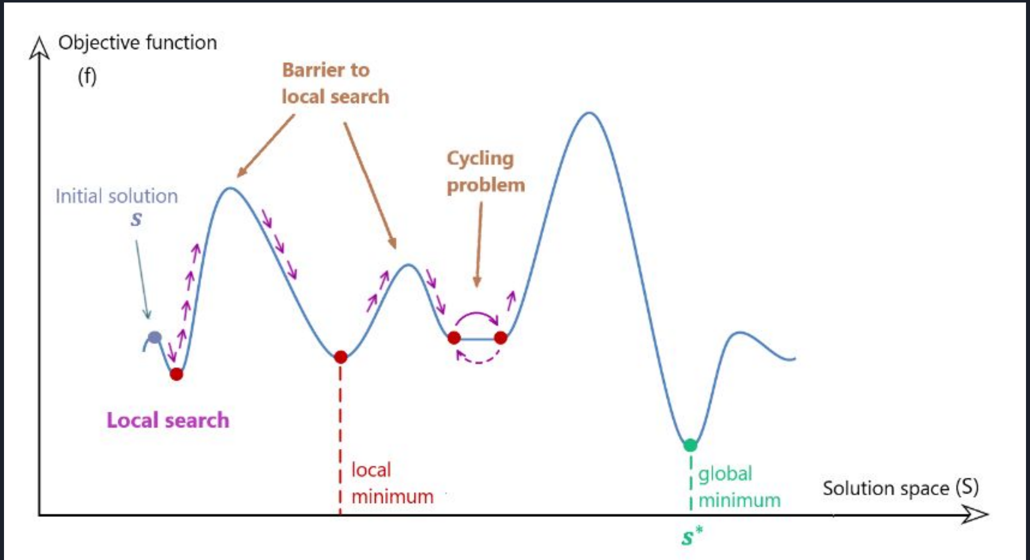
```
Tant que le critère d’arrêt n’est pas atteint faire
    Pour k = 1 à m faire
        Choisir une ville au hasard
        Pour chaque ville non visitée i faire
            Choisir une ville j parmi les villes restantes selon (1)
        Fin Pour
        Déposer une quantité de phéromones sur le trajet selon (2)
    Fin Pour
    Evaporer les pistes de phéromones selon (3)
Fin Tant que
```

Tabu Search

Corentin BAZAN

Principe

- Recherche locale
- Parcours de solutions voisines
- Liste tabou: solutions à ne plus visiter
- Donne la solution optimale du voisinage (pas globale)



Algorithme

- Fonction objectif: f

- MaxIter: nombre d’itération max

- Choix d’une solution
- Recherche de solutions voisines
- Choix de la meilleure solution voisine non-tabou
- Mise-à-jour de la liste tabou.

Algorithm 1: Tabu Search

Data: S - the search space, $maxIter$ - the maximal number of iterations, f - the objective function, the definition of neighborhoods, and the aspiration criteria.

Result: the best solution found

Choose the initial candidate solution $s \in S$

$s^* \leftarrow s$ `// Initialize the best-so-far solution.`

$k \leftarrow 1$

while $k \leq MaxIter$ **do**

  `/* Sample the allowed neighbors of` s `*/`

  Generate a sample $V(s,k)$ of the allowed solutions in $N(s,k)$

    `//` $s' \in V(s,k) \iff (s' \notin T) \vee (a(k,s') = true)$

  Set s to the best solution in $V(s,k)$

  `/* Update the best-so-far if necessary */`

  if $f(s) < f(s^*)$ **then**

    $s^* \leftarrow s$

  end

  Update T and a

  `/* Start another iteration */`

  $k \leftarrow k + 1$

end

Alternatives et améliorations

- Ajustement de la liste tabou
  - Politique de mise-à-jour
  - Structure
  - Memoire adaptative

- Structure du voisinage

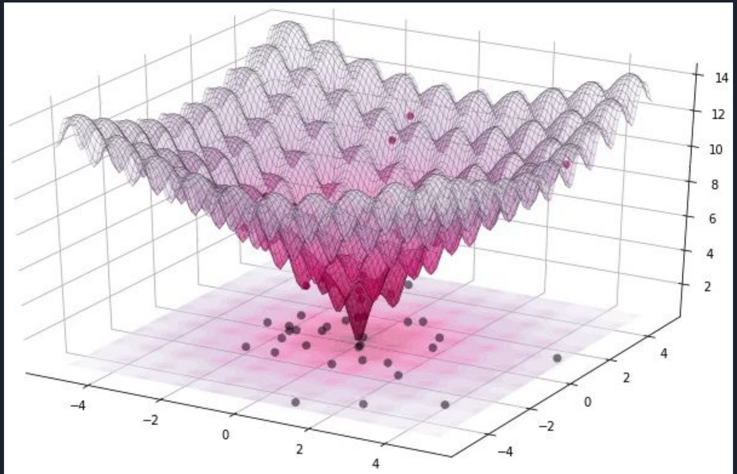
- Critère d’aspiration

- Utilisation hybride avec d’autre méthodes de recherche locales

ADE - Adaptive Differential Evolution

Louis GRAS

Principe



- Initialisation (Latin hypercube sampling)
- Mutation

$$V_{i,G+1} = X_{i,G} + F \cdot (X_{\text{best},G} - X_{i,G}) + F \cdot (X_{r1,G} - X_{r2,G}) + F \cdot (X_{r3,G} - X_{r4,G})$$

- CrossOver (with CL probability)
- Adaptation

$$F_{nouveau} = F_{ancien} + \alpha \cdot (F_{succès} - F_{ancien}) + rand(0,1) \cdot (F_{r1} - F_{r2})$$

- Répétition
- Point d’arrêt

Usability

+ Flexibilité & Robustesse

- Coût computationnel élevé et hyperparamètres délicats

- Aéronotique
- Génie civil
- Modélisation biologiques
- Finance

Alternatives et améliorations