

# Résolution du *Pigment Sequencing Problem* avec les algorithmes génétiques

**Présenté par:**

*Tafsir GNA*

**Supervisé par:**

*Dr Ing. Vinasétan Ratheil HOUNDJ*

*&*

*Professeur Mahouton Norbert HOUNKONNOU*

Institut de Formation et de Recherche en Informatique (IFRI)

17 septembre 2017

## 1 Etat de l'art

- Le dimensionnement de lots en planification de production
- Le "Pigment Sequencing Problem" (PSP)
- Les algorithmes génétiques

## 2 Matériel et Méthodes

- Outils de test
- Modèle et formulation utilisés
- Aspects généraux aux deux méthodes de recherche proposées
- Méthodes de recherche proposées

## 3 Résultats et discussion

- Données et paramètres de test
- Résultats expérimentaux des algorithmes génétiques parallèles hiérarchiques fine-grained et coarse-grained
- Résultats expérimentaux des algorithmes génétiques parallèles hiérarchiques master-slave et coarse-grained
- Discussion

## Introduction

# Le dimensionnement de lots en planification de production

## Critères de classification

- L'échelle de temps ;
- Le nombre de niveaux ;
- Le nombre de produit ;
- Les contraintes de capacité ;
- Les demandes ;
- Les coûts et temps de lancement ou préparation (setup).

# Le dimensionnement de lots en planification de production

## Classes de problèmes de dimensionnement de lots I

- Problèmes de petite taille ou à courtes périodes ;
- Problèmes de grande taille ou à longues périodes ;
- Problèmes de très grande taille ou à très longues périodes.

# Le dimensionnement de lots en planification de production

## Classes de problèmes de dimensionnement de lots II

- Discrete lot-sizing and scheduling problem (DLSP);
- Continuous setup lot-sizing problem (CSLP);
- Proportional lot-sizing and scheduling problem (PLSP);
- General lot-sizing and scheduling problem (GLSP).

# Le dimensionnement de lots en planification de production

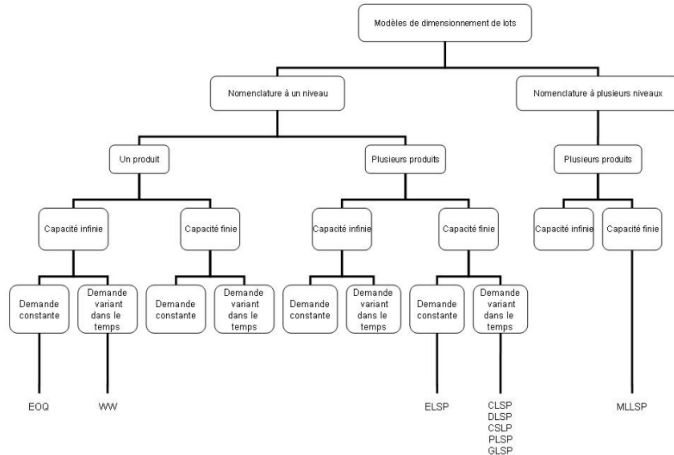


Figure – Exemple de classification des modèles de dimensionnement de lots []

# Le "Pigment Sequencing Problem" (PSP)

## Description

Objectif :

- trouver un plan de production de plusieurs articles à partir d'une machine avec des coûts de transition ;
- respectant la capacité de production de la machine ;
- minimisant les coûts de stockage et de transition.



# Le "Pigment Sequencing Problem" (PSP)

## Modèles et formulations

- Modèles basés sur la programmation en nombres mixtes[] (MIP1, MIP2, MIP3) ;
- Modèle basé sur la Programmation par contraintes[] ;
- Modèle basé sur le recuit simulé[].

# Les algorithmes génétiques

## Algorithme génétique standard

Générer la population initiale  $P_i$

Évaluer la population  $P_i$

**while** *le critère de terminaison n'est pas satisfait* **do**

    Sélectionner les éléments de  $P_i$  à copier dans  $P_{i+1}$

    Appliquer le croisement aux éléments de  $P_i$  et les mettre dans  $P_{i+1}$

    Appliquer la mutation aux éléments de  $P_i$  et les mettre dans  $P_{i+1}$

    Évaluer la nouvelle population  $P_{i+1}$

$P_i = P_{i+1}$

**end**

**Algorithm 1:** Algorithme génétique standard [?]

# Les algorithmes génétiques

## Opérations génétiques

- la sélection ;
- le croisement ;
- la mutation.

# Les algorithmes génétiques

## Les algorithmes génétiques parallèles

- Les algorithmes génétiques parallèles de type master-slave ;
- Les algorithmes génétiques parallèles coarse-grained ;
- Les algorithmes génétiques parallèles fine-grained.

# Les algorithmes génétiques

## Hierarchisation entre les algorithmes génétiques parallèles

- Les algorithmes génétiques parallèles et hiérarchiques entre coarse-grained et master-slave ;
- Les algorithmes génétiques parallèles et hiérarchiques entre coarse-grained et fine-grained.

- Système d'exploitation : Linux Ubuntu 16.04 LTS ;
- Processeur : Intel® Core™ i7 CPU L 640 @ 2.13GHz x 4 ;
- Mémoire : 3,7 Gio ;
- Type du système d'exploitation : 64 bits.

# Modèle et formulation utilisés I

## MIP1

$$\min \sum_{i,j,t} q^{i,j} \chi_t^{i,j} + \sum_{i,t} h^i s_t^i \quad (1)$$

$$s_0^i = 0, \forall i \quad (2)$$

$$x_t^i + s_{t-1}^i = d_t^i + s_t^i, \forall i, t \quad (3)$$

$$x_t^i \leq y_t^i, \forall i, t \quad (4)$$

$$\sum_i y_t^i = 1, \forall t \quad (5)$$

$$\chi_t^{i,j} = y_{t-1}^i + y_t^j - 1, \forall i, j, t \quad (6)$$

$$x, y, \chi \in \{0, 1\}, s \in \mathbb{N}, i \in \{0..NI\}, t \in \{1..NT\} \quad (7)$$

avec les variables de décisions suivantes :

# Modèle et formulation utilisés II

## MIP1

- $x_t^i$  : variable binaire de production qui vaut 1 si l'article  $i$  est produit à la période  $t$  et 0 sinon ;
- $y_t^i$  : variable binaire de setup qui vaut 1 si la machine est préparée pour la production de l'article  $i$  et 0 sinon ;
- $s_t^i$  : variable entière de stockage qui contient le nombre d'articles  $i$  stockés à la période  $t$  ;
- $\chi_t^{i,j}$  : variable binaire de transition qui vaut 1 si à la période  $t$ , on est passé de la production de l'article  $i$  à l'article  $j$  et 0 sinon.



# Aspects généraux aux deux méthodes de recherche proposées

## Représentation génétique

$$ch_{Tn} = \{(I_{T1}), \dots, (I_{T2}), \dots, (I_{T3}), (I_{T4}), \dots, (I_{T(n-1)}), \dots, (I_{Tn})\}$$

où  $ch_{Tn}$  est un chromosome dont l'horizon de planification est de  $Tn$  périodes et  $I_{Ti}$  est la variable entière qui indique l'article produit à la période  $Ti$ .

**Illustration** avec les demandes  $D_{I1} = (0, 1, 0, 0, 1)$  et  $D_{I2} = (1, 0, 0, 0, 1)$

T1	T2	T3	T4	T5
2	1	0	2	1

T1, T2, T3, T4, T5 : Période 1, 2, 3, 4, 5

Figure – Représentation génétique adoptée

# Aspects généraux aux deux méthodes de recherche proposées

## Initialisation I

---

**Algorithme** : Processus de génération de la population initiale

---

**Données** : instance de PSP à traiter, taille de la population

**Résultat** : Population initiale constituée

*queue*  $\leftarrow \square$

*noeud*  $\leftarrow$  *nouveauNoeud()*

**tant que** *taille(populationInitiale)* est inférieure à *taillePopulation* **faire**

**si** *noeud.chromosome* est prêt **alors**

*populationInitiale.ajouter*(*noeud.chromosome*)

**sinon**

*noeudFils*  $\leftarrow$  *noeud.obtenirSuccesseurs()*

*noeudFils.trier*(décroissant)

*queue.ajouter*(*noeudFils*)

**si** *queue* est vide **alors**

**retourner** *populationInitiale*

**fin**

*noeud*  $\leftarrow$  *queue.dernier()*

**fin**

**fin**

**retourner** *populationInitiale*

---

# Aspects généraux aux deux méthodes de recherche proposées

## Initialisation II

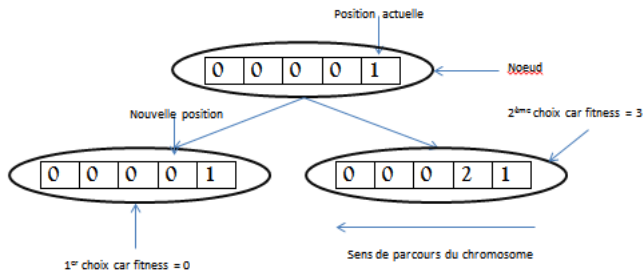


Figure – Schéma illustratif de l'application de l'algorithme du Hill climbing à une instance de PSP

# Aspects généraux aux deux méthodes de recherche proposées

## Opérateurs génétiques I

### Sélection

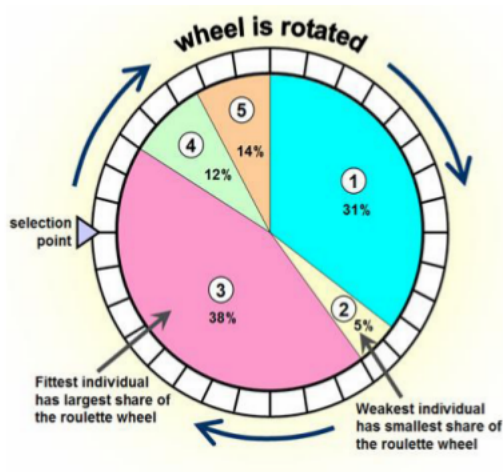


Figure – Schéma illustratif de la méthode de *roulette wheel*

### Croisement

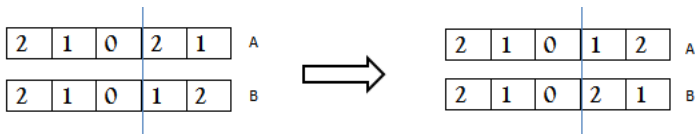


Figure – Schéma illustratif de la méthode de croisement utilisée

### Mutation



Figure – Schéma illustratif de la méthode de mutation utilisée

# Aspects généraux aux deux méthodes de recherche proposées

## Evaluation

---

**Algorithme** : Algorithme utilisé dans le processus d'évaluation d'un chromosome

---

**Données** : chromosome, cout\_stockage, cout\_transition

**Résultat** : eval

*eval*  $\leftarrow$  0

*//On calcule le cout de stockage de chaque production*

**pour** *gene in chromosome* **faire**

**si** *gene.value*  $\neq$  0 **alors**

*date\_limite*  $\leftarrow$  *getDateLimite(gene)*

*temp*  $\leftarrow$  (*date\_limite* - *gene.periode*) \* *cout\_stockage(gene.value)*

*evaluation*  $\leftarrow$  *evaluation* + *temp*

**fin**

**fin**

*//On calcule le cout de transition de entre deux productions*

**pour** *gene in chromosome* **faire**

**si** *gene.valeur*  $\neq$  0 **alors**

*next\_gene*  $\leftarrow$  *obtenirProchainGene(chromosome)*

**si** *transition(gene, prochain\_gene)* est vrai **alors**

*temp*  $\leftarrow$  *cout\_transition(gene, prochain\_gene)*

*evaluation*  $\leftarrow$  *evaluation* + *temp*

**fin**

**fin**

**fin**

**retourner** *evaluation*

---

### Deux critères de terminaison

- Convergence de la population sur une solution ;
- Absence de meilleures solutions à partir de celle trouvée.



# Aspects généraux aux deux méthodes de recherche proposées

## Fonction de faisabilité

---

**Algorithme** : Algorithme utilisé comme fonction de faisabilité

---

**Données** : chromosome, deadlines

*// On commence par reduire le surplus de production ;*

**pour**  $i \leftarrow 1$  à *Nombre\_Periodes* **faire**

*article*  $\leftarrow$  chromosome.obtenirArticle(*i*) ;

**si** *estEnSurplus(article, deadline(i))* **alors**

        supprimerProduction(*article*) ;

**fin**

**fin**

*// On compense le manque de production ;*

**pour** *article* in *liste\_articles* **faire**

*article\_deadlines*  $\leftarrow$  deadlines(*article*) ;

**pour** *deadline* in *article\_deadlines* **faire**

**si** *nonProduit(dealine)* **alors**

            produire(*article*) ;

**fin**

**fin**

**fin**

---

# Méthodes de recherche proposées

Algorithmes génétiques parallèles et hiérarchiques coarse-grained et master-slave

Problématiques :

- Fréquence de migration ;
- Choix et nombre de migrants ;
- Topologie de connexions ;
- Méthode d'intégration des migrants ;

# Méthodes de recherche proposées

Algorithmes génétiques parallèles et hiérarchiques coarse-grained et fine-grained

Problématiques :

- Fréquence de migration ;
- Choix et nombre de migrants ;
- Topologie de connexions ;
- Méthode d'intégration des migrants ;

# Méthodes de recherche proposées

## Autres algorithmes implémentés I

- Hybridation

---

**Algorithme** : Algorithme utilisé comme fonction de faisabilité

---

**Données** : chromosome, deadlines

*// On commence par reduire le surplus de production ;*

**pour**  $i \leftarrow 1$  à *Nombre\_Periodes* **faire**

*article*  $\leftarrow$  *chromosome.obtenirArticle(i)* ;

**si** *estEnSurplus(article, deadline(i))* **alors**

*supprimerProduction(article)* ;

**fin**

**fin**

*// On compense le manque de production ;*

**pour** *article* in *liste\_articles* **faire**

*article\_deadlines*  $\leftarrow$  *deadlines(article)* ;

**pour** *deadline* in *article\_deadlines* **faire**

**si** *nonProduit(dealine)* **alors**

*produire(article)*;

**fin**

**fin**

**fin**

---

- Table de hash

---

**Algorithme** : Algorithme implémentant l'utilisation de la table de hash dans nos méthodes proposées

---

**Données** : chromosome, table\_hash, max\_len

---

solution  $\leftarrow$  chromosome.solution;

**si** *solution in table\_hash* **alors**

    indice  $\leftarrow$  table\_hash[solution] ;

    chromosome.valeurFitness  $\leftarrow$  table\_hash[solution][valeurFitness] ;

**sinon**

    chromosome.valeurFitness  $\leftarrow$  evaluation(solution) ;

    table\_hash.ajouter([solution, chromosome.valeurFitness]);

**si** longueur(table\_hash) > max\_len **alors**

        table\_hash.enleverDernier() ;

**fin**

**fin**

---

# Résultats et discussion

# Conclusion

Merci pour votre attention