Compte rendu – TP Ordonnancement de Tâches

Membres du groupe

- Samuel DAMESSI
- Timoty RAZAFINDRABE
- Ayman SALOUH

Modélisation

Conventions de notations

Symbole	Description
$\mathcal{J}$	Ensemble des **jobs** (ou tâches), indexés par j
$\mathcal{M}$	Ensemble des **machines**, indexées par m
$\mathcal{O}$	Ensemble des **opérations** indexées par o
$\text{compat}_{j,o}$	Ensemble des **machines compatibles** avec l'opération o du job j
$d_{j,o,m}$	**Durée** d'exécution de l'opération o du job j sur la machine m
$e_{j,o,m}$	**Énergie** consommée pour exécuter l'operation o du job j sur la machine m
e_m^{start}	**Énergie** pour **allumer** la machine m
e_m^{stop}	**Énergie** pour **éteindre** la machine m
t_m^{start}	**Durée** pour **allumer** la machine m
t_m^{stop}	**Durée** pour **éteindre** la machine m

Symbole	Description
e_m^{idle}	**Énergie** par unité de temps pour une machine m **inactive mais allumée**
$T_m^{\max}$	**Instant limite** au-delà duquel la machine m **doit être arrêtée**
$x_{j,o,m}$	Variable binaire : 1 si o du job j est affectée à m, 0 sinon
$s_{j,o,m}$	Instant de **début** de l'opération o du job j sur la machine m
$a_m^k,\ b_m^k$	Début et fin de la k-ième **plage d'activité** de la machine m
$C_{\max}$	**Durée totale** du planning
E	Consommation énergétique totale
α, β	Coefficients de pondération
λ	Coefficient de pénalisation pour contraintes violées

---

1. Variables de décision, contraintes et objectifs

Variables de décision

- $x_{j,o,m} \in \{0, 1\}$: 1 si l'opération o du job j est affectée à la machine m, 0 sinon.
- $s_{j,o,m} \in \mathbb{R}^+$: instant de début de l'opération o du job j sur la machine m.
- $a_m^k, b_m^k \in \mathbb{R}^+$: début et fin de la k-ième plage d'activité de la machine m avec $a_m^k \leq b_m^k$.

Contraintes 1. Affectation unique

Chaque opération doit être affectée à **exactement une machine** parmi celles compatibles avec elle.

$$\sum_{m \in \text{Compat}(j,o)} x_{j,o,m} = 1$$

$$j \in \mathcal{J}, o \in \mathcal{O}, m \in \mathcal{M}$$

2. Compatibilité machine/opération

Une opération **ne peut être affectée** qu'à une machine compatible.

$$x_{j,o,m} = 0 \quad \text{si } m \notin \text{compat}_{j,o}$$
$$j \in \mathcal{J}, o \in \mathcal{O}, m \in \mathcal{M}$$

3. Précédence des opérations au sein d'une tâche

Les opérations d'une même tâche doivent être exécutées **dans l'ordre**. Une opération ne peut commencer qu'après la fin de la précédente.

$$s_{j,o+1,m} \geq s_{j,o,m} + \sum_{m \in \text{Compat}(j,o)} x_{j,o,m} \cdot d_{j,o,m}$$
$$j \in \mathcal{J}, \quad o \in \mathcal{O}, \quad m \in \mathcal{M}$$

4. Non-chevauchement sur une même machine

Deux opérations affectées à la **même machine** ne peuvent pas être exécutées en même temps. Pour toute paire d'opérations affectées à une machine donnée, l'une doit terminer avant que l'autre ne commence.

$$\text{Si } x_{j,o,m} = x_{j',o',m} = 1$$
$$s_{j,o,m} + d_{j,o,m} \leq s_{j',o',m} \quad \text{ou} \quad s_{j',o',m} + d_{j',o',m} \leq s_{j,o,m}$$
$$j \in \mathcal{J}, o \in \mathcal{O}, m \in \mathcal{M}$$

5. Respect des plages actives des machines

Une opération ne peut être exécutée que **durant une période où la machine est active** (entre l'allumage complet et l'extinction).

Pour toute opération affectée à une machine, son intervalle d'exécution `[début, fin]` doit être entièrement contenu dans une plage active de cette machine.

$$s_{j,o} \in \bigcup_k [a_m^k + t_m^{\text{start}},\ b_m^k] \quad \text{et} \quad s_{j,o} + d_{j,o,m} \in \bigcup_k [a_m^k + t_m^{\text{start}},\ b_m^k] \quad \text{si } x_{j,o,m} = 1$$

$$j \in \mathcal{J}, o \in \mathcal{O}, m \in \mathcal{M}$$

6. Limite maximale des plages d'activité machine

Le planning d'utilisation d'une machine ne peut pas dépasser une **durée maximale autorisée** définie par l'entreprise.

Pour toute période active k d'une machine m, on impose :

$$a_m^k \leq b_m^k \leq T_m^{\max} \quad \forall m \in \mathcal{M}, \forall k$$

Objectifs

- **Minimisation de la consommation d'énergie totale** (incluant allumage, fonctionnement actif, inactivité, extinction).
- **Minimisation de la durée totale du planning**.

2. Fonction objectif agrégée

Une fonction multicritère pondérée peut être proposée :

$$\min \left(\alpha \cdot E + \beta \cdot C_{\max} \right)$$

avec :

- E : énergie totale consommée
- $C_{\max}$: durée totale du planning
- α, β : coefficients de pondération.

3. Évaluation d'une solution

Solution réalisable Une solution est **réalisable** si toutes les contraintes sont respectées.

Son évaluation repose sur :

- E : somme des énergies d'allumage, de traitement, d'inactivité et d'extinction des machines utilisées
- $C_{\max}$: dernier instant d'activité de toutes les machines,

Solution non réalisable Une solution non réalisable est évaluée à l'aide d'une fonction pénalisée :

$$f_{\text{pénalisée}} = f + \lambda \cdot \text{Violations}$$

où :

- f est la valeur de la fonction objectif originale
- λ est un coefficient de pénalité élevé representant le cout d'une violation.
- Violations mesure l'ampleur des contraintes violées

4. Instance sans solution réalisable

Exemple

- 1 job j_1 avec 2 opérations o_0, o_1 ;
- 1 seule machine m_1 compatible ;
- Durées : d_{j_1,o_0,m_1} = 60, d_{j_1,o_1,m_1} = 70 ;
- Limite de disponibilité : $T_{m_1}^{\text{max}} = 100$.

Analyse La somme des durées est $60 + 70 = 130$,ce qui dépasse la limite de temps autorisée.

Donc **aucune solution réalisable n'existe** dans cette instance.

Premières heuristiques

Afin d'obtenir une solution initiale au problème d'ordonnancement, nous proposons deux heuristiques : une déterministe gloutonne et une non-déterministe. Ces heuristiques visent à générer rapidement des plannings raisonnables, qui peuvent ensuite être améliorés via des méthodes de recherche locale.

1. Heuristique gloutonne déterministe

Principe général L'algorithme construit la solution opération par opération, en attribuant à chaque opération la machine admissible minimisant une **fonction de coût** combinant la durée d'exécution et la consommation d'énergie. Le choix est fait de manière *myope*, sans retour arrière, ce qui caractérise une approche gloutonne.

La fonction de coût utilisée est :

$Coût(o, j, m) = (\alpha \times e_{j,o,m}) + (\beta \times d_{j,o,m})$

où :

- $e_{j,o,m}$ est l'énergie consommée pour exécuter l'operation o du job j sur la machine m
- $d_{j,o,m}$ est la durée d'exécution de l'opération o du job j sur la machine m
- α, β sont des coefficients positifs permettant de pondérer l'importance de la durée et de l'énergie.

Optimisation de la consommation passive Lorsqu'une machine reste inactive entre deux opérations, l'algorithme évalue s'il est préférable : - de la laisser allumée (consommation passive),

- ou de l'éteindre temporairement et de la rallumer juste à temps.

Ce choix dépend :

- du **coût énergétique d'allumage/extinction**,
- du **temps nécessaire pour rallumer** la machine,
- de la **durée d'inactivité**.

Cela permet de limiter la consommation énergétique dans les phases creuses, en intégrant une décision locale sur l'état de la machine. Il faut noter que n'ayant pas une vision complete ce heuristique ne produit pas la meilleure solution et peut parfois produire des solution non réalisable

Complexite Soit :

- $|\mathcal{O}|$ le nombre total d'opérations
- $|\mathcal{M}|$ le nombre de machines.

Pour chaque opération, l'algorithme évalue le coût sur toutes les machines admissibles. La complexité est donc :

La complexité est donc : $O(|\mathcal{O}| \times |\mathcal{M}|)$

2. Heuristique non-déterministe

Principe général L'algorithme construit la solution opération par opération, en attribuant à chaque opération une machine admissible choisie **aléatoirement**. Aucune évaluation de coût n'est réalisée lors de cette sélection.

Le choix est uniforme parmi les machines admissibles à l'opération, ce qui permet de générer des solutions différentes à chaque exécution, sans guidage par un critère d'optimisation.

Optimisation de la consommation passive Lorsqu'une machine reste inactive entre deux opérations, l'algorithme évalue s'il est préférable :

- de la laisser allumée (consommation passive),
- ou de l'éteindre temporairement et de la rallumer juste à temps.

Ce choix dépend :

- du **coût énergétique d'allumage/extinction**,
- du **temps nécessaire pour rallumer** la machine,
- de la **durée d'inactivité**.

Cela permet de limiter la consommation énergétique dans les phases creuses, en intégrant une décision locale sur l'état de la machine.

Complexite Soit :

- $|\mathcal{O}|$ le nombre total d'opérations

Aucune recherche parmi les machines n'est effectuée. Chaque opération se voit attribuer une machine en temps constant (tirage aléatoire), et l'optimisation de consommation est locale.

$\mathcal{O}(|\mathcal{O}|)$

Comme l'heuristique gloutone certaines solutions produitent par cette heuristique ne sont pas forcement realisable ## Recherche locale

1. Proposition de deux voisinages de solutions

Pour améliorer les solutions initiales obtenues par les heuristiques, nous proposons deux types de voisinages permettant d'explorer l'espace des solutions voisines.

Voisinage 1 : Réaffectation d'une opération

- **Principe** : On choisit une opération et on modifie la machine sur laquelle elle est exécutée, en la déplaçant vers une autre machine admissible.
- **Taille du voisinage** : Environ $|\mathcal{O}| \times (|\mathcal{M}| - 1)$, où $|\mathcal{O}|$ est le nombre total d'opérations et $|\mathcal{M}|$ le nombre de machines.
- **Complexité** : Taille polynomiale par rapport à la taille de l'instance.

- **Couverture de l'espace des solutions** : Ce voisinage permet de modifier progressivement l'affectation des opérations, mais ne suffit pas à atteindre toutes les permutations possibles, notamment les réordonnancements.

Voisinage 2 : Échange d'opérations entre machines

- **Principe** : On choisit deux opérations sur deux machines différentes et on échange leurs affectations (machines et éventuellement leurs positions temporelles).
- **Taille du voisinage** : Environ $O(|\mathcal{O}|^2)$ où $|\mathcal{O}|$ est le nombre total d'opérations, car on considère toutes les paires d'opérations.
- **Complexité** : Taille polynomiale.
- **Couverture de l'espace des solutions** : Ce voisinage est plus riche et permet de parcourir un plus large espace de solutions, incluant des permutations complexes, mais ne garantit pas l'accessibilité de toutes les solutions.

2. Implémentation des voisinages

Les deux voisinages seront implémentés dans le module `optim.neighborhoods` sous forme de générateurs de solutions voisines à partir d'une solution courante.

3. Implémentation des algorithmes de recherche locale

Deux algorithmes de recherche locale seront développés dans le module `optim.local_search`, utilisant la classe `NonDeterminist` pour générer la solution initiale.

- **Algorithme 1** :
  Utilise uniquement le premier voisinage.
  À chaque itération, explore le voisinage et accepte la **première solution améliorante** rencontrée.
- **Algorithme 2** :
  Utilise les deux voisinages.
  À chaque itération, explore successivement chaque voisinage et sélectionne la **meilleure solution améliorante** parmi les deux.
  Un critère d'arrêt additionnel pourra être ajouté (par exemple nombre maximal d'itérations sans amélioration).

4. Comparaison des algorithmes glouton et de recherche locale

Nous avons comparé les performances de l'heuristique gloutonne (Greedy) et des algorithmes de recherche locale (FirstNeighborLocalSearch avec un voisinage, BestNeighborLocalSearch avec deux voisinages) sur plusieurs instances de taille variée avec des valeurs differentes (`jsp1`, `jsp10`, `jsp50`, `jsp100`). Chaque heuristique non-déterministe a été exécutée 100 fois, en conservant la meilleure solution.

Instance	Greedy (best eval)	Greedy (temps moyen)	FirstNeighborLS (best eval)	FirstNeighborLS (temps moyen)	BestNeighborLS (best eval)	BestNeighborLS (temps moyen)
jsp1	35.5	~0.000s	30.5	0.005s	32.5	0.008s
jsp10	inf (infeasible)	0.003s	208	0.859s	199.5	6.422s
jsp50	253	0.004s	185.5	0.723s	198.5	6.400s
jsp100	inf (infeasible)	0.003s	287	1.810s	268	12.480s

Analyse

- **Temps de calcul** : L'heuristique gloutonne est très rapide, quasiment instantanée, même sur les instances les plus grandes. Les recherches locales nécessitent plus de temps, surtout avec plusieurs voisinages.
- **Qualité des solutions** : La recherche locale améliore significativement la qualité par rapport au glouton sur les instances faisables (`jsp1`, `jsp50`). Sur les plus grandes instances (`jsp10`, `jsp100`), le glouton génère parfois des solutions infaisables (violations de contraintes).
- **Robustesse** : La recherche locale, en explorant plusieurs voisins, trouve des solutions réalisables de meilleure qualité, au prix d'un coût computationnel plus élevé.

Conclusion

L'heuristique gloutonne permet une construction rapide de solutions mais sans garantie de faisabilité ni de qualité optimale. Les algorithmes de recherche locale offrent un compromis en améliorant la faisabilité et la qualité, au prix d'un temps de calcul plus important, proportionnel à la complexité de l'instance et du nombre de voisinages utilisés.