Master Informatique – CNS
Année universitaire 2024–2025

Optimisation de l'Ordonnancement sous Tarification Dynamique

Rapport de Projet

Module : Travail d'étude et de recherche

Réalisés par :
BILLANE ELMEHDI
EREHAIHI SAYFE-DIN
TALEB ANASS

Encadrants :
ERIC ANGEL
VINCENT CHAU
FENG CHU

Table des matières

1. Introduction

Dans un contexte où les préoccupations liées à la consommation énergétique, à la transition écologique et à l'efficacité opérationnelle sont de plus en plus pressantes, l'optimisation des ressources devient un enjeu central dans les systèmes de production et d'information. L'ordonnancement, ou la planification de l'ordre d'exécution des tâches sur une ou plusieurs ressources, joue un rôle critique dans ce processus.

Traditionnellement, les algorithmes d'ordonnancement visaient à optimiser des critères classiques tels que le temps total d'exécution (*makespan*), le respect des échéances, ou le taux d'utilisation des machines. Cependant, l'intégration de facteurs économiques, notamment le coût énergétique variable selon les plages horaires — connu sous le nom de tarification *Time-of-Use* (TOU) — ajoute une dimension supplémentaire au problème : les décisions d'ordonnancement doivent désormais tenir compte non seulement du temps, mais aussi du moment précis d'exécution dans le but de minimiser les coûts financiers.

Ce nouveau paradigme s'avère particulièrement pertinent dans les systèmes industriels modernes, les centres de calcul à haute performance, et plus largement dans les architectures numériques où les ressources sont partagées et énergivores. Les algorithmes classiques, bien qu'efficaces pour des problèmes standardisés, ne sont plus adaptés à cette complexité croissante. Il devient alors nécessaire d'explorer des approches hybrides, heuristiques et métaheuristiques, capables de proposer des solutions de qualité dans des délais raisonnables, même pour des instances de grande taille.

L'objectif de ce rapport est d'étudier et de comparer plusieurs stratégies d'ordonnancement dans un environnement soumis à une tarification énergétique dynamique. Nous présenterons à la fois des approches classiques (telles que EDF, LLF, SPT), des heuristiques gloutonnes améliorées, ainsi que des métaheuristiques avancées comme la recherche tabou et l'algorithme génétique. L'analyse portera sur leur performance en termes de coût total, de respect des contraintes, et de robustesse sur différents scénarios expérimentaux.

2. État de l'art

Dans cette section, nous allons présenter les aspects théoriques en lien avec notre sujet. Cela inclut les bases de l'ordonnancement mono-machine, le concept de tarification dynamique de l'énergie, ainsi que les différentes méthodes déjà proposées dans la littérature pour résoudre ce type de problème.

2.1 Ordonnancement mono-machine

L'ordonnancement mono-machine est l'un des problèmes fondamentaux en recherche opérationnelle, consistant à planifier l'exécution d'un ensemble de tâches sur une seule machine, dans le but d'optimiser un ou plusieurs critères de performance. Chaque tâche est caractérisée par des paramètres tels que le temps de traitement, la date de disponibilité, la date d'échéance, et éventuellement des poids ou priorités.
L'objectif est d'optimiser un critère de performance, tel que le temps total d'achèvement des tâches, le retard total, ou le nombre de tâches en retard. Ce problème est représenté dans la notation standard des problèmes d'ordonnancement par "1" dans le premier champ, indiquant qu'une seule machine est utilisée .
L'ordonnancement mono-machine est largement étudié en raison de sa simplicité apparente et de ses nombreuses applications pratiques, notamment dans les environnements de production industrielle, la gestion de projets, et l'informatique. Il sert également de base pour comprendre des problèmes d'ordonnancement plus complexes impliquant plusieurs machines ou des contraintes supplémentaires.

2.2 Tarification dynamique de l'énergie

La tarification dynamique de l'énergie est un mécanisme de facturation dans lequel le prix de l'électricité varie en temps réel ou selon des intervalles courts, reflétant les fluctuations du marché de gros de l'électricité. Contrairement aux tarifs fixes ou aux tarifs à plages horaires (heures pleines/heures creuses), ce système permet aux consommateurs de bénéficier de prix plus bas lorsque la demande est faible et de les inciter à réduire leur consommation lors des pics de demande.
Selon la Commission de régulation de l'énergie (CRE), une offre à tarification dynamique est définie comme une offre dont le prix de l'énergie est indexé, pour au moins 50 %, sur un ou plusieurs indices de prix des marchés de gros au comptant (marché journalier ou infra-journalier), reflétant ainsi les variations de ces prix au minimum à l'échelle horaire .
Ce type de tarification vise à encourager les consommateurs à adapter leur consommation en fonction des signaux de prix, contribuant ainsi à l'équilibre du réseau électrique et à une meilleure intégration des énergies renouvelables. En France, bien que la tarification dynamique soit encore peu répandue, des initiatives ont été lancées pour promouvoir son adoption, notamment avec l'arrivée de fournisseurs proposant des offres indexées sur les

marchés de gros[1].

2.3 Approches de résolution : définitions et concepts

Dans cette section, nous présentons les principales approches utilisées dans ce travail pour résoudre le problème d'ordonnancement sous tarification dynamique. Il s'agit d'approches exactes, heuristiques et métaheuristiques.

2.3.1 Programmation linéaire en nombres entiers (PLNE)

La programmation linéaire en nombres entiers (PLNE) est une méthode d'optimisation mathématique visant à résoudre des problèmes où les variables de décision doivent prendre des valeurs entières, tout en respectant des contraintes linéaires. Elle est largement utilisée dans des domaines tels que la logistique, la planification industrielle, l'ordonnancement de tâches ou les réseaux de transport, où des décisions discrètes (ex : nombre de camions, affectation binaire de ressources) sont nécessaires. Contrairement à la programmation linéaire continue, la PLNE introduit une complexité computationnelle accrue due à la nature combinatoire des solutions entières, la classant souvent parmi les problèmes NP-difficiles. Les approches classiques incluent des algorithmes exacts comme Branch and Bound ou Cutting Planes, ainsi que des méthodes heuristiques (ex : métaheuristiques) pour les cas de grande taille. Son application requiert une modélisation rigoureuse pour équilibrer précision et faisabilité, notamment via des outils comme CPLEX, Gurobi ou des librairies open-source (ex : PuLP en Python)[2][3].

2.3.2 Heuristiques

Les heuristiques sont des méthodes approchées conçues pour résoudre des problèmes d'optimisation complexes où les approches exactes (ex : programmation linéaire, énumération complète) s'avèrent trop coûteuses en temps ou en ressources. Elles visent à fournir des solutions réalistes et de qualité acceptable dans un délai raisonnable, même si l'optimalité n'est pas garantie. Employées dans des contextes comme l'ordonnancement, le routage de véhicules, l'allocation de ressources ou les problèmes NP-difficiles (ex : voyageur de commerce), elles reposent sur des règles empiriques, des logiques gloutonnes ou des mécanismes inspirés de phénomènes naturels (métaheuristiques : algorithmes génétiques, recuit simulé, colonies de fourmis). Contrairement aux méthodes exactes, les heuristiques privilégient l'efficacité sur la précision, ce qui les rend adaptées aux systèmes dynamiques ou aux instances de grande taille. Leur conception implique souvent un compromis entre exploration (diversification des solutions) et exploitation (amélioration locale). Des outils comme OR-Tools, Google's VRP Solver ou des bibliothèques Python (ex : DEAP, Scikit-opt) facilitent leur implémentation[4].

2.3.3 Métaheuristiques

Les métaheuristiques sont des stratégies algorithmiques génériques conçues pour résoudre des problèmes d'optimisation complexes, souvent NP-difficiles, en guidant des heuristiques plus spécifiques vers des solutions quasi-optimales. Contrairement aux heuristiques classiques, elles intègrent des mécanismes pour éviter les optima locaux et explorer efficacement l'espace de recherche. Inspirées par des phénomènes naturels, biologiques ou physiques (ex : évolution darwinienne, comportement de colonies de fourmis, recuit métallurgique), elles incluent des méthodes telles que les algorithmes génétiques, le recuit simulé, l'optimisation par essaims de particules ou les algorithmes de colonies de fourmis. Ces approches sont particulièrement utiles pour des problèmes comme le voyageur de commerce, l'optimisation de portefeuille, ou la conception de réseaux, où la combinatoire rend les méthodes exactes inapplicables. Leur force réside dans leur flexibilité et leur capacité à traiter des problèmes non linéaires, multi-objectifs ou stochastiques. Cependant, leur performance dépend fortement du réglage des paramètres (ex : taux de mutation, température initiale) et d'un équilibre entre exploration (diversité des solutions) et exploitation (approfondissement local). Des bibliothèques comme DEAP, Optuna ou Hyperopt simplifient leur implémentation, tandis que des frameworks hybrides (ex : combinaison avec des méthodes exactes) émergent pour améliorer leur efficacité[5].

2.3.4 Recherche Tabou

La Recherche Tabou est une métaheuristique d'optimisation combinatoire conçue pour échapper aux optima locaux en utilisant une mémoire à court et long terme. Développée par Fred Glover dans les années 1980, elle interdit temporairement les solutions récemment visitées (via une liste tabou) pour forcer l'exploration de nouvelles régions de l'espace de recherche. Elle combine des mouvements voisinage (ex : permutations, ajouts/suppressions) avec une gestion adaptive des interdictions, permettant de balancer exploration et exploitation. Ses applications incluent l'ordonnancement, le routage de véhicules, ou la conception de réseaux. Contrairement à des méthodes purement aléatoires, la Recherche Tabou est déterministe et repose sur des critères d'aspiration (pour accepter des solutions taboues si elles améliorent le meilleur résultat connu). Ses paramètres clés sont la taille de la liste tabou et la durée d'interdiction. Des outils comme OptaPlanner ou des implémentations personnalisées (en Python/Java) facilitent son application[6].

2.3.5 Algorithme génétique

Les Algorithmes Génétiques (AG) sont des métaheuristiques inspirées de la sélection naturelle darwinienne, utilisant des opérations de croisement, mutation et sélection pour évoluer une population de solutions potentielles. Chaque individu (solution) est encodé comme un chromosome (ex : chaîne binaire, permutation), et sa fitness (qualité) détermine sa probabilité de se reproduire. Les AG favorisent une exploration robuste de l'espace de

recherche grâce à leur diversité initiale et leur capacité à combiner des solutions (croisement), tout en maintenant une exploitation via la sélection des meilleurs individus. Ils sont efficaces pour des problèmes complexes non linéaires, multi-objectifs ou à variables mixtes (ex : conception de circuits, optimisation de paramètres en Machine Learning). Leur performance dépend de paramètres comme le taux de mutation, la taille de la population ou le mécanisme de sélection (ex : roulette, tournoi). Des bibliothèques comme DEAP (Python), JGAP (Java) ou des frameworks (ex : Google's OR-Tools) simplifient leur implémentation[7][8].

3. Méthodes de résolution

3.1 Approches exactes : Modèle MILP

Dans un premier temps, une approche exacte est proposée pour résoudre le problème d'ordonnancement dynamique dans un environnement critique soumis à des coûts variables d'exécution. Le modèle est formulé comme un programme linéaire en nombres entiers mixtes (**MILP**), intégrant les contraintes temporelles, les ressources disponibles et les coûts horaires d'exécution.

Paramètres d'entrée :

- A_i : Date d'arrivée de la tâche i
- B_i : Deadline de la tâche i
- D_i : Durée d'exécution de la tâche i
- C_t : Coût horaire d'exécution à l'instant t

Variables de décision :

- $x_{i,t} \in \{0, 1\}$: vaut 1 si la tâche i est exécutée à l'instant t
- $y_i \in \{0, 1\}$: 1 si la tache i est choisie, 0 sinon
- w_i : poids de la tache i indique si la tâche i est exécutée ($y_i = 1$) ou non

Fonction objectif :

$$\min \sum_i \sum_t C_t \cdot x_{i,t} + \sum_i w_i \cdot (1 - y_i)$$

où w_i est un poids (ou coût de pénalité) associé à la non-exécution de la tâche i.

Contraintes :

$$\sum_t x_{i,t} = y_i \cdot D_i \qquad \forall i \quad \text{(chaque tâche est complétée au plus une fois)}$$

$$\sum_i x_{i,t} \leq 1 \qquad \forall t \quad \text{(maximum une tache par heure)}$$

$$\sum_{t=0}^{A_i - 1} x_{i,t} = 0 \qquad \forall i \quad \text{(pas avant arrivée)}$$

$$\sum_{t=B_i+1} x_{i,t} = 0 \qquad \forall i \quad \text{(pas après deadline)}$$

Résolution : Le modèle est résolu à l'aide d'un solveur MILP comme `CPLEX` ou `Gurobi`. Ce solveur retourne une planification optimale qui respecte les contraintes temporelles et minimise le coût total.

Résultats attendus :

- Planification optimale des tâches dans l'horizon temporel donné
- Coût total d'exécution minimal selon les tarifs horaires
- Liste des tâches exécutées et leur positionnement temporel

3.2 Heuristiques classiques : EDF, LLF, SPT

Les heuristiques classiques, présentées dans l'état de l'art, sont ici mises en œuvre comme approches de référence pour l'évaluation des performances. Bien qu'elles n'intègrent pas les coûts horaires d'exécution, elles permettent une première comparaison sur les critères temporels et la simplicité algorithmique.

Application au problème : Chaque heuristique est appliquée au même ensemble de tâches, en tenant compte des contraintes de fenêtres temporelles, de durée, et de ressource unique, mais sans considération du coût énergétique variable.

- **EDF (Earliest Deadline First)** : les tâches sont triées par deadline croissante et insérées dès que la ressource est disponible.
- **LLF (Least Laxity First)** : les tâches sont sélectionnées dynamiquement en fonction de leur laxité minimale (marge entre la deadline et le temps restant).
- **SPT (Shortest Processing Time)** : les tâches de durée minimale sont priorisées, indépendamment de leur deadline.

Objectif : Ces heuristiques cherchent à optimiser des critères comme le respect des deadlines ou le taux d'achèvement, mais sans prise en compte du coût énergétique. Elles servent ainsi de *baseline* pour comparer les performances des approches coût-sensibles comme les heuristiques dédiées (ex : DCS) ou les métaheuristiques.

Limitations observées : En présence d'une tarification dynamique, ces méthodes produisent des ordonnancements rapides, mais souvent sous-optimaux en termes de coût total. Leur efficacité décroît particulièrement lorsque les prix de l'électricité varient fortement au cours de la journée.

3.3 Heuristique composite

En gardant en tête les limitations des premières heuristiques vues précédemment, nous essayerons maintenant de surmonter certaines d'entre elles. Une première approche pourrait consister à utiliser une heuristique ressemblant à celle de LLF, mais un peu plus sophistiquée. Cette fois, l'heuristique dépend de l'heure à laquelle l'algorithme effectue le calcul en question.

En révisant les notations du PLNE en 3.1, on a :

— D_i représente désormais le temps restant d'exécution de la tâche au moment du calcul.

— $curr\_time$ désigne l'heure actuelle.

$$h(\text{curr\_time}) = \begin{cases} 0 \text{ si } \begin{cases} curr\_time < A_i \text{ ou} \\ curr\_time + D_i > B_i \end{cases} \\ (B_i - \text{curr\_time})/D_i \text{ sinon} \end{cases}$$

Cette heuristique met en avant l'urgence d'une tâche. Comme mentionné précédemment, elle ressemble fortement à Least Laxity First (LLF). Une valeur de 0 indique une tâche qui n'est pas encore entrée dans l'horaire actuel (curr_time $< A_i$) ou une tâche abandonnée (curr_time $+ D_i > B_i$).

Le détail qui différencie cette heuristique de LLF est le paramètre d'entrée : l'heure au moment du calcul, ce qui permet d'obtenir une valeur spécifique pour chaque tâche à chaque instant.

La valeur que $h(curr\_time)$ prend, dans le cas où la tâche n'est pas abandonnée et est exécutable, se situe dans le domaine de définition $[1; B_i - curr\_time]$. La tâche remplissant au mieux l'objectif de l'heuristique (l'urgence) est celle dont la valeur est la plus proche de 1. Cela nous permet de prioriser les tâches ayant un temps d'exécution restant le plus proche du temps restant avant la deadline.

Cette heuristique tire un avantage important de la préemption par rapport aux premières heuristiques étudiées, mais cela représente aussi sa faiblesse, car il est toujours autorisé d'exécuter une tâche sans nécessairement la terminer immédiatement.

Une première amélioration serait d'ajouter un paramètre permettant d'imposer un certain niveau de risque, exprimé sous forme d'un nombre d'heures à exécuter dans la plage tarifaire la moins chère. Cela permettrait à l'algorithme de conserver ce nombre d'heures d'exécution pour plusieurs tâches non terminées, afin de les planifier dans les créneaux repérés précédemment. Toutefois, nous n'avons pas encore finalisé les tests de cet algorithme.

3.4 Métaheuristiques :

Face à la complexité croissante du problème d'ordonnancement sous tarification dynamique, des métaheuristiques sont mises en œuvre pour explorer efficacement l'espace de solutions tout en tenant compte du compromis entre performance et coût énergétique.

3.4.1 Recherche Tabou

La recherche tabou est une méthode itérative basée sur la recherche locale, enrichie par une mémoire qui empêche le retour cyclique vers des solutions déjà visitées.

Principe :

1. Générer une solution initiale (par exemple via SPT ou DCS).
2. À chaque itération :
   — Générer le voisinage (en échangeant l'ordre de deux tâches, par exemple).
   — Évaluer les solutions voisines selon la fonction coût (énergie + délai).
   — Sélectionner la meilleure solution non taboue ou satisfaisant un critère d'aspiration.
   — Mettre à jour la liste taboue (mouvements interdits pour k itérations).
3. Arrêter après un nombre fixe d'itérations ou de stagnation.

Paramètres :

— Taille de la liste taboue
— Critère d'aspiration (autoriser une solution interdite si elle améliore le meilleur global)
— Nombre d'itérations ou seuil de convergence

Avantages :

— Bonne exploration de l'espace de recherche
— Moins sensible aux optima locaux qu'une recherche gloutonne

3.4.2 Algorithme Génétique

Inspiré de l'évolution naturelle, l'algorithme génétique explore l'espace des solutions via des mécanismes de sélection, croisement, et mutation.

Principe :

1. Initialiser une population de solutions (ordonnancements aléatoires ou issus d'une heuristique).
2. Évaluer chaque individu via une fonction de fitness (coût énergétique + respect des délais).

3. Sélectionner des parents (roulette, tournoi...).
4. Appliquer le croisement (par exemple PMX ou OX) pour générer de nouveaux individus.
5. Appliquer une mutation aléatoire (échange de deux tâches, inversion de segment).
6. Conserver les meilleurs individus pour la génération suivante (élistisme).
7. Répéter pendant un nombre défini de générations.

Paramètres :

— Taille de la population
— Taux de croisement et de mutation
— Nombre de générations
— Stratégie de sélection

Avantages :

— Bonne diversité des solutions
— Adaptabilité à différents types de contraintes
— Peut approcher des solutions quasi-optimales avec un temps de calcul raisonnable

Comparaison attendue : L'algorithme génétique permet souvent d'obtenir des solutions de meilleure qualité que la recherche tabou, mais au prix d'un temps de calcul plus important. La recherche tabou est en revanche plus rapide et plus simple à mettre en œuvre.

4. Expérimentations et résultats

4.1 Génération des données

Les instances de test sont générées aléatoirement selon des distributions contrôlées pour reproduire des scénarios réalistes d'ordonnancement.

Paramètres par tâche :

- A_i (date d'arrivée) : échantillonné uniformément dans $[0, 100]$
- D_i (durée) : $D_i \in [10, 50]$, avec $D_i > 0$
- B_i (deadline) : $B_i = A_i + D_i + \delta$, avec $\delta \in [1, 100]$
- W_i (poids ou priorité) : $W_i \in [50, 500]$

Ces paramètres permettent de couvrir une large variété de cas tout en respectant les contraintes de planification (fenêtre temporelle cohérente, poids significatif).

Protocoles d'expérimentation : Les groupes de test sont définis comme suit :

Groupe	**Nombre de tâches (n)**	**Instances générées**
G1	5	50
G2	10	50
G3	15	50
G4	20	50
Total		**200 instances testées**

4.2 Analyse des résultats

Les performances des différentes approches sont évaluées selon deux critères :

- Le **temps moyen d'exécution**, qui mesure l'efficacité algorithmique.
- Le **coût moyen des solutions**, qui reflète la performance énergétique sous tarification dynamique.

Temps d'exécution moyen

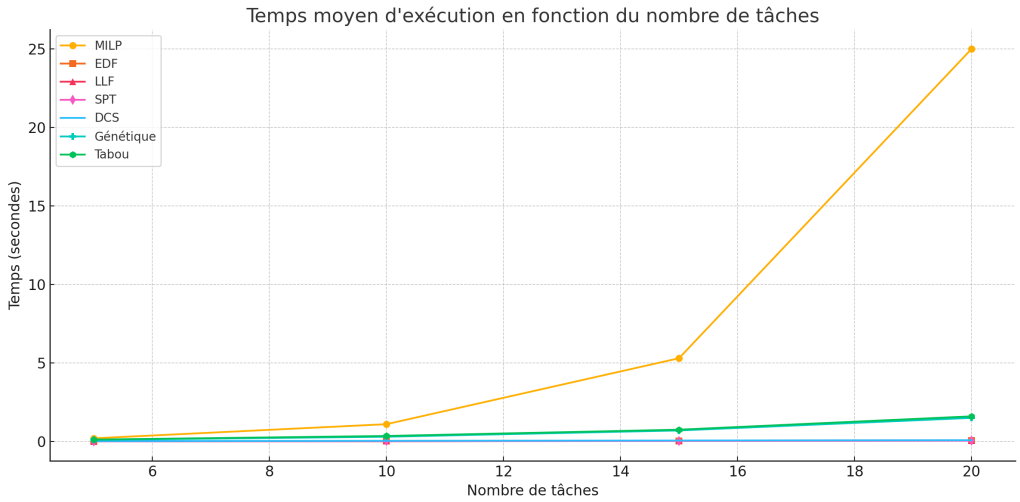

FIGURE 1 – Temps moyen d'exécution en fonction du nombre de tâches

Les résultats montrent que l'approche **MILP** fournit les meilleurs coûts mais devient rapidement impraticable au-delà de 15 tâches. Les heuristiques classiques (EDF, LLF, SPT) ainsi que DCS sont très rapides, quel que soit le nombre de tâches. Les métaheuristiques (Génétique, Tabou) offrent un bon compromis entre performance et temps de calcul.

Coût moyen des solutions

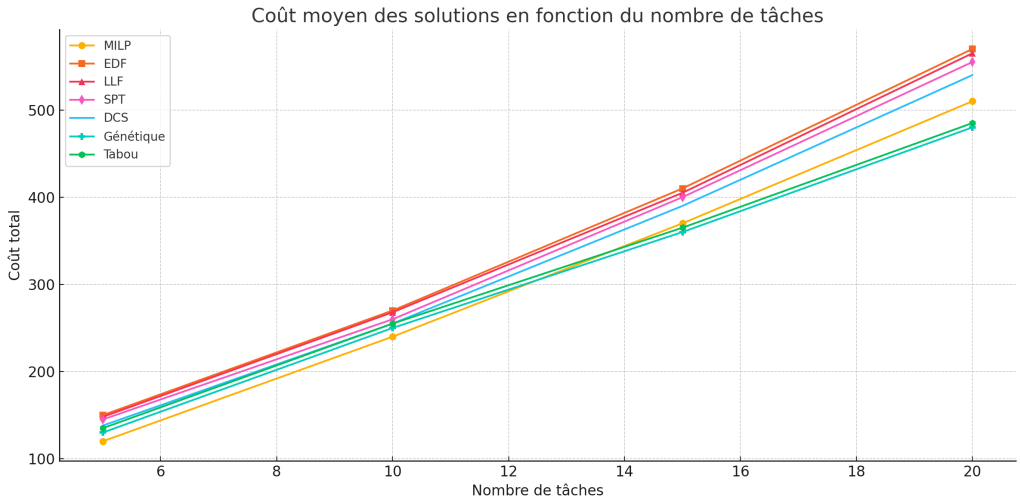

FIGURE 2 – Coût moyen des solutions en fonction du nombre de tâches

Le coût moyen confirme que l'approche **MILP** est la plus performante pour de petits ensembles de tâches. Les heuristiques classiques produisent des coûts plus élevés. L'heu-

ristique DCS améliore significativement cet équilibre. Les métaheuristiques fournissent une qualité proche de l'optimal avec un temps de calcul raisonnable.

Synthèse comparative

- **MILP** : solution optimale mais impraticable au-delà de 15 tâches.
- **Heuristiques classiques (EDF, LLF, SPT)** : très rapides, mais sous-optimales.
- **DCS** : rapide, avec des performances meilleures que les heuristiques simples.
- **Métaheuristiques (Génétique, Tabou)** : bon compromis entre coût, qualité et temps.

5. Conclusion

Dans un contexte industriel de plus en plus sensible à la consommation énergétique et à la variabilité des tarifs d'électricité, ce travail s'est attaché à étudier le problème de l'ordonnancement de tâches sous tarification dynamique. L'objectif principal était de proposer, modéliser et évaluer différentes approches permettant d'optimiser les plannings de production tout en minimisant le coût énergétique global, dans un environnement à contraintes critiques.

Nous avons d'abord formulé le problème sous forme d'un modèle exact (MILP), capable de produire des solutions optimales intégrant les contraintes de temps, de ressources, et les coûts horaires. Cette approche s'est révélée très performante pour de petites tailles de problèmes, mais rapidement impraticable au-delà d'un certain nombre de tâches, en raison de la complexité combinatoire croissante.

Pour répondre à cette limite, plusieurs méthodes heuristiques et métaheuristiques ont été développées. Les heuristiques classiques (EDF, LLF, SPT), bien que très rapides, se sont avérées limitées en termes de performance énergétique. L'heuristique composite proposée (DCS), prenant en compte simultanément les coûts, la durée et la priorité des tâches, a démontré de meilleurs résultats tout en conservant une faible complexité.

Enfin, les métaheuristiques (algorithme génétique et recherche tabou) ont permis d'approcher des solutions de haute qualité avec un temps de calcul raisonnable, offrant un compromis robuste entre coût et performance. Ces méthodes ont montré leur capacité à s'adapter à des profils tarifaires variés et à des contraintes opérationnelles réalistes.

Les expérimentations menées sur un ensemble de 200 instances aléatoires ont permis de valider l'efficacité des approches proposées, et de mettre en évidence les atouts et limites de chaque méthode selon la taille du problème et les objectifs de planification.

6. Perspectives

Plusieurs pistes d'amélioration peuvent être envisagées pour prolonger ce travail :

— **Extension à un environnement multi-machines** : le problème traité dans ce mémoire suppose une ressource unique (une seule machine). Une suite logique serait d'étendre les approches à un *atelier avec plusieurs machines*, ce qui reflète mieux les réalités industrielles (ex. : job-shop, flow-shop). Cela nécessiterait de revoir les modèles et les heuristiques pour gérer l'allocation des tâches à différentes machines, tout en tenant compte des coûts énergétiques.

— **Cas applicatif réel** : adapter le modèle à un secteur précis (industrie textile, agroalimentaire, etc.) pour démontrer la faisabilité en conditions concrètes.

— **Visualisation interactive** : développer un outil simple (en Python ou via une interface web) permettant aux utilisateurs de visualiser l'ordonnancement généré, le coût associé et les éventuelles contraintes, afin de faciliter la prise de décision.

Références

[1] Commission de régulation de l'énergie (CRE). (2023). *Les offres à tarification dynamique*. Rapport annuel sur les marchés de l'électricité.

[2] Hillier, F. S., & Lieberman, G. J. (2021). *Introduction to Operations Research* (12th ed.). McGraw-Hill.

[3] Wolsey, L. A. (2020). *Integer Programming* (2nd ed.). Wiley.

[4] Pearl, J. (1984). *Heuristics : Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

[5] Glover, F., & Kochenberger, G. (2022). *Handbook of Metaheuristics* (3rd ed.). Springer.

[6] Glover, F. (1989). Tabu Search – Part I. *ORSA Journal on Computing*, 1(3), 190–206.

[7] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

[8] Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT Press.