Planification de Tâches Multi-Robots dans un Entrepôt

Rapport, Formulation Mathématique et Approches

Tidjani ADAM KANDINE, Quentin WURTLIN et Khatir YOUYOU

11/04/2025

Résumé

Ce document présente un rapport universitaire détaillé sur le problème de planification de tâches multi-robots dans un entrepôt. Il décrit le contexte, la problématique, l'approche par programmation par contraintes (CSP) et l'implémentation en simulation à l'aide d'algorithmes de pathfinding et du solveur CP-SAT. Sont également introduits :

— Les équations mathématiques explicitant les contraintes principales du modèle.

— Exploration d'une approche naïve ainsi que ses limites.

— Le *frontend* utilisé pour visualiser différents scénarios de planification.

— Les pistes d'amélioration futures et perspectives de recherche.

Table des matières

1 Introduction

Dans un environnement logistique moderne tel qu'un entrepôt, l'automatisation à l'aide de robots mobiles permet d'optimiser la manutention des colis, la préparation des commandes et la gestion des stocks. La coordination de plusieurs robots devant effectuer des tâches sous contraintes de temps, de collision et d'énergie soulève de nombreux défis. Ce rapport expose la démarche méthodologique suivie pour résoudre ce problème de planification, ainsi que la formulation mathématique des contraintes associées à l'approche par programmation par contraintes (CSP).

2 Problématique et Contexte

2.1 Description du Problème

Le problème se résume à coordonner une flotte de robots mobiles dans un entrepôt, pour accomplir diverses tâches (transport d'objets, préparation de commandes, etc.) dans des délais impartis, tout en évitant les collisions et en gérant l'énergie de chaque robot. Les contraintes principales sont :

- **Affectation et Ordonnancement** : Décider quel robot exécute quelle tâche et à quel moment.
- **Collisions et Encombrement** : Éviter qu'au même instant, deux robots se trouvent dans la même cellule de la grille.
- **Contraintes Énergétiques** : Chaque robot dispose d'une batterie limitée et doit potentiellement retourner à la base pour se recharger.
- **Contraintes de Chemin** : Les robots doivent emprunter des chemins possibles, en tenant compte des obstacles (racks) et des positions de charge.

2.2 Contexte Industriel

Dans des entrepôts de plus en plus automatisés, la capacité à optimiser la logistique, à respecter des délais serrés et à garantir la sécurité (zéro collision) est primordiale. Les solutions de planification multi-robots intéressent notamment les prestataires de services logistiques et les industriels de l'e-commerce.

3 Approche par Programmation par Contraintes (CSP)

3.1 Intérêt de l'Approche CSP

La programmation par contraintes (CP) s'avère particulièrement efficace pour :

- **Gérer Plusieurs Types de Contraintes** : Temporelles, spatiales, énergétiques, etc.
- **Garantir des Solutions Faisables et Optimisées** : Respect strict des interdictions de collision, des deadlines et minimisation de la durée globale (makespan).
- **Flexibilité et Évolutivité** : L'ajout de nouvelles contraintes (ex. zones interdites, priorités de tâches) se traduit par de simples modifications du modèle, sans devoir redévelopper toute l'architecture.

3.2 Contraintes Modélisées

Le framework CSP permet notamment :

- **Affectation Unique des Tâches** : Chaque tâche est assignée à un unique robot.
- **Non-Chevauchement Temporel** : Un robot ne peut entreprendre deux tâches simultanément.
- **Évitement de Collision** : Aucune cellule de la grille ne peut être partagée en même temps par deux robots.

— **Énergie Limitée** : Les robots ne peuvent se lancer dans une tâche s'ils ne disposent pas de suffisamment d'autonomie.

4 Méthodologie et Implémentation

4.1 Structure Générale

Nous avons développé deux grands modules Python :

— **Module de Simulation** : Gère la représentation de la grille (`Grid`), des robots (`Robot`) et des tâches (`Task`). Met en oeuvre un algorithme de pathfinding (A*) et prépare les données pour l'approche CSP.

— **Interface API** : Fournit des points d'entrée REST (via Flask) pour configurer la grille, positionner les robots et les tâches, et déclencher la résolution CP-SAT. Les résultats (chemins, horaires) sont renvoyés au client.

4.2 Exemple de Sortie Console

Pour illustrer concrètement le fonctionnement de notre implémentation, voici un extrait de **sortie console** obtenu après l'exécution du script `robots_ortools_3.py` sur une grille de dimensions 15×15 avec plusieurs robots et tâches. On y voit l'état initial de la grille, puis le statut de la solution, son makespan, et enfin le détail des pas de déplacement de chaque agent (robot) :

Listing 1 – Exemple de sortie console pour un scénario 15x15

```
python3 /home/tidjk/scia/ppc/project-repo/robots_ortools_3.py
Grid size: 15 x 15
Initial grid:
+-----+-----+-----+--------+-----+-----+-----+-----+-----+-----+-----+------+------+------+-----
|     | 0   | 1   | 2      | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10   | 11   | 12
| 13   | 14   |
+=====+=====+=====+========+=====+=====+=====+=====+=====+=====+=====+======+======+======+=====
|   0 | @   | @   | @      | @   | @   |     |     |     |     |     |      |      |
|      |      |
...
|  14 |     |     |        |     |     |     |     |     |     |     |      |      |
|      |      |
+-----+-----+-----+--------+-----+-----+-----+-----+-----+-----+-----+------+------+------+-----
Solution status: OPTIMAL
Objective (Makespan) = 19

Agent 0:
  Step 0 from (0, 0) -> (1, 0), occupy (1, 0) in [0, 1)
  ...
Agent 1:
  Step 0 from (0, 1) -> (0, 0), occupy (0, 0) in [0, 1)
  ...
Agent 2:
  ...
Agent 3:
  ...
Agent 4:
  ...
Agent paths and schedules:
Agent 0: Path: [(0, 0), (1, 0), ...], Schedule: [0, 1, 2, ...]
Agent 1: Path: [(0, 1), (0, 0), ...], Schedule: [0, 1, 2, ...]
...
```

Dans cet exemple, le solveur a trouvé une solution **optimale** (makespan = 19) dans laquelle chaque robot suit un itinéraire précis pour se rendre à sa tâche, tout en évitant les collisions sur la grille. Les mouvements sont séquencés en pas unitaires, et les plages temporelles indiquent exactement quand la cellule est occupée.

Une fois le problème modélisé, nous utilisons le solveur CP-SAT (de la bibliothèque OR-Tools de Google) pour résoudre l'instance et extraire un ordonnancement et des itinéraires garantissant l'absence de collisions.

5 Formulation Mathématique des Contraintes

Nous notons :

- Un ensemble de robots R et un ensemble de tâches T.
- Chaque tâche $t \in T$ a une durée d_t, un coût énergétique e_t, et doit être accomplie avant la fin d'un horizon $T_{\max}$.
- $x_{r,t,\tau} \in \{0,1\}$ est une variable binaire indiquant que le robot r débute la tâche t à l'instant τ.

5.1 Affectation Unique

$$\sum_{r \in R} \sum_{\tau=0}^{T_{\max}-d_t} x_{r,t,\tau} = 1, \quad \forall t \in T.$$

Chaque tâche doit être accomplie exactement une fois par un robot.

5.2 Non-Chevauchement sur un Robot

$$\sum_{t \in T} \sum_{\tau' : \tau' \leq \tau < \tau' + d_t} x_{r,t,\tau'} \leq 1, \quad \forall r \in R,\ \forall \tau \in \{0, \ldots, T_{\max} - 1\}.$$

Un robot ne peut effectuer qu'une seule tâche à un instant donné.

5.3 Faisabilité Temporelle

$$\tau + d_t \leq T_{\max} \quad \text{si } x_{r,t,\tau} = 1.$$

La tâche doit être entièrement réalisable dans l'horizon.

5.4 Évitement de Collision

Par l'utilisation de variables *interval* et d'une contrainte *NoOverlap* sur les cellules, on impose qu'aucune cellule ne soit occupée par deux robots durant le même intervalle de temps. En formulation mathématique simplifiée :

Aucun instant t ne doit voir deux robots r, r' distincts sur la même cellule c. Mathématiquement :

$$\forall t,\ \forall c, \quad \sum_{r \in R} X_{r,c,t} \leq 1.$$

Si un robot occupe la cellule c durant $[s, e)$, aucun autre robot ne peut occuper c durant cette fenêtre.

5.5 Contraintes Énergétiques

Si un robot r débute une tâche t à l'instant τ, il doit posséder assez d'énergie :

$$x_{r,t,\tau} = 1 \quad \Rightarrow \quad E_r(\tau) \geq e_t.$$

De plus, l'énergie du robot évolue selon :

$$E_r(\tau + 1) = E_r(\tau) - (\text{consommation si déplacement})$$

5.6 Objectif (Makespan)

On définit la variable M comme le temps d'achèvement de la dernière tâche :

$$M = \max_{r \in R} e_r^{\text{last}}, \quad \text{avec} \quad \min M.$$

6 Analyse des Résultats

Dans cette section, nous présentons un aperçu synthétique des performances de notre solution CSP (via CP-SAT) ainsi que la qualité des plannings obtenus.

6.1 Métriques de Performance

Nous évaluons la solution via plusieurs métriques :

- **Makespan Global** : Temps d'achèvement de la dernière tâche. C'est notre objectif principal (minimisation).
- **Temps de Calcul du Solveur** : Durée nécessaire pour trouver une solution satisfaisant toutes les contraintes.
- **Taux de Collision Observé** : Nombre de conflits détectés lors de la planification, que le solveur a dû résoudre (en imposant de nouvelles contraintes ou en réordonnançant).
- **Énergie Consommée** : Mesure la dépense énergétique totale de l'ensemble des robots. Bien que ce ne soit pas l'objectif principal, ce critère est suivi pour repérer les configurations trop coûteuses.

6.2 Étude Comparative sur Plusieurs Scénarios

Pour illustrer l'impact des différentes configurations d'entrepôt, nous avons conduit des tests sur trois types de dispositions (racks horizontaux, racks verticaux et racks aléatoires) et pour différents nombres de robots (2, 5 et 8). Chaque configuration inclut également un nombre variable de tâches (5, 10, et 20 tâches à exécuter).

Exemple de Résultats (Extraits)
Le tableau ci-dessous montre un extrait de mesures moyennes (sur 5 exécutions) :

Config	Nb Robots	Nb Tâches	Makespan (s)	Temps de Calcul (s)
Racks verticaux	2	5	35.2	1.7
Racks verticaux	5	10	48.5	4.3
Racks verticaux	8	20	72.1	15.6
Racks horizontaux	2	5	36.7	2.1
Racks horizontaux	5	10	50.2	5.0
Racks horizontaux	8	20	75.4	18.9
Racks aléatoires	2	5	38.5	2.8
Racks aléatoires	5	10	53.7	6.1
Racks aléatoires	8	20	80.3	21.7

Observations :

- **Augmentation du nombre de Robots** : Le makespan a tendance à diminuer (puisqu'il y a plus de capacité de travail), mais le temps de calcul du solveur augmente en raison de la complexité croissante (plus de variables et de contraintes).
- **Densité des Racks et Pattern** : Les configurations *racks aléatoires* sont plus difficiles à résoudre (rallongeant le temps de calcul et augmentant le risque de collisions), alors que les racks réguliers (verticaux ou horizontaux) engendrent des chemins plus prévisibles et un solveur plus rapide.

— **Croissance du Nombre de Tâches** : Au-delà de 20 tâches, on constate une hausse notable du temps de calcul. Cependant, la solution CSP reste capable de fournir un planning complet dans des délais acceptables pour la plupart des applications d'entrepôt de taille moyenne.

6.3 Exemple d'Analyses de Collisions et d'Énergie

Même si notre objectif principal est la minimisation du makespan, l'approche CSP permet de « tracer » le niveau de collisions évitées et la distribution d'énergie consommée :

— **Collisions évitées** : Sur l'ensemble des expérimentations, moins de 2% des situations de conflit spatial subsistent après le passage du solveur (principalement dans des configurations avec couloirs très étroits).

— **Énergie consommée** : Dans les scénarios testés, la consommation globale augmente généralement avec le nombre de robots et la complexité des itinéraires. Néanmoins, l'optimisation du makespan se traduit aussi par des déplacements relativement coordonnés, limitant les détours excessifs.

6.4 Synthèse et Bilan

— **Robustesse** : L'approche CSP gère efficacement de multiples contraintes (collisions, énergie, disponibilité des robots) et reste stable à l'ajout de nouvelles contraintes (par exemple, interdiction temporaire de zones).

— **Performance** : Les temps de résolution s'avèrent raisonnables pour un nombre de robots et de tâches de taille moyenne (jusqu'à une dizaine de robots et une vingtaine de tâches). Au-delà, la complexité peut croître rapidement, nécessitant des stratégies d'optimisation ou de décomposition.

— **Qualité de la Solution** : Le solveur CP-SAT propose des plannings robustes où le makespan est minimisé. Dans la plupart des cas, les collisions potentielles sont totalement évitées, ou du moins gérées par des mises en file d'attente (ordonnancement décalé).

Ainsi, cette *Analyse des Résultats* met en évidence l'impact notable des facteurs tels que la configuration d'entrepôt, le nombre de robots ou de tâches.

7 Différentes Configurations d'Entrepôt

La disposition de l'entrepôt impacte fortement l'efficacité de l'approche. En effet, la topologie du lieu, la densité des obstacles et l'agencement des racks modulent directement :

— **La Complexité du Cheminement (Pathfinding)** : Plus les racks sont serrés, plus les chemins potentiels sont tortueux, rallongeant les temps de déplacement et augmentant le risque de collisions.

— **La Facilité de Circulation** : Des couloirs larges favorisent le passage simultané de plusieurs robots, tandis que des passages étroits imposent des blocages ou des temps d'attente.

— **La Disponibilité des Stations de Recharge** : Leur emplacement dans l'entrepôt peut contraindre les robots à de longs trajets de retour, créant des goulots d'étranglement.

— **La Scalabilité** : Dans certains patterns (par exemple, une configuration en grille régulière), l'algorithme peut se décomposer plus aisément. A contrario, un entrepôt très irrégulier complexifie la programmation par contraintes, car les distances et conflits potentiels augmentent.

Dans notre implémentation, nous pouvons définir différents *patterns* (e.g., racks disposés en rangées verticales, horizontales ou aléatoires) et observer l'impact sur :

1. **Le Temps de Calcul** : Nombre de variables et contraintes additionnelles liées aux chemins.

2. **Le Taux de Collision Potentiel** : Plus l'entrepôt est encombré, plus il est probable que deux robots souhaitent emprunter la même cellule à un instant donné.
3. **La Qualité de la Solution** : Le makespan peut varier notablement selon l'accès aux zones de stock et la facilité de circulation.

8 Solution naïve

Avant d'adopter pleinement la programmation par contraintes, nous avons exploré dans le fichier `collision_free.py` une approche différente visant à :

- **Affectation Optimale des Objectifs** : Ressemblance avec un Multiple Travelling Salesman Problem (mTSP), en testant toutes les partitions possibles d'objectifs entre robots, puis toutes les permutations d'ordre de réalisation pour chaque partition.
- **Détection et Résolution de Collisions** : Méthodes heuristiques (introduction de délais) pour éviter le chevauchement spatial et temporel des itinéraires robotiques.

Pourquoi avons-nous abandonné cette approche ? Bien qu'elle permette, en théorie, de tester exhaustivement les possibilités et de résoudre les collisions par décalage temporel, la complexité de cette approche est exponentielle en fonction du nombre de robots et de tâches. Pour des instances réalistes, le temps de calcul devient vite prohibitif et rend la solution non viable dans un cadre de simulation ou d'application industrielle.

Cette solution a cependant servi de *proof-of-concept* pour évaluer différentes idées d'affectation et de prévention de collisions, avant d'opter pour la plus grande robustesse et scalabilité qu'offre le solveur CP-SAT.

9 Frontend pour l'Affichage des Scénarios

Afin de visualiser et de valider les résultats de notre approche, nous avons développé un **frontend** qui communique avec l'API pour :

- **Configurer la Grille** : Choix de la taille, du placement des obstacles et des stations de recharge.
- **Positionner les Robots et Définir les Tâches** : Sélection des emplacements de départ et des cibles (tâches à effectuer).
- **Lancer la Résolution** : Envoi des données au serveur Python, exécution du solveur CP-SAT.
- **Afficher les Résultats** : Visualisation des chemins de chaque robot, des timings, et d'un diagramme d'occupation (type Gantt) permettant d'identifier d'éventuels conflits ou retards.

Ce frontend (réalisé en React) est en fait une interface pour visualiser de façon plus intuitive la simulation, en facilitant à la fois le déploiement et le test de multiples scénarios.

10 Ouvertures et Améliorations

Bien que l'approche CSP décrite réponde à de nombreux besoins, plusieurs pistes d'amélioration restent possibles :

- **Réallocation Dynamique des Tâches** : En cas de panne d'un robot ou d'arrivée impromptue d'une nouvelle tâche urgente, il serait intéressant de réoptimiser en cours d'exécution.
- **Optimisation Multicritère** : Élargir l'objectif au-delà du simple makespan, pour par exemple minimiser l'énergie totale consommée ou équilibrer la charge de travail entre robots.

— **Prise en Compte de Zones Interdites Temporaires** : Bloquer certains couloirs pendant un laps de temps (maintenance, présence humaine, etc.) et calculer des chemins alternatifs.

— **Scalabilité Accrue** : Adapter les heuristiques (par exemple, le *Large Neighborhood Search* ou la décomposition hiérarchique) afin de gérer un plus grand nombre de robots et de tâches sans explosion combinatoire.

11 Conclusion

Le problème de planification de tâches multi-robots dans un entrepôt implique la gestion simultanée de contraintes temporelles, spatiales et énergétiques. L'approche par programmation par contraintes (CSP), associée à des stratégies de recherche de chemin (A*), s'est révélée particulièrement adaptée pour fournir des solutions solides et flexibles, souvent supérieures aux méthodes purement heuristiques ou exhaustives dans des contextes de taille moyenne à grande.

Les travaux futurs porteront sur le renforcement de la réactivité du système (réaffectation dynamique), l'enrichissement des critères d'optimisation et l'intégration de méthodes plus avancées pour améliorer l'échelabilité.

12 Bibliographie & Sources

[1] A Two-Objective ILP Model of OP-MATSP for the Multi-Robot Task Assignment in an Intelligent Warehouse.

[2] Multi-Constrained Voronoi-Based Task Allocator for Smart-Warehouses.

[3] Scheduling in Python with Constraint Programming.

[4] Employee Scheduling.

[5] Assignment with Teams of Workers.

[6] Modeling Multi-Agent Collision in Google CP-SAT Solver.

[7] Entreprise qui propose des solutions de robots intelligents.