Configuration de flottes de robots hétérogènes pour la planification de mission: une approche basée programmation par contraintes

Lucas Ligny[1, 2], Gauthier Picard[1], Stéphanie Roussel[1], Elise Vareilles[2]

[1] DTIS, ONERA, Université de Toulouse, France
[2] Centre Génie Industriel, Université Toulouse, IMT Mines Albi, Albi, France

{lucas.ligny,gauthier.picard,stephanie.roussel}@onera.fr elise.vareilles@mines-albi.fr

Résumé

Avec l'autonomie croissante des robots et la variété des tâches qu'ils sont désormais en mesure de réaliser, ceux-ci sont de plus en plus souvent utilisés dans le cadre de missions complexes et critiques. Cet article s'intéresse au problème de la configuration des flottes de robots pour la planification de mission. Dans nos travaux, la planification d'une mission, avant sa phase d'exécution, repose sur la résolution de trois problèmes de configuration intrinsèquement liés : le problème de configuration d'agents, noté ACP, le problème de configuration de la flotte d'agents, noté FCP et le problème de configuration du plan de mission, noté PCP. Dans cet article, nous considérons la planification de mission comme un seul et unique problème, composé de trois sous-problèmes interdépendants. Un modèle de programmation par contraintes (PPC) est proposé pour le problème global. Notre modélisation est évaluée sur le solveur IBM ILOG CP Optimizer et sur soixante-quatre instances de type Recherche & Sauvetage générées artificiellement.

Mots-clés

Configuration, flotte hétérogène, PPC, Search & Rescue.

Abstract

With the increasing autonomy of robots and the variety of tasks they are now able to perform, they are increasingly used in complex and critical missions. This paper focuses on the problem of configuring robot fleets for mission planning. In our work, planning a mission, before its execution phase, is based on solving three intrinsically linked configuration problems : the agent configuration problem, denoted ACP, the agent fleet configuration problem, denoted FCP and the mission plan configuration problem, denoted PCP. In this paper, we consider mission planning as a single problem, composed of three interdependent subproblems. A constraint programming (CP) model is proposed for the global problem. Our model is evaluated on the IBM ILOG CP Optimizer solver and on sixty-four artificially generated Search & Rescue instances.

Keywords

Configuration, heterogeneous fleet, CP, Search & Rescue.

1 Introduction

Aujourd'hui, les véhicules autonomes ou *robots* sont de plus en plus impliqués au sein de missions complexes (*p. ex.* surveillance de zones, livraison de colis, logistique d'entrepôts, *etc.*). Cette présence s'explique notamment par l'hétérogénéité de ces robots, provenant d'une part de leur disposition à accéder à des zones différentes via des routes aériennes, terrestres ou maritimes, et d'autre part de leur capacité à manier plusieurs charges utiles différentes, les rendant extrêmement pratiques et efficaces dans l'exécution de missions complexes. La performance et la robustesse de flottes constituées de robots sont des enjeux importants pour le succès d'une mission. Ainsi, le problème de la configuration d'une flotte de robots est essentiel, notamment au sein de missions pour lesquelles le budget alloué ou la durée d'exécution sont restreints. Par exemple, dans le cadre d'une mission *Recherche & Sauvetage*, le temps mis pour venir en aide à chacune des personnes impliquées dans la catastrophe est critique.

La configuration d'une flotte de robots peut se définir à plusieurs niveaux. Dans le cas où la flotte est déjà constituée, le premier niveau de configuration correspond à l'affectation de la succession des tâches à effectuer pour chacun des robots. Il s'agit d'un problème d'optimisation nommé *problème de configuration du plan* (PCP). C'est à ce niveau que la performance et la robustesse de la flotte sont évaluées. Cependant, lorsque les missions nécessitent de traiter un nombre croissant de tâches ou lorsque des robots sont devenus obsolètes, la composition de la flotte doit être remise en question. Ces considérations sont prises en compte par le *problème de configuration de la flotte* (FCP), qui inclut le dimensionnement et l'organisation de la flotte de robots pour lui permettre de s'adapter de la meilleure façon possible aux différentes tailles et complexités de mission. Enfin, il peut être nécessaire d'acquérir de nouvelles compétences afin d'effectuer des tâches plus variées. Le *problème de configuration des agents* (ACP) consiste à personnaliser chaque robot en choisissant ses équipements dans un catalogue, lui conférant ainsi des compétences propres.

Ces trois problèmes de configuration, *problème de configuration du plan* (PCP), *problème de configuration de la flotte* (FCP) et *problème de configuration des agents* (ACP), sont

interconnectés, c'est-à-dire que chaque décision prise dans le cadre de l'étude de l'un de ces problèmes a un impact direct sur l'ensemble des autres décisions. Par exemple, une flotte de robots mal dimensionnée peut être à l'origine de l'échec d'une mission et, à l'inverse, une mission peut imposer des configurations de flotte ou de robots spécifiques. Ainsi, afin d'éviter des cycles itératifs de configuration pour atteindre les objectifs de performance et de robustesse désirés, une approche holistique doit être envisagée. L'objectif de cet article est précisément d'étudier ces problèmes de configuration par une telle approche. Les contributions de cet article sont les suivantes.

1. le problème incluant les trois niveaux de configuration, nommé *problème global de configuration* (GCP), est formellement défini ;
2. un modèle de programmation par contraintes (PPC) est établi pour ce problème ;
3. de nouvelles instances académiques sont proposées ;
4. l'approche PPC est évaluée sur les nouvelles instances avec le solveur IBM ILOG CP Optimizer [10].

La structure de l'article est la suivante. Un état de l'art sur les problèmes de configuration de flottes d'agents hétérogènes est établi dans la section 2. Nous décrivons le problème global dans la section 3, puis nous présentons le modèle PPC dans la section 4. Nos expérimentations et les résultats associés sont établis dans la section 5. Enfin, une conclusion et une discussion des perspectives sont proposées dans la section 6.

2 État de l'art

Cette section analyse les travaux connexes à nos problématiques de configuration.

2.1 Configuration du plan

Le problème PCP présente des aspects liés aux problèmes de tournées de véhicules (*vehicle routing*) et aux problèmes d'ordonnancement de tâches (*production scheduling*). Ces problèmes sont extensivement étudiés dans la littérature ; parmi les plus célèbres, nous pouvons nommer le Capacitated Vehicle Routing Problem with Time-Windows [19] et le Job Shop Problem [9].
Le problème PCP présente également des contraintes liées aux ressources, c'est-à-dire que certaines tâches ne peuvent être effectuées que par des ressources spécialisées. Ainsi, le problème PCP fait partie d'une classe de problèmes nommée Resource-Constrained Routing and Scheduling Problems, présentée dans une revue de littérature de Paraskevopoulos *et al.* [13]. Dans cette revue, deux variantes principales de ces problèmes sont étudiées : le Skill VRP [1] et le Technician Routing and Scheduling Problem [14]. Le Skill VRP consiste à minimiser les coûts de trajets sous contraintes de qualification de techniciens (certaines tâches nécessitent un certain niveau de qualification). Les modèles proposés pour ce problème sont des programmes linéaires en nombres entiers. Ce problème est une spécialisation du Site-Dependant Vehicle Routing Problem (SDVRP) étudié par Cordeau et Laporte dans [3]. Ce problème prend en compte de façon plus générale des contraintes de compatibilités entre les véhicules et les tâches à effectuer. Les instances de Cordeau et de Laporte ont été résolues par une recherche taboue.
Le Technician Routing and Scheduling Problem (TRSP) considère les visites d'un technicien dans des fenêtres de temps, nécessitant à la fois des compétences spécifiques mais également des outils et des pièces détachées. Les instances de TRSP sont résolues par une matheuristique basée sur la méthode Adaptative Large Neighborhood Search (ALNS) développée par Ropke et Pisinger dans [17].

2.2 Couplage avec les décisions d'investissement

De nombreuses applications nécessitent de remettre en cause le dimensionnement de la flotte en vue de certaines missions. Une telle application peut être celle d'une entreprise de transport public souhaitant renouveller sa flotte de véhicules, pour des considérations économiques ou écologiques. Yıldırım et Yıldız [22] proposent une méthode basée sur la génération de colonnes pour adresser le problème d'optimisation d'une flotte de bus et tirer parti des nouvelles technologies de recharge des batteries de bus électriques. Lemme *et al.* [11] évaluent par une programmation linéaire en nombres entiers le nombre de véhicules électriques à considérer dans une flotte d'un système de partage de voiture selon différents scénarios (*p. ex.* avec une contrainte plus ou moins forte sur les émissions de gaz à effet de serre selon la politique adoptée). Sayarshad et Tavakkoli-Moghaddam [18] proposent un modèle d'optimisation bi-niveau afin de dimensionner une flotte de wagons sous incertitudes de demandes. Ce modèle stochastique est résolu par la méthode de recuit simulé. Gamst et Pisinger [6] proposent différents types de décisions d'investissement pour le TRSP. Nous pouvons notamment citer l'évaluation de l'entraînement d'un technicien, le faisant ainsi monter en compétence, ou alors l'emploi d'un nouveau technicien. Une génération de colonnes est utilisée pour résoudre ce problème, en incluant les décisions d'investissement dans le problème maître, et les décisions d'affectation de tâches dans le sous-problème.

2.3 Application aux flottes de robots

Les trois niveaux de configuration pour une flotte de robots hétérogènes ont été identifiés par Pouré *et al.* [15]. Dans cet article, les auteurs proposent une formalisation des entrées et des sorties des trois problèmes de configuration. La résolution de ces problèmes est envisagée à la fois de façon séquentielle (la sortie de ACP vue comme entrée de FCP, et la sortie de FCP vue comme entrée de PCP) et itérative (l'évaluation de la performance obtenue dans PCP impacte les décisions prises dans ACP et FCP). Gigante *et al.* [8] proposent une approche basée sur la théorie des jeux pour configurer une flotte de drones aériens dans le but de réaliser des tâches d'observation de zones fixes. Cet article traite de nombreuses contraintes opérationnelles propres aux missions effectuées par des robots, telles que les contraintes de charge maximale, d'endurance (autonomie énergétique

du véhicule) ou encore de traficabilités (accessibilité à certaines zones). Notre contribution vis-à-vis de cet article est la proposition d'un modèle de programmation par contraintes pour résoudre le problème.

3 Description du problème

Dans cette section, nous décrivons les entrées et les sorties du problème GCP, ainsi que les postulats établis. Un exemple *fil rouge* d'une mission *Recherche & Sauvetage* est également introduit pour illustrer chacun des concepts.

3.1 Définitions et notations des entrées

Description de l'instance. Une instance de GCP est un tuple $(\mathcal{P}, \mathcal{E}, \mathcal{T}, \Pi, H, C, b)$ tel que :

- $\mathcal{P}$ est l'ensemble des plateformes disponibles dans le stock. Une plateforme est un robot non-équipé ;
- $\mathcal{E}$ est l'ensemble des équipements disponibles dans le stock. Un équipement est une charge utile fournissant une compétence spécifique ;
- $\mathcal{T}$ est l'ensemble des tâches devant être réalisées pour réussir la mission. Chaque tâche requiert une compétence spécifique fournie par un équipement ;
- $\Pi \subset \mathcal{T} \times \mathcal{T}$ est l'ensemble des précédences à respecter entre les tâches. Une précédence est une relation entre deux tâches, imposant la première tâche à être effectuée avant la seconde tâche ;
- $H \in \mathbb{N}^*$ est l'horizon temporel considéré pour la mission ;
- $C \in \mathbb{R}^+$ est le budget maximal alloué à la mission ;
- b est la base, *i.e.* le point de départ et de retour pour les robots de la flotte.

Dans les paragraphes suivants, nous décrivons les caractéristiques plus précises de chacun de ces éléments.

Description des plateformes. Pour chaque plateforme $p \in \mathcal{P}$:

- $c_p \in \mathbb{R}^+$ est son coût ;
- $W_p \in \mathbb{R}^+$ est la charge maximale qu'elle peut embarquer ;
- $d_p : \mathcal{T}_p \cup \{b\} \times \mathcal{T}_p \cup \{b\} \to \mathbb{N}$ est la fonction qui renvoie la durée nécessaire à la plateforme p pour se déplacer d'une opération à une autre. Une opération est soit une tâche atteignable par la plateforme p, soit un départ ou un retour à sa base ;
- $\mathcal{T}_p \subseteq \mathcal{T}$ est l'ensemble des tâches atteignables par la plateforme p.

Description des équipements. Pour chaque équipement $e \in \mathcal{E}$:

- $c_e \in \mathbb{R}^+$ est son coût ;
- $w_e \in \mathbb{R}^+$ est son poids ;
- $\mathcal{P}_e \subseteq \mathcal{P}$ est l'ensemble des plateformes compatibles avec l'équipement e ;
- $\mathcal{E}_e \subseteq \mathcal{E}$ est l'ensemble des équipements compatibles avec l'équipement e.

Description des tâches. Pour chaque tâche $t \in \mathcal{T}$:

- $\delta_t \in \mathbb{N}^*$ est sa durée d'exécution ;
- $\mathcal{E}_t \subseteq \mathcal{E}$ est l'ensemble des équipements possédant la compétence requise par la tâche t.

p	c_p	W_p	d_p	$\mathcal{T}_p$
🚁	30	2	$20 \cdot f$	$\mathcal{T}$
🚙	100	10	$50 \cdot f$	$\mathcal{T}$

TABLE 1 – Caractéristiques des plateformes de l'instance *fil rouge*. Les valeurs de la fonction f sont définies dans la table 2.

f	🏠	🖼	⚙	🧰
🏠	0	1	1	1
🖼	1	0	0	1
⚙	1	0	0	1
🧰	1	1	1	0

TABLE 2 – Valeurs de la fonction de distance f

Description des précédences. Pour chaque précédence $(t_1, t_2) \in \Pi$, t_1 représente la tâche devant être effectuée avant la tâche t_2.

Exemple 1 (instance *fil rouge*) *Nous considérons la mission Recherche & Sauvetage suivante.*

- *l'ensemble des plateformes disponibles $\mathcal{P}$ est constitué d'un drone 🚁 et d'un rover 🚙. Leurs caractéristiques sont présentées dans la table 1. Ces plateformes peuvent accéder aux mêmes tâches mais se déplacent à une vitesse différente. Ainsi, les valeurs des fonctions $d_{🚁}$ et $d_{🚙}$ sont déterminées à partir d'une fonction de distance $f : \mathcal{T} \cup \{b\} \times \mathcal{T} \cup \{b\} \to \mathbb{N}$ définie dans la table 2 ;*
- *l'ensemble des équipements disponibles $\mathcal{E}$ est constitué d'une caméra 📷, d'un bras mécanique 🔧, d'un parachute 🪂 et d'une remorque 🚚. Leurs caractéristiques sont présentées dans la table 3 ;*
- *l'ensemble des tâches à réaliser $\mathcal{T}$ est constitué d'une observation 🖼, d'une réparation ⚙ et d'une livraison 🧰. Leurs caractéristiques sont présentées dans la table 4 ;*
- *l'ensemble des précédences Π est constitué d'une unique relation imposant que la tâche d'observation 🖼 soit effectuée avant la tâche de réparation ⚙. Ainsi, $\Pi = \{(🖼, ⚙)\}$;*

e	c_e	w_e	$\mathcal{P}_e$	$\mathcal{E}_e$
📷	15	1	$\mathcal{P}$	$\mathcal{E}$
🔧	25	3	{🚙}	$\mathcal{E}$
🪂	1	1	{🚁}	$\mathcal{E}$
🚚	10	6	{🚙}	$\mathcal{E}$

TABLE 3 – Caractéristiques des équipements de l'instance *fil rouge*

t	δ_t	$\mathcal{E}_t$
🖼	30	{📷}
⚙	60	{🔧}
📦	12	{🪂, 🚚}

TABLE 4 – Caractéristiques des tâches de l'instance *fil rouge*

— $H = 360$;
— $C = 180$;
— *la base* b *est l'entrepôt* 🏭.

3.2 Définitions et notations des sorties

Une solution pour une instance de GCP est un couple $(\mathcal{A}, \mathcal{R})$, avec $\mathcal{A}$ représentant l'ensemble des agents de la flotte configurée pour la mission et $\mathcal{R}$ l'ensemble des routes de chacun des agents.

Description des agents. Pour chaque agent $a \in \mathcal{A}$:
— $p_a \in \mathcal{P}$ est la plateforme utilisée par l'agent a ;
— $\mathcal{E}_a \subseteq \mathcal{E}$ est l'ensemble des équipements associés à l'agent a.

Description des routes. Pour chaque route $r \in \mathcal{R}$:
— $agent_r \in \mathcal{A}$ est l'agent utilisé pour parcourir la route r ;
— $\mathbf{V}_r$ est la séquence des visites effectuées sur la route r. Pour tout $i \in [\![1, |\mathbf{V}_r|]\!]$ [1] , une visite $v_r^i \in \mathbf{V}_r$ est un couple (o_r^i, h_r^i) tel que :
— o_r^i est l'opération réalisée pendant la visite v_r^i vérifiant :

$$\begin{cases} o_r^i = b & \text{si } i = 1 \text{ ou } i = |\mathbf{V}_r| \\ o_r^i \in \mathcal{T} & \text{si } i \in [\![2, |\mathbf{V}_r - 1|]\!] \end{cases};$$

— $h_r^i \in [\![0, H]\!]$ est la date à laquelle la visite v_r^i est effectuée.

Une solution est *faisable* si les agents, formés d'associations valides entre les plateformes et les équipements, sont capables de réaliser l'intégralité des tâches dans le respect des contraintes opérationnelles (budget, précédences, compatibilités, *etc.*).
Une solution est *optimale* si elle est faisable et si elle minimise le *makespan*, correspondant à la date de retour à la base du dernier agent.

Exemple 2 (solutions pour l'instance *fil rouge*) *Une solution faisable pour l'instance fil rouge est la suivante :*
— $\mathcal{A} = \{a_1, a_2\}$ *avec :*
— $p_{a_1} =$ 🚁 *et* $p_{a_2} =$ 🚜 ;
— $\mathcal{E}_{a_1} = \{$📷$\}$ *et* $\mathcal{E}_{a_2} = \{$🔧, 🚚$\}$;
— $\mathcal{R} = \{r_1, r_2\}$ *avec :*
— $agent_{r_1} = a_1$ *et* $agent_{r_2} = a_2$;
— $\mathbf{V}_{r_1} = [($🏭$, 0), ($🖼$, 20), ($📦$, 70), ($🏭$, 222)]$.
— $\mathbf{V}_{r_2} = [($🏭$, 0), ($⚙$, 50), ($📦$, 160), ($🏭$, 222)]$.

1. La notation $[\![\cdot, \cdot]\!]$ désigne l'ensemble des entiers entre deux bornes incluses.

Les dates auxquelles sont effectuées les tâches sont calculées au plus tôt. Ainsi, l'agent a_1 *commence l'observation* 🖼 *à la date 20, la termine à la date 50 et retourne à l'entrepôt* 🏭 *à la date 70. L'agent* a_2 *commence la réparation* ⚙ *à la date 50 (la contrainte de précédence étant bien respectée), la termine à la date 110, puis enchaîne sur la livraison* 📦 *à la date 160, qui se termine à la date 172. La date de retour à l'entrepôt* 🏭 *de l'agent* a_2 *est 222, qui est donc la valeur du makespan associé à cette solution. Le budget dépensé est 180, ce qui satisfait la contrainte de budget maximal.*
Cette solution n'est pas optimale, il est en effet plus performant d'affecter la livraison 📦 *à l'agent* a_1 *car le drone* 🚁 *est plus rapide. La solution optimale pour l'instance fil rouge est la suivante :*
— $\mathcal{A} = \{a_1, a_2\}$ *avec :*
— $p_{a_1} =$ 🚁 *et* $p_{a_2} =$ 🚜 ;
— $\mathcal{E}_{a_1} = \{$📷, 🪂$\}$ *et* $\mathcal{E}_{a_2} = \{$🔧$\}$;
— $\mathcal{R} = \{r_1, r_2\}$ *avec :*
— $agent_{r_1} = a_1$ *et* $agent_{r_2} = a_2$;
— $\mathbf{V}_{r_1} = [($🏭$, 0), ($🖼$, 20), ($📦$, 70), ($🏭$, 102)]$;
— $\mathbf{V}_{r_2} = [($🏭$, 0), ($⚙$, 50), ($🏭$, 160)]$.
Il est important de noter que l'ordre des opérations effectuées par l'agent a_1 *est important : si la livraison* 📦 *est effectuée avant l'observation* 🖼*, alors celle-ci ne sera pas effectuée avant la date 82, ce qui dégrade le makespan de 32 unités de temps en raison de la contrainte de précédence (l'agent* a_2 *doit attendre la fin de l'observation* 🖼 *réalisée par l'agent* a_1 *avant de commencer la réparation* ⚙*).*

3.3 Postulats

Les postulats établis pour l'étude de ce problème sont les suivants.

1. les relations de compatibilité entre les équipements doivent être cohérentes entre elles, c'est-à-dire que $\forall (e_1, e_2) \in \mathcal{E}^2$:
$$e_2 \in \mathcal{E}_{e_1} \iff e_1 \in \mathcal{E}_{e_2};$$
2. les précédences entre les tâches forment un graphe dirigé acyclique ;
3. les agents constitués sont libres de se déplacer sans risque de conflits avec les autres agents. Ainsi, dans le cadre de la minimisation de la durée de la mission, les agents ont toujours la possibilité d'emprunter le plus court chemin entre les tâches ;
4. les agents présents dans la solution sont utiles, c'est-à-dire que chaque agent doit effectuer au moins une tâche ;
5. les agents ne peuvent pas coopérer pour la réalisation des tâches. Les tâches n'ont donc besoin que d'un seul agent pour être réalisées et aucune synchronisation d'agents n'est à considérer ;
6. la mission est statique, c'est-à-dire que l'environnement et les tâches sont parfaitement connus à l'avance ;
7. les plateformes et les équipements sont sans défauts. La robustesse de la configuration n'est donc pas évaluée dans cet article.

4 Modèle PPC

Dans cet article, nous utilisons la formulation à base de variables de type *intervalle* du langage de modélisation OPL [20]. Une variable d'intervalle permet de représenter une tâche ou une activité par sa date de début, sa date de fin, sa durée et sa présence obligatoire ou facultative dans la solution. Ces informations sont respectivement accessibles via les opérateurs startOf, endOf, lengthOf et presenceOf. Il est possible de préciser la valeur de chaque opérateur lorsque l'intervalle optionnel sur lequel il s'applique est absent. Par exemple, $\mathsf{startOf}(\mathbf{itv}, 0)$ renvoie la date de début de l'intervalle $\mathbf{itv}$ si celui-ci est présent, et 0 s'il est absent.

4.1 Variables de décision

Pour chaque tâche $t \in \mathcal{T}$, nous considérons la variable d'intervalle $\mathbf{itv}_t \in [\![0, H]\!]$ représentant l'exécution de la tâche t. Sa présence est obligatoire et sa durée est constante et égale à δ_t. Pour chaque plateforme $p \in \mathcal{P}$ et chaque tâche $t \in \mathcal{T}_p$, nous considérons la variable d'intervalle $\mathbf{itv}_{t,p} \in [\![0, H]\!]$ représentant l'exécution de la tâche t par la plateforme p. Sa présence est facultative et sa durée est constante et égale à δ_t. Pour chaque équipement $e \in \mathcal{E}$, nous considérons la variable entière $\mathbf{pf}_e \in \mathcal{P}_e \cup \{p^\emptyset\}$ représentant la plateforme associée à l'équipement e. La notation $p^\emptyset$ représente la plateforme fictive des équipements inutilisés. Pour chaque plateforme $p \in \mathcal{P}$, nous considérons les variables suivantes.

- $\mathbf{itv}_p^\alpha \in [\![0, H]\!]$ est la variable d'intervalle représentant la tâche de départ de la base. Sa présence est facultative et sa durée est nulle ;
- $\mathbf{itv}_p^\Omega \in [\![0, H]\!]$ est la variable d'intervalle représentant représentant la tâche d'arrivée à la base. Sa présence est facultative et sa durée est nulle.

4.2 Expressions intermédiaires

Pour chaque plateforme $p \in \mathcal{P}$, nous considérons la variable booléenne $\mathbf{u}_p \in \{0, 1\}$ indiquant si la plateforme p est utilisée. Cette variable est définie dans l'équation (1).

$$\mathbf{u}_p = \left(\left[\sum_{t \in \mathcal{T}_p} \mathsf{presenceOf}(\mathbf{itv}_{t,p}) \right] > 0 \right) \tag{1}$$

Pour chaque équipement $e \in \mathcal{E}$, nous considérons la variable booléenne $\mathbf{y}_e \in \{0, 1\}$ indiquant si l'équipement e est utilisé. Cette variable est définie dans l'équation (2).

$$\mathbf{y}_e = \left(\mathbf{pf}_e \neq p^\emptyset \right) \tag{2}$$

Pour chaque équipement $e \in \mathcal{E}$ et pour chaque plateforme $p \in \mathcal{P}$, nous considérons la variable booléenne $\mathbf{x}_{e,p} \in \{0, 1\}$ indiquant si l'équipement e est associé à la plateforme p. Cette variable est définie dans l'équation (3).

$$\mathbf{x}_{e,p} = (\mathbf{pf}_e = p) \tag{3}$$

Il est important de noter que les définitions de $\mathbf{u}_p$, $\mathbf{y}_e$ et $\mathbf{x}_{e,p}$ sont permises grâce au mécanisme de réification de contraintes, qui associe une variable booléenne à la satisfaction d'une contrainte.

4.3 Contraintes

Grâce aux notations définies précédemment, nous définissons les contraintes de notre modèle.
La contrainte de respect du budget maximal est exprimée dans l'équation (4).

$$\sum_{p \in \mathcal{P}} c_p \mathbf{u}_p + \sum_{e \in \mathcal{E}} c_e \mathbf{y}_e \leq C \tag{4}$$

La contrainte de charge maximale des plateformes est exprimée dans l'équation (5). Le respect des durées de déplacement entre les tâches d'une même plateforme est garanti par l'équation (6) via l'opérateur noOverlap du langage OPL. Cet opérateur prend en entrée un ensemble d'intervalles pour lesquels il impose le non-chevauchement, ainsi que la fonction des temps de transition entre les tâches qui doivent être respectés.

$$\forall p \in \mathcal{P}, \qquad \sum_{e \in \mathcal{E}} w_e \mathbf{x}_{e,p} \leq W_p \tag{5}$$

$$\forall p \in \mathcal{P}, \qquad \mathsf{noOverlap}\left(\{\mathbf{itv}_{t,p}\}_{t \in \mathcal{T}_p}, d_p \right) \tag{6}$$

Les contraintes (7) empêchent deux équipements incompatibles d'être associés à la même plateforme.

$$\forall e_1 \in \mathcal{E}, \forall e_2 \in \mathcal{E} \setminus \mathcal{E}_{e_1}, \qquad \mathbf{y}_{e_1} \wedge \mathbf{y}_{e_2} \rightarrow \left(\mathbf{pf}_{e_1} \neq \mathbf{pf}_{e_2} \right) \tag{7}$$

Les contraintes d'association d'un unique intervalle optionnel à chaque tâche $t \in \mathcal{T}$ sont exprimées grâce à l'opérateur alternative du langage OPL dans l'équation (8). Cet opérateur prend en entrée un intervalle et un ensemble d'intervalles optionnels. Si l'intervalle donné en entrée est présent, alors exactement un des intervalles optionnels donnés en entrée est présent et leurs dates de débuts, dates de fins et durées sont égales.

$$\forall t \in \mathcal{T} \qquad \mathsf{alternative}\left(\mathbf{itv}_t, \{\mathbf{itv}_{t,p}\}_{p \in \mathcal{P}_t} \right) \tag{8}$$

Les contraintes de précédence entre les tâches sont gérées par l'opérateur endBeforeStart du langage OPL dans l'équation (9). Cet opérateur prend en entrée deux intervalles et impose que le premier intervalle se termine avant le début du second intervalle, avec un délai éventuel à respecter. Cette contrainte est levée si l'un des deux intervalles est absent.

$$\forall (t_1, t_2) \in \Pi \qquad \mathsf{endBeforeStart}(\mathbf{itv}_{t_1}, \mathbf{itv}_{t_2}) \tag{9}$$

Les contraintes (10) et (11) assurent que toutes les tâches d'une plateforme s'effectuent après son départ de la base.

$$\forall p \in \mathcal{P}, \forall t \in \mathcal{T}_p, \qquad \mathsf{endBeforeStart}\left(\mathbf{itv}_p^\alpha, \mathbf{itv}_{t,p}, d_p(b, t) \right) \tag{10}$$

$$\forall p \in \mathcal{P}, \qquad \mathsf{presenceOf}\left(\mathbf{itv}_p^\alpha \right) \geq \mathbf{u}_p \tag{11}$$

Les contraintes (12) et (13) assurent que toutes les tâches d'une plateforme s'effectuent avant son retour à la base.

$$\forall p \in \mathcal{P}, \forall t \in \mathcal{T}_p, \qquad \mathsf{endBeforeStart}\left(\mathbf{itv}_{t,p}, \mathbf{itv}_p^\Omega, d_p(t, b) \right) \tag{12}$$

$$\forall p \in \mathcal{P}, \qquad \mathsf{presenceOf}\left(\mathbf{itv}_p^\Omega \right) \geq \mathbf{u}_p \tag{13}$$

Enfin, les contraintes de compatibilité entre les agents et les tâches, issues des compétences apportées par les équipements et des requêtes des tâches, sont exprimées dans l'équation (14).

$$\forall p \in \mathcal{P}, \forall t \in \mathcal{T}_p, \qquad \sum_{e \in \mathcal{E}_t} \mathbf{x}_{e,p} \geq \mathsf{presenceOf}(\mathbf{itv}_{t,p}) \quad (14)$$

4.4 Formulation du problème

Le critère retenu est la minimisation du *makespan* exprimé dans l'équation (15). La formulation du problème est la suivante :

$$\begin{aligned} &\textit{minimize} && \max_{p \in \mathcal{P}} \mathsf{endOf}\left(\mathbf{itv}_p^{\Omega}\right) && (15) \\ &s.t. && (4) - (14) \end{aligned}$$

5 Expérimentations

5.1 Génération d'instances

Nous avons généré un ensemble d'instances afin de tester notre approche. Dans l'instance *fil rouge* présentée dans la section 3.1, tous les éléments du stock ont des caractéristiques différentes. Cependant, dans les instances de nos expérimentations, certains éléments sont identiques et ont donc les mêmes caractéristiques. Dans cette section, nous utilisons une notation unique pour désigner les éléments d'un même type (*p. ex.* représente un drone quelconque du stock disponible). De façon similaire, nous utilisons une notation unique pour désigner les tâches du même type, c'est-à-dire de même durée et nécessitant les mêmes compétences.

Méthodologie. Pour générer ces instances, nous avons créé un squelette d'instance générique pour lequel certains paramètres sont fixés, puis nous avons ajouté de la variabilité à ce squelette afin d'enrichir notre jeu de données.

Paramètres généraux. Les principales caractéristiques des instances générées sont les suivantes.

- les caractéristiques des plateformes sont présentées dans la table 5. Pour chaque platforme $p \in \mathcal{P}$, les valeurs de la fonction d_p sont déterminées à partir d'une fonction de distance $f_p : \mathcal{T}_p \cup \{b\} \times \mathcal{T}_p \cup \{b\} \rightarrow \mathbb{R}^+$, qui représente la distance que chaque plateforme doit parcourir pour passer d'une tâche à une autre, et dont le calcul est détaillé dans la suite de cette section [2] ;
- les caractéristiques des équipements sont présentées dans la table 6. Par rapport à l'instance *fil rouge*, un nouveau type d'équipement est disponible dans le stock : l'émetteur ;
- les caractéristiques des tâches sont présentées dans la table 7. Par rapport à l'instance *fil rouge*, un nouveau type de tâche peut être réalisé : l'activation de balises ;
- l'ensemble des précédences est vide ;
- le budget maximal alloué est choisi dans $C^{min} \times \{1, 2, 3, 4\}$ où C^{min} représente le budget minimal pour effectuer la mission. Notons que ce budget minimal est obtenu par énumération des associations entre les types de plateformes et les types d'équipements, ce qui est permis par la faible combinatoire de notre étude sur ces grandeurs.
- la base est l'entrepôt .

p	c_p	W_p	d_p	$\mathcal{T}_p$
	3000	2	$20 \cdot f_{}$	$\mathcal{T}$
	10000	10	$50 \cdot f_{}$	$\mathcal{T}$

TABLE 5 – Caractéristiques des plateformes des instances générées

e	c_e	w_e	$\mathcal{P}_e$	$\mathcal{E}_e$
	1500	1	$\mathcal{P}$	$\mathcal{E}$
	300	1	$\{\}$	$\mathcal{E} \setminus \{\}$
	2500	3	$\{\}$	$\mathcal{E}$
	100	1	$\{\}$	$\mathcal{E} \setminus \{\}$
	1000	6	$\{\}$	$\mathcal{E}$

TABLE 6 – Caractéristiques des équipements des instances générées

t	δ_t	$\mathcal{E}_t$
	300	$\{\}$
	600	$\{\}$
	600	$\{\}$
	120	$\{,\}$

TABLE 7 – Caractéristiques des tâches des instances générées

Nombre de tâches. Le nombre de tâches à effectuer est choisi dans $\{4, 8, 20, 48\}$. Les proportions des types de tâches sont présentées dans la colonne "Proportion" de la table 8.

Dimensionnement du stock. Le stock de plateformes est considéré illimité. En pratique, une plateforme par tâche compatible est considérée dans le stock disponible. La distribution des équipements par rapport aux compétences requises par les tâches est choisie dans $\{same, opposite\}$. La distribution $same$ correspond à un nombre d'équipements en adéquation avec la proportion de chaque tâche, alors que la distribution $opposite$ présente un nombre d'équipements en excès pour les tâches rares, et en défaut pour les tâches répandues. Les stocks d'équipements, définis en fonction de leur distribution, sont présentés dans la table 8.

2. Les valeurs de d_p sont arrondies à l'entier supérieur.

t	Proportion	e	Stock ($same$)	Stock ($opp.$)
	50%		9	1
	15%		5	9
	15%		5	9
	20%		3	3
			3	3

TABLE 8 – Proportion des types de tâches et stock d'équipements dans les instances générées

Localisation des opérations. Les tâches à réaliser et la base sont localisées sur un graphe. L'ensemble des nœuds de ce graphe est extrait des 51 nœuds de l'instance CMT01 proposée dans le cadre de l'étude du Capacitated Vehicle Routing Problem (CVRP) dans [2]. Pour chaque instance, $|\mathcal{T}| + 1$ nœuds sont extraits afin de permettre la répartition des tâches et de la base sur des nœuds différents. La position de la base est choisie dans $\{side, center\}$. La position $side$ correspond au nœud du graphe le plus éloigné de tous les autres, alors que la position $center$ correspond au nœud le plus central. Les types de tâches sont répartis aléatoirement entre les nœuds restants.

Calcul des durées de déplacement pour . Nous souhaitons qu'un drone puisse naviguer librement entre les nœuds en considérant le graphe complet. Ainsi, les valeurs de $f_{}$ pour chaque couple d'opérations sont égales à la distance euclidienne entre les nœuds sur lesquels ces opérations sont localisées.

Calcul des durées de déplacement pour . Nous souhaitons également qu'un rover ne puisse naviguer qu'entre les nœuds proches en considérant un graphe planaire et connexe. Ce graphe est obtenu en appliquant une triangulation de Delaunay [4] aux nœuds de chaque instance. Dans ce graphe, la distance entre deux nœuds adjacents est égale à la distance euclidienne. Pour obtenir la distance entre deux nœuds non adjacents, il faut déterminer le plus court chemin qui les relie. Un tel chemin existe car le graphe est connexe. Nous utilisons l'algorithme de Floyd-Warshall [5, 21] afin de calculer ces plus courts chemins et donc définir la fonction de distance $f_{}$.

Calcul de l'horizon. Il est nécessaire de définir l'horizon temporel H afin de borner les variables d'intervalle du modèle. Dans ce but, nous définissons dans l'équation (16) la durée maximale de parcours d'une arête du graphe par une plateforme quelconque :

$$D^{max} = \max_{p \in \mathcal{P}} d_p \tag{16}$$

Ainsi, en considérant une unique plateforme qui effectuerait l'intégralité des tâches et se déplacerait vers, depuis et entre chaque opération avec la durée D^{max}, nous obtenons une borne supérieure pour les variables d'intervalle, que nous prenons comme définition de l'horizon dans l'équation (17).

$$H = \left(\sum_{t \in \mathcal{T}} \delta_t \right) + (|\mathcal{T}| + 1)\, D^{max} \tag{17}$$

Nommage des instances. La convention choisie pour les noms des instances est "t{*nTasks*}–c{*budget*}–{*distrib*}–{*base*}" avec :

- *nTasks* représente le nombre de tâches à effectuer : "4", "8", "20" ou "48" ;
- *budget* représente le facteur associé au budget maximal alloué : "1", "2", "3" ou "4" ;
- *distrib* représente la distribution des équipements : "same" ou "opp" (pour une distribution *opposite*) ;
- *base* représente la position de la base : "side" ou "center".

Ainsi, un ensemble de $4 \times 4 \times 2 \times 2 = 64$ instances ont été générées à partir de ces paramètres variables. Ces instances sont disponibles dans [12].

5.2 Environnement de test

Toutes nos expérimentations ont été réalisées sous Windows Subsystem for Linux 2 (WSL2 version 2.2.4, distribution Ubuntu 24.04.1 LTS et architecture 64-bits) avec un processeur Intel Core i5-1345U, 6 cœurs (12 threads) et 8 Go de RAM. Le modèle a été écrit et testé avec Java 23 et IBM ILOG CP Optimizer 22.1.2.

5.3 Résultats

Le modèle défini dans la section 4 utilise le formalisme du langage OPL. De ce fait, aucune traduction des variables de décision, des contraintes ou des critères n'est requise pour l'utilisation de CP Optimizer. Chaque résolution est paramétrée avec une limite de temps égale à 60 secondes.
Pour des raisons de place, nous ne présentons pas les résultats des 64 instances, uniquement les 16 instances avec la distribution des équipements $same$ et la position de la base $side$ (instances $same$-$side$), et les 16 instances avec la distribution des équipements $opposite$ et la position de la base $center$ (instances $opposite$-$center$). Les résultats des instances $same$-$side$ sont détaillés dans la table 9. Nous observons que la résolution converge en quelques secondes seulement sur les instances allant jusque 20 tâches. Pour les instances comprenant 48 tâches (et pour l'instance t20-c3-same-side), l'optimalité n'a pas pu être prouvée en 60 secondes et aucun gap d'optimalité n'a pas être obtenu, ce qui montre les limites de la modélisation actuelle. Une autre modélisation, non présentée dans cet article, a été développée afin de renforcer la confiance en ces résultats. Cette modélisation, basée sur des variables de décision ensemblistes, a été implémentée et testée avec le solveur Hexaly Optimizer [7]. Les valeurs de *makespan* obtenues avec Hexaly Optimizer 13.5 en 60 secondes sont identiques à celles présentées dans la table 9. Le choix de Hexaly Optimizer pour cette validation est justifié par ses excellentes performances pour la résolution de problèmes de tournées de véhicules.
Les résultats des instances $opposite$-$center$ sont détaillés dans la table 10. Similairement aux résultats de la table 9,

Instance	Makespan	Opt.	Time (s)
t4-c1-same-side	4480	*	0.02
t4-c2-same-side	3524	*	0.03
t4-c3-same-side	3524	*	0.03
t4-c4-same-side	3524	*	0.03
t8-c1-same-side	4890	*	0.15
t8-c2-same-side	3814	*	0.07
t8-c3-same-side	3814	*	0.07
t8-c4-same-side	3814	*	0.16
t20-c1-same-side	7765	*	0.35
t20-c2-same-side	5459	*	1.34
t20-c3-same-side	4654		60.00
t20-c4-same-side	4654	*	1.05
t48-c1-same-side	14838		60.00
t48-c2-same-side	10389		60.00
t48-c3-same-side	7426		60.00
t48-c4-same-side	6606		60.00

TABLE 9 – Valeurs de *makespan* pour les instances *same-side*. Un astérisque (*) indique une preuve d'optimalité.

Instance	Makespan	Opt.	Time (s)
t4-c1-opp-center	2553	*	0.02
t4-c2-opp-center	2250	*	0.03
t4-c3-opp-center	2250	*	0.06
t4-c4-opp-center	2250	*	0.04
t8-c1-opp-center	4557	*	0.10
t8-c2-opp-center	2879	*	1.52
t8-c3-opp-center	2879	*	1.63
t8-c4-opp-center	2879	*	1.86
t20-c1-opp-center	7645	*	1.10
t20-c2-opp-center	4909		60.00
t20-c3-opp-center	4909		60.00
t20-c4-opp-center	4909		60.00
t48-c1-opp-center	14851		60.00
t48-c2-opp-center	10093		60.00
t48-c3-opp-center	10093		60.00
t48-c4-opp-center	10093		60.00

TABLE 10 – Valeurs de *makespan* pour les instances *opposite-center*. Un astérisque (*) indique une preuve d'optimalité.

la résolution converge rapidement sur les instances de petite taille. Nous observons cependant deux différences notables :

— Pour les instances de petite taille, les valeurs optimales de *makespan* sont plus basses dans la table 10 que dans la table 9. En effet, la base étant sur une position centrale, les tâches sont moins éloignées de la base et peuvent donc être effectuées plus rapidement par les agents lorsque le budget le permet.

— Pour les instances de grande taille, les valeurs optimales de *makespan* sont plus élevées dans la table 10 que dans la table 9. En effet, une seule caméra est disponible pour effectuer la totalité des observations, ce qui limite l'amélioration du *makespan* même lorsque le budget alloué est élevé.

Focus sur l'instance t20-c1-same-side. Dans l'instance t20-c1-same-side, les paramètres sélectionnés sont les suivants.

— 20 tâches doivent être réalisées ;
— le budget est le plus petit possible ;
— les distributions des tâches et des équipements sont en adéquation ;
— la base est excentrée.

Nous représentons sur la figure 1 la carte associée à la solution pour cette instance. Sur cette carte, dont les axes représentent les coordonnées des nœuds associés aux tâches, nous avons représenté les tâches à effectuer et les routes empruntées par les agents. La base est située en haut à droite de la carte aux coordonnées (62, 63). Les bandes grises reliant les tâches correspondent au graphe planaire d'un rover. Il est aisé de voir que cette configuration respecte la contrainte de budget minimal. En effet, seule une instance de chaque type de plateforme est utilisée :

— l'agent "Drone 1" est un drone équipé d'une caméra et d'un émetteur ;
— l'agent "Rover 1" est un rover équipé d'un bras mécanique et d'une remorque.

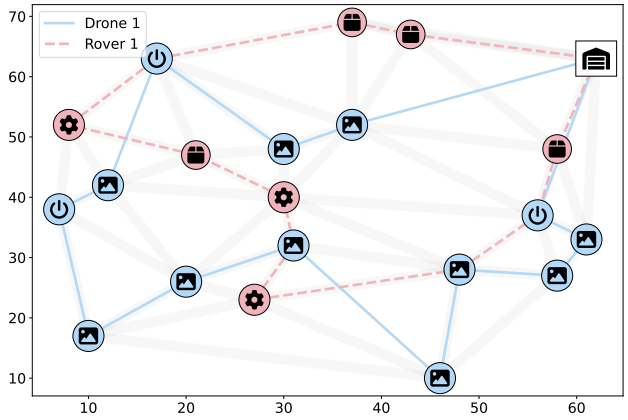

FIGURE 1 – Carte de la solution pour l'instance t20-c1-same-side

Par ailleurs, toujours sous contrainte de budget minimal, il n'est pas possible d'écarter définitivement un type de plateforme. En effet, puisqu'un drone ne peut pas être équipé d'un émetteur et d'un parachute simultanément pour des raisons de compatibilité, alors il est nécessaire d'utiliser un rover afin d'effectuer la livraison.

Le diagramme de Gantt de la solution pour cette même instance est représenté sur la figure 2. Sur ce diagramme, on observe que la durée planifiée la plus longue correspond à la route du rover. Ce résultat est attendu puisque le rover se déplace significativement plus lentement que le drone. On observe par ailleurs que les observations sont effectuées par le drone, ce qui est également attendu puisque celles-ci sont en plus grand nombre.

Notons ici que le drone peut effectuer ses tâches moins optimalement et sans impact sur le *makespan*. Pour une

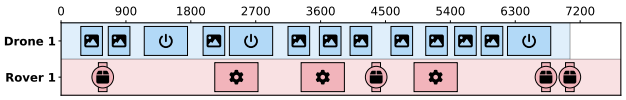

FIGURE 2 – Diagramme de Gantt de la solution pour l'instance t20-c1-same-side

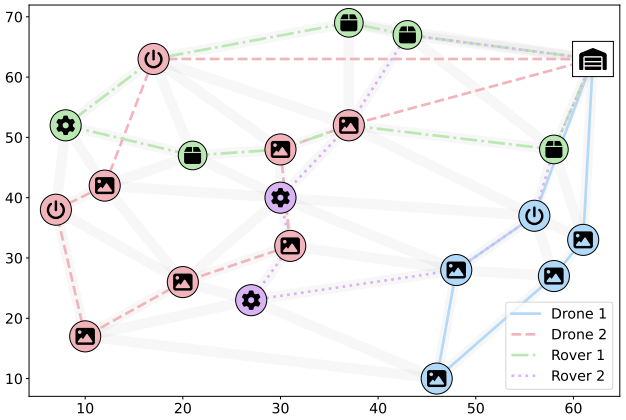

FIGURE 3 – Carte de la solution pour l'instance t20-c2-same-side

meilleure visualisation, nous avons décidé d'afficher dans nos figures les solutions optimales pour les critères de *makespan* et de durée des routes, traités dans un ordre lexicographique. Les performances de cette nouvelle modélisation ne sont pas abordées dans cet article.

Focus sur l'instance t20-c2-same-side. Les résultats précédemment présentés se comparent naturellement à la situation pour laquelle le budget maximal autorisé est doublé. La carte associée à la solution pour l'instance t20-c2-same-side est représentée sur la figure 3. Dans cette solution, nous observons l'utilisation d'un drone et d'un rover supplémentaires pour effectuer l'ensemble des tâches. Le diagramme de Gantt associé à cette solution est représenté sur la figure 4. Nous observons que la valeur du *makespan* a diminué de 30%, ce qui montre l'impact du budget sur la performance de la mission.

Fronts de Pareto des solutions pour les instances t20. Les fronts de Pareto représentés sur la figure 5 montrent les résultats des instances pour lesquelles 20 tâches doivent

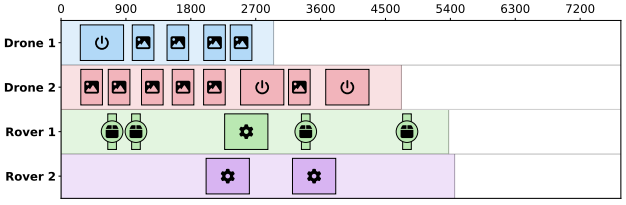

FIGURE 4 – Diagramme de Gantt de la solution pour l'instance t20-c2-same-side

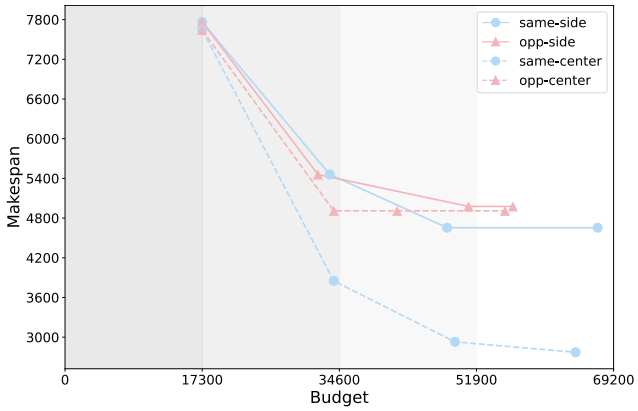

FIGURE 5 – Fronts de Pareto des solutions pour les instances *t20*

être réalisées (instances *t20*). Chaque front de Pareto est composé de 4 points représentant chaque solution par son budget sur l'axe des abscisses et la valeur du *makespan* sur l'axe des ordonnées. Les bandes grises représentent le budget maximal alloué, la contrainte étant progressivement relâchée de la gauche vers la droite.

Chaque front montre que la performance de la mission est fortement dégradée (*i.e.* la valeur du *makespan* croît) lorsque le budget alloué pour la mission se restreint. De plus, on observe qu'à partir d'un certain budget de mission, le *makespan* est difficilement améliorable par l'augmentation du budget de mission et donc par l'ajout potentiel de rovers ou de drones supplémentaires.

Nous observons qu'une distribution des équipements *same* permet d'atteindre des meilleures performances de mission qu'une distribution *opposite*. De plus, nous observons qu'une position de base *center* permet également d'obtenir de meilleures performances qu'une position de base *side*, ce qui est attendu.

6 Conclusion

Dans cet article, nous avons i) défini formellement le problème global incluant les trois niveaux de configuration de flottes hétérogènes, ii) établi un modèle de programmation par contraintes pour ce problème, et iii) testé et évalué ce modèle sur de nouvelles instances avec le solveur CP Optimizer.

L'un de nos travaux futurs est l'évaluation de notre modèle avec d'autres solveurs de programmation par contraintes tels que Choco-solver [16]. Ce travail pourra être complété par l'évaluation d'autres approches heuristiques (*p. ex.* recherche locale) ou exactes (*p. ex.* modèle MILP). Nous envisageons également d'enrichir le modèle afin de traiter des scénarios réalistes lors d'expérimentations in situ. Enfin, nous remettrons en cause chacun des postulats énoncés dans la section 3.3, dans le but d'améliorer la pertinence de notre problème face aux enjeux réels de configuration.

Remerciements

Les travaux menés dans la thèse de Lucas Ligny sont financés par l'Agence de l'innovation de défense (AID) du Ministère des Armées.

Références

[1] Paola Cappanera, Luís Gouveia, and Maria Grazia Scutellà. *The Skill Vehicle Routing Problem*, pages 354–364. Springer Berlin Heidelberg, 2011.

[2] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. The vehicle routing problem. In Nicos Christofides, Aristide Mingozzi, Paolo Toth, and Claudio Sandi, editors, *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.

[3] Jean-François Cordeau and Gilbert Laporte. A tabu search algorithm for the site dependent vehicle routing problem with time windows. *INFOR : Information Systems and Operational Research*, 39(3) :292–298, August 2001.

[4] Boris Delaunay. Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles*, pages 793–800, 1934.

[5] Robert W. Floyd. Algorithm 97 : Shortest path. *Commun. ACM*, 5(6) :345, June 1962.

[6] Mette Gamst and David Pisinger. Decision support for the technician routing and scheduling problem. *Networks*, 83(1) :169–196, September 2023.

[7] Frédéric Gardi, Thierry Benoist, Julien Darlay, Bertrand Estellon, and Romain Megel. *Mathematical Programming Solver Based on Local Search*. Wiley, June 2014.

[8] Gabriella Gigante, Domenico Pascarella, Salvatore Luongo, Carlo Di Benedetto, Angela Vozella, and Giuseppe Persechino. Game-theoretic approach for the optimal configuration computing of an interoperable fleet of unmanned vehicles. *Expert Systems*, 35(5) :e12293, 2018. e12293 10.1111/exsy.12293.

[9] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9) :1563–1581, November 1966.

[10] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling : 20+ years of scheduling with constraints at ibm/ilog. *Constraints*, 23(2) :210–250, March 2018.

[11] Rafael F.F. Lemme, Edilson F. Arruda, and Laura Bahiense. Optimization model to assess electric vehicles as an alternative for fleet composition in station-based car sharing systems. *Transportation Research Part D : Transport and Environment*, 67 :173–196, 2019.

[12] Lucas Ligny, Gauthier Picard, Stéphanie Roussel, and Elise Vareilles. Jeu de données pour le problème de configuration de flottes hétérogènes. https://doi.org/10.57745/S8V5MM, 2025.

[13] Dimitris C. Paraskevopoulos, Gilbert Laporte, Panagiotis P. Repoussis, and Christos D. Tarantilis. Resource constrained routing and scheduling : Review and research prospects. *European Journal of Operational Research*, 263(3) :737–754, December 2017.

[14] Victor Pillac, Christelle Guéret, and Andrés L. Medaglia. A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7) :1525–1535, October 2012.

[15] Thomas Pouré, Elise Vareilles, Stéphanie Roussel, and Gauthier Picard. Configuration of heterogeneous agent fleet : a preliminary generic model. In *ConfWS'24 : 26th International Workshop on Configuration*. CEUR Workshop Proceedings, 2024.

[16] Charles Prud'homme and Jean-Guillaume Fages. Choco-solver : A java library for constraint programming. *Journal of Open Source Software*, 7(78) :4708, October 2022.

[17] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4) :455–472, November 2006.

[18] Hamid Reza Sayarshad and Reza Tavakkoli-Moghaddam. Solving a multi periodic stochastic model of the rail–car fleet sizing by two-stage optimization formulation. *Applied Mathematical Modelling*, 34(5) :1164–1174, 2010.

[19] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2) :254–265, April 1987.

[20] Pascal Van Hentenryck. *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA, 1999.

[21] Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1) :11–12, January 1962.

[22] Şule Yıldırım and Barış Yıldız. Electric bus fleet composition and scheduling. *Transportation Research Part C : Emerging Technologies*, 129 :103197, 2021.