MajecSTIC 2009
Avignon, France, du 16 au 18 novembre 2009

Gestion dynamique des activités des chariots cavaliers sur un terminal portuaire à conteneurs en environnement incertain : approche par intelligence collective

Gaëtan Lesauvage

Université du Havre, LITIS EA 4108, BP 540, 76058 Le Havre - France.

Contact : gaetanlesauvage@gmail.com

Résumé
Le projet CALAS a pour but de mettre au point un système de positionnement laser capable de localiser précisément les chariots cavaliers sur un terminal à conteneurs afin d'envisager une optimisation de leur activité. Un terminal à conteneurs est un système ouvert sujet à la dynamicité où un grand nombre d'événements peuvent survenir, notamment les arrivées et les départs de conteneurs. Notre but est d'optimiser le déplacement des chariots cavaliers afin d'améliorer la performance globale du terminal. L'état exact du système n'est pas connu de manière fiable. Notre système d'optimisation doit donc être tolérant aux fautes et adaptatif. Dans ce contexte nous proposons une approche de résolution du problème d'affectation des missions qui utilise une méta-heuristique basée sur Ant Colony. Nous avons construit un simulateur capable de tester et de comparer différentes politiques d'ordonnancement.

Abstract
The CALAS project consists in a laser measure system allowing to localize precisely straddle carriers in a container terminal. The information given by such a tool makes an optimization possible. A box terminal is an open system subject to dynamics, in which many events can occur. Among others, they concern container arrivals and departures. We aim to optimize the straddle carrier handling in order to improve the terminal management. The exact state of the system is unknown. The optimization process that we try to build must be fail-safe and adaptive. In this context, we propose an approach using a meta-heuristic based on Ant Colony to resolve the problem of assigning missions to straddle carriers. We built a simulator which is able to test and to compare different scheduling policies.

Mots-clés : intelligence collective, colonies de fourmis colorées, graphe dynamique, optimisation multi-critères, problème dynamique de ramassage et de livraison avec fenêtre de temps.

Keywords: swarm intelligence, colored ant colony system, dynamic graph, multiple criteria optimization, dynamic pickup and delivery problem with time window.

---

1. Introduction

Le projet CALAS a pour objectif de localiser précisément les véhicules de manutention au sein d'un terminal portuaire à conteneurs. Une interface logicielle permet de gérer les données récupérées par le système de balises laser. Ce projet est le fruit de la collaboration entre *Laser Data Technology Terminal* et les *Terminaux de Normandie*. Le but de ce projet est de connaître en temps réel l'état exact du terminal, c'est-à-dire à la fois des conteneurs et des véhicules de manutention. Un terminal portuaire à conteneurs est divisé en trois parties principales.Chaque partie est composée de travées dans lesquelles des conteneurs peuvent être empilés. Ces allées sont accessibles par des routes à sens unique. La première zone est située le long d'un canal maritime où les navires viennent s'amarrer au quai. Cette zone est destinée au (dé)chargement des navires. La seconde zone est utilisée pour (dé)charger des camions ou des trains. Elle est située le long de la route et des

rails qui permettent aux véhicules d'accéder au terminal. Enfin, la troisième zone sert d'espace de stockage reliant les deux autres zones. Lors d'une mission de chargement, les conteneurs sont déplacés depuis cette zone vers le navire, le camion ou le train à charger. Au contraire, lors d'une mission de déchargement, le conteneur ainsi déchargé d'un véhicule est stocké dans cette zone intermédiaire.
Gérer un terminal à conteneurs implique trois types de tâches :
– Préparer le (dé)chargement d'un navire ;
– Préparer le (dé)chargement d'un camion ou d'un train ;
– Optimiser la zone de stockage.
Dans le but d'accomplir ces tâches, les conteneurs sont déplacés d'une position à une autre. De tels déplacements sont appelés des missions. Chaque mission est assignée à un seul véhicule de manutention. Parmi ces véhicules, nous nous occuperons des chariots cavaliers. Ces portiques roulants enjambent les conteneurs et permettent d'en manipuler différents types en les agrippant par le dessus.
Un terminal à conteneurs est un système ouvert sujet à la dynamicité. Bien qu'une partie des missions à effectuer soit connue avant de commencer la planification, de nouvelles missions arrivent au cours de la journée lorsque la planification a déjà été établie. De plus, l'heure prévue d'arrivée des camions, des trains et des navires n'est pas suffisamment précise pour prévoir les livraisons de conteneurs. Si un camion est en retard, le chariot cavalier devra attendre son arrivée alors qu'il aurait pu être affecté à une autre mission pendant ce temps. Le système est également influencé par les comportements humains car les chariots cavaliers sont conduits par des individus qui peuvent choisir de respecter ou non la planification des missions.

2. Travaux connexes

Dans un tel système, le temps d'attente des navires, des camions et des trains doit être aussi court que possible. Trois moyens différents ont déjà été utilisés pour résoudre des instances concrètes de ce problème. Tout d'abord, l'approche analytique est basée sur une étude de facteurs interconnectés qui doivent être pris en compte afin d'améliorer l'efficacité du système. Dans [13], un système d'aide à la décision est décrit. Il a été créé en étudiant des décisions étroitement reliées prises quotidiennement dans un terminal à conteneurs. Les auteurs ont évalué ce système dans un terminal à Hong-Kong et ont mesuré une réduction de 30% des temps d'attente des navires et de 35% des coûts de manutentions des conteneurs. La seconde approche est la simulation. Elle consiste à mettre au point un simulateur capable de mettre à l'épreuve plusieurs méthodes d'optimisation. Dans [7], les auteurs ont utilisé à la fois un algorithme génétique et un système de réseau de neurones pour la régulation des opérations de stockage des conteneurs. Dans [14], les auteurs ont essayé d'améliorer la performance du port Maasvlakte de Rotterdam en étudiant son design. Leur simulation apporte de l'information sur la longueur des quais, la capacité de stockage, la gestion et le transport de l'équipement du terminal. Ces résultats ont été très utiles pour la création des prochains terminaux. Enfin, la dernière approche est le système multi-agent (SMA). Thurston et Hu [15] ont tenté d'améliorer la performance du terminal grâce à une replanification dynamique et coopérative des portiques de quai et des chariots cavaliers. Henesey *et al.* [6] ont développé cette idée. Leurs agents essayent d'atteindre leur propre objectif en cherchant, coordonnant, communiquant et négociant avec d'autres agents. Ils prennent leurs décisions selon un mécanisme de négociation basé sur le modèle boursier.
Il existe donc plusieurs manières de conduire une optimisation et plusieurs critères ont été pris en compte. Nous nous sommes intéressés à la gestion des déplacements des véhicules au sein d'un terminal ainsi qu'à l'allocation de leurs missions. Dans ce contexte, nous sommes en présence d'un problème de tournées de véhicules.

3. Problèmes de tournées de véhicules (VRP)

Les problèmes de tournées de véhicules (*Vehicle Routing Problems*) ont largement été étudiés et présentent un intérêt pratique depuis leur apparition dans de nombreux procédés industriels. En général, VRP est formulé comme suit. Un ou plusieurs véhicules doivent partir d'un dépôt, visiter un ensemble de clients, collecter ou livrer des biens, et revenir au dépôt. Le but est de

minimiser les itinéraires des véhicules. Plusieurs sous-problèmes différents appartiennent à la classe des VRP. Chaque sous-problème comporte une légère variation du problème principal. Nous distinguons les instances statiques et dynamiques de ces problèmes car les méthodes de résolution sont différentes.

3.1. Problèmes de Tournées de Véhicules avec Fenêtres de Temps (VRPTW)

Le problème de tournées de véhicule avec fenêtres de temps [2] (*Vehicle Routing Problem with Time Windows*) consiste à faire visiter un ensemble de villes à un ensemble de véhicules d'une certaine capacité et d'optimiser la distance totale à parcourir. Prenons par exemple une usine italienne qui produit des jouets. Elle doit livrer un ensemble de magasins à travers le pays. Les biens sont transportés par des camions qui ont une capacité limitée et qui doivent partir de l'entrepôt de l'usine. Les livraisons ne peuvent se faire que pendant un certain interval de temps. Si un camion arrive trop tôt, il devra attendre. La solution de ce problème doit minimiser la distance totale parcourue par les camions.

Le problème dynamique de tournées de véhicules avec fenêtres de temps (*Dynamic Vehicle Routing Problem with Time Windows*) inclut l'aspect dynamique des nouvelles requêtes. En reprenant l'exemple précédent, si les magasins peuvent demander à être livrés pendant qu'une planification (et donc un itinéraire) a déjà été calculée, alors ce problème est un problème dynamique de tournées de véhicules avec fenêtres de temps.

3.2. Problème de collecte et de livraison (PDP)

La sous-classe de problème de collecte et de livraison la plus fréquemment rencontrée dans la réalité concerne les *Pickup and Delivery Problems* de type "un à un" (One to One PDP). Il s'agit de collecter un bien à un emplacement et à le livrer à un autre emplacement. Le principal problème de de cette sous-classe est le problème de tournées de véhicules avec collecte et livraison (Vehicle Routing Problem with Pickup and Delivery). Dans ce problème, il faut calculer les meilleurs itinéraires pour une flotte de véhicules dans le but de déplacer des objets sur un graphe. Chaque itinéraire doit commencer et se terminer au dépôt.

Lorsque le problème concerne le transport de personnes, il est appelé problème de transport à la demande (Dial-A-Ride Problem). Plusieurs cas particuliers de problèmes de tournées de véhicules avec collecte et livraison sont également couramment rencontrés dans la réalité. Le problème de la grue empileuse (Stacker Crane Problem) est un problème à simple véhicule avec capacité unitaire. Dans un autre sous-problème, les véhicules sont autorisés à déposer temporairement leur chargement sur des points spécifiques appelés points de transbordement. Le but est d'être capable de répondre rapidement aux demandes des clients. Ce problème est appelé problème de tournées de véhicules avec collecte et livraison et transbordement. Lorsque plusieurs requêtes ne sont pas connues à l'avance, les problèmes statiques décrits ci-dessus deviennent dynamiques. Ces problèmes dynamiques de collecte et de livraison [11] (Dynamic Pickup and Delivery Problem ou Dynamic Vehicle Routing Problem with Pickup and Delivery) consistent à optimiser les itinéraires des véhicules afin de collecter un chargement à un endroit, puis de le livrer à sa destination en adaptant ces itinéraires aux nouvelles missions sans pour autant recalculer tout à partir de zéro. La plupart du temps, ce problème doit gérer des fenêtres de temps (Dynamic Vehicle Routing Problem with Pickup and Delivery and Time Windows). En effet, pour commencer une mission, les véhicules doivent attendre le début de la fenêtre de temps. S'il ne la respecte pas, alors le véhicule devra attendre, devenant indisponible pour d'autres missions, et donc inutile.

Comme expliqué précédemment, la classe des problèmes de tournées de véhicules comporte un grand nombre de sous-problèmes spécifiques. Il est donc très important de bien identifier notre propre problème.

3.3. Identification du problème

Dans notre problème [9], plusieurs véhicules (chariots cavaliers) de capacité unitaire doivent accomplir des missions (en déplaçant des conteneurs au sein du terminal). Ils peuvent également utiliser des points de transbordement pour améliorer les performances. La particularité de notre système est que les chariots cavaliers peuvent commencer leur tournées depuis n'importe quel emplacement du terminal, *i.e.* ils ne sont pas obligés de commencer à partir du dépôt. De plus, chaque mission possède deux fenêtres de temps. La première concerne le ramassage du conteneur,

et la seconde est celle dans laquelle le conteneur doit être livré. Si un véhicule arrive en avance pour collecter ou livrer un conteneur, il devra attendre le début de la fenêtre de temps. D'autre part, si le chariot cavalier est en retard, c'est-à-dire que sa fenêtre de temps est déjà dépassée, dans certains cas, la mission doit être abandonnée et une nouvelle mission concernant le même conteneur sera insérée dans le système.
Pour toutes ces raisons, notre problème appartient à la classe des problèmes dynamiques de tournées de véhicules avec collectes et livraisons et fenêtres de temps (DVRPPD-TW).
Trois problèmes inter-connectés doivent être résolus :
– Minimiser les déplacements des chariots cavaliers : problème de plus court chemin ;
– Minimiser les ressources : problème de partitionnement ;
– Minimiser les délais d'attente des clients : problème d'ordonnancement.
Afin de construire une planification correcte, le système doit prendre en compte le concept de plus court chemin. Parallèlement, prendre en compte les distances dans l'ordonnancement tend à réduire la distance globale parcourue par les cavaliers. De plus, nous avons défini un niveau de qualité de service (Quality Of Service) pour satisfaire des clients tout en réduisant les coûts de fonctionnement. Il s'agit d'un problème dynamique de grande échelle qui requiert une solution en temps réel. Nous proposons un algorithme en ligne basé sur *Ant Colony Optimization* [4] et plus précisément sur une version colorée de cet algorithme d'intelligence collective [1].

4. Gestion des chariots cavaliers et colonies de fourmis

Ant Colony [4] est une méta-heuristique qui fait apparaître une solution en plaçant des fourmis artificielles sur un espace de solutions. Le système est auto-régulé. En effet, les fourmis déposent de la phéromone sur tout ou partie de la solution trouvée selon sa qualité (rétro-action positive). Cependant, la piste de phéromone s'évapore progressivement (rétro action négative). La rétro-action positive fait converger l'algorithme vers une solution de qualité alors que la rétro-action négative empêche l'algorithme de tomber dans un extremum local.
Ant Colony à une colonie fournit en sortie une liste de missions à accomplir [3, 12]. Le problème est d'assigner un chariot cavalier à une mission.
Nous proposons d'employer une solution utilisant des fourmis colorées [1]. Dans notre modèle, chaque chariot cavalier représente une colonie avec sa propre couleur. La convergence est assurée par le fait que les fourmis sont attirées par la phéromone de leur propre colonie et repoussées par celles des autres colonies. Cette approche simule un mécanisme de collaboration et de compétition entre colonies et fournit une liste triée de missions pour chaque chariot cavalier.

4.1. Modélisation

Notre algorithme utilise une représentation du problème sous forme de graphe. Dans ce graphe orienté, chaque sommet représente une mission. Tout d'abord, nous relions les missions entre elles par des liens de précédence. Une mission est dite précédente à une autre si sa fenêtre de temps commence avant celle de l'autre mission. Une fois ce graphe de précédence établi, un sommet coloré est ajouté au graphe pour chaque chariot cavalier. Ces nœuds sont reliés à chaque mission compatible du graphe de précédence par un arc de sa couleur. Ensuite, les arcs ajoutés lors de la phase de construction du graphe de précédence sont colorés en fonction de la compatibilité entre le chariot cavalier et les missions. En effet, si deux missions, reliées par un arc dans le graphe de précédence, sont compatibles avec le chariot cavalier de couleur c, alors l'arc entre ces deux sommets prendra la couleur c. S'il existe déjà un arc coloré entre ces deux sommets, alors au lieu de changer la couleur de cet arc, un nouvel arc est ajouté et prend la couleur c. Enfin, s'il reste des arcs non colorés, ils sont tout simplement supprimés du graphe. Nous obtenons donc un multi-graphe permettant de faire fonctionner notre algorithme de fourmis colorées.

Prenons une instance simple de notre problème où deux chariots cavaliers doivent exécuter quatre missions. La compatibilité entre ces véhicules et les missions est comme décrite dans la Fig. 1. Nous construisons alors le graphe de précédence (*c.f.* partie gauche Fig. 2). Nous ajoutons ensuite les sommets représentant les chariots cavaliers (*c.f.* partie centrale Fig. 2). Enfin, nous colorons les arcs comme décrit précédemment. La partie droite de la Fig. 2 montre le multi-graphe obtenu en utilisant cette procédure.

Chariots cavaliers :

Nom	Couleur
s0	vert
s1	bleu

Missions :

Nom	Début	Fin	Chariots cavaliers compatibles
m0	5h00	6h00	s0, s1
m1	5h30	6h00	s0
m2	7h00	9h00	s0
m3	6h00	7h30	s0, s1

FIG. 1 – Exemple d'une instance simple de notre problème

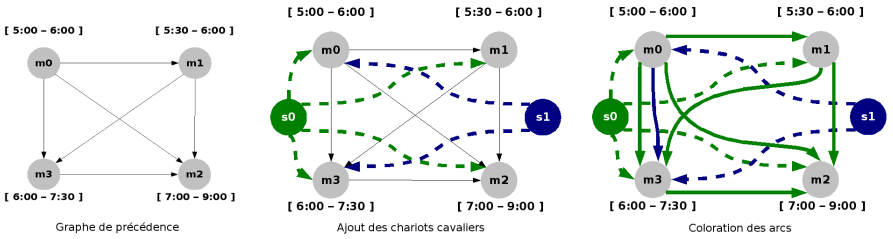

FIG. 2 – Étapes de construction du graphe de missions du problème décrit sur la Fig. 1

Nous introduisons des pondérations sur les arcs du graphe afin d'influencer les fourmis lors de leurs déplacements. Le poids d'un arc symbolise la pertinence de l'enchaînement de deux missions par le même chariot cavalier. Cette partie de notre modèle permet une grande flexibilité. En effet, ce système permet de mettre à l'épreuve différentes politiques de pondération.

Notre heuristique prend en compte à la fois les coûts d'exécution des missions et les fenêtres de temps temporellement proches. En effet, si deux missions ont des fenêtres de temps qui sont trop rapprochées et qu'elles sont affectées au même chariot cavalier, alors l'exécution de la première mission provoquera le dépassement de la fenêtre de temps de la seconde. Il est très important d'empêcher ce phénomène en le prenant en compte dans la modélisation du graphe. Ainsi le calcul du poids de l'arc reliant deux missions utilise une pénalité de proximité temporelle. Nous avons également défini un concept de priorité. Plus la fin d'une fenêtre de temps est proche, plus la priorité de la mission est élevée. La fonction de pondération des arcs prend également en compte la distance entre le lieu de livraison de la première mission et le lieu de collecte de la seconde.

4.2. Algorithme de fourmis colorées

Dans cet algorithme, à chaque chariot cavalier correspond une colonie de la même couleur. Au départ de l'algorithme, chaque individu d'une colonie est placé sur le nœud représentant son chariot cavalier. Ensuite, les fourmis parcourent le graphe en utilisant uniquement les arcs de leur couleur. Lorsqu'une fourmi atteint un nœud, elle choisit sa prochaine destination en fonction de trois critères : le taux de phéromone de sa propre couleur, le taux de phéromone des autres couleurs et le poids de l'arc.

La fourmi est attirée par la phéromone de sa couleur et repoussée par celle des autres couleurs. Une fois qu'une fourmi atteint sa destination, elle dépose de la phéromone de sa couleur en fonction de la qualité de la solution représentée par son choix de destination. Lorsqu'un chariot cavalier de couleur c demande une nouvelle mission à effectuer, il choisit la mission qui a le taux le plus élevé de phéromone de couleur c.

Notre approche par colonies de fourmis semble pertinente pour la résolution du problème considéré, de par sa nature dynamique, de par la taille de l'espace de solution et de par la contrainte du temps réel.

Le principal avantage d'Ant Colony est de fournir une solution dite *anytime* c'est-à-dire qu'il est possible d'obtenir une solution valide à n'importe quel moment de l'exécution du programme. C'est un algorithme en ligne qui s'adapte relativement facilement aux changements de l'environ-

nement. En effet, les fourmis renforcent les taux de phéromone pour se rapprocher de la solution optimale. Parallèlement, le processus d'évaporation permet de contrôler l'algorithme afin de l'empêcher de se piéger dans optimum local et également de permettre de prendre en compte les événements dynamiques.
Ant Colony comporte un très grand nombre de paramètres comme le taux d'évaporation, l'évaluation de la qualité de la solution, la quantité de phéromone déposée par une fourmi, la vitesse de déplacement des fourmis, *etc*... Il s'agit du principal défaut de cette méta-heuristique. La qualité de la solution est étroitement liée à ces paramètres en interactions.
Nous avons essayé de rendre ces paramètres auto-adaptatifs. Pour cela nous utilisons une méthode locale pour adapter certains de ces paramètres pendant l'exécution de l'algorithme.

4.3. Division du travail

Comme il existe plusieurs colonies distinctes et que chacune d'elles ne possède qu'une vision locale de son environnement, il n'est pas possible d'utiliser un système de marquage de phéromone basé sur une caractéristique globale du système. Une colonie ne peut pas comparer la qualité de sa solution avec les solutions des autres colonies. Nous devons donc utiliser la même méthode de dépôt de phéromone pour chaque colonie. Cependant, nous sommes capables d'adapter la quantité de phéromone déposée par les fourmis d'une colonie en fonction des compétences d'un véhicule envers une tâche. En effet, nous observons qu'il est possible de réduire le temps de service d'une mission en spécialisant les véhicules dans un type de missions. Nous pouvons augmenter la quantité de phéromone déposée par un véhicule donné pour une tâche concernant une zone spécifique du terminal à conteneurs (zone des quais, zone de route/rail, ou zone intermédiaire) et décroitre la quantité déposée pour des tâches situées dans les autres zones du terminal. Dans le même temps, nous procédons inversement pour tous les autres véhicules. Nous essayons ainsi de spécialiser les véhicules dans un type de tâches et nous sommes capables de réguler ces quantités de phéromone en prenant en compte à la fois les critères de préférence et de distance.

4.4. Réduction des ressources

Toujours dans l'objectif de réduire les coûts de production, nous essayons de diminuer le nombre de chariots cavaliers dans le système. Notre solution courante au problème dans son entier a tendance à distribuer les missions sur l'ensemble des véhicules. Chaque véhicule a donc environ le même taux d'activité. Cependant, si ce taux tombe en dessous du niveau bas d'un certain seuil objectif, il est possible de conclure qu'un véhicule pourrait être supprimé. D'autre part, si ce taux est supérieur au niveau haut d'un seuil objectif défini, il est possible de dire qu'un véhicule supplémentaire devrait être ajouté à la flotte de véhicules.

5. Simulateur

Le simulateur comporte deux parties principales. La première est la simulation du terminal à conteneurs (*c.f.* Fig. 3 partie gauche). La seconde partie est celle de l'algorithme de fourmis colorées (*c.f.* Fig. 3 partie droite). Cette partie du simulateur utilise la librairie GraphStream[1] qui permet de gérer des graphes dynamiques facilement [5]. Le simulateur utilise un moteur de temps discret qui doit itérer chaque objet de la simulation à chaque pas de temps. Pendant la simulation, le fichier de scénario est lu et les événements dynamiques qu'il contient sont répercutés à la fois sur la vue du terminal et sur celle d'*Ant Colony*. De cette manière, le système peut simuler la dynamicité de l'arrivée des missions et de la disponibilité des véhicules.
Afin de générer des instances de test pertinents et d'obtenir des résultats cohérents, nous avons défini plusieurs niveaux de dynamicité. Dans [8], Allan Larsen indique deux principales mesures du degré de dynamicité. Tout d'abord, le degré de dynamicité (*Degree Of Dynamism* : dod) [10] est le rapport entre le nombre de requêtes dynamiques et le nombre total de requêtes. Le principal point négatif de cette mesure est de ne pas prendre en compte la date d'arrivée des requêtes dans le système. En effet, avec dod, si les requêtes dynamiques entrent dans le système en début de planification, le système est considéré comme aussi dynamique que si elles entraient en toute fin.

[1] `http://graphstream-project.org/`

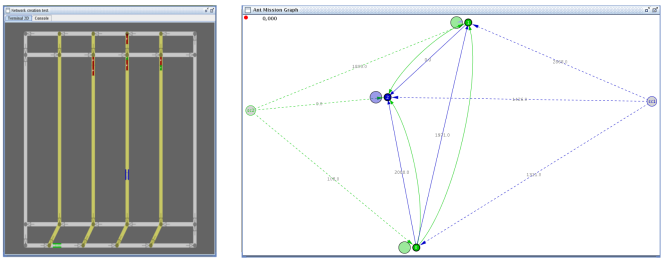

FIG. 3 – Vue du terminal et du graphe de missions dans le simulateur

	Statique	Semi Dynamique	Dynamique
d o d	0	0.5	1
e d o d	0	0.25	1
Temps final	22693	22276	22693
Nombre de fenêtres dépassées	3	5	7
Temps total de dépassement	6467	8477	12485

FIG. 4 – Résultats des simulations

Pourtant, plus ces requêtes sont connues tard et plus les délais de livraison sont court. Ces retards ont un impact négatif sur les performances du système. Pour cette raison, Larsen *et al.* dans [8] a défini le degré effectif de dynamicité (*Effective Degree Of Dynamism* : edod) selon cette formule :

$$\mathrm{edod} = \frac{\sum_{i=1}^{\eta_d} \frac{t_i}{T}}{\eta_d + \eta_s} \tag{1}$$

Ici, η_s et η_d sont respectivement le nombre de requêtes statiques et dynamiques. t_i est la date d'arrivée d'une requête i (avec $0 < t_i < T$) et T correspond à la date de fin de simulation. Cette mesure prend en compte la moyenne des dates d'arrivées des requêtes dans le système. Plus les requêtes dynamiques arrivent tard, et plus edod sera important. Si $\mathrm{edod} = 0$, alors le système est totalement statique. Sinon, si $\mathrm{edod} = 1$, le système est purement dynamique.

6. Résultats préliminaires

Étant toujours dans la phase de collecte de données de la part de nos partenaires, nous ne pouvons que simplement tester la pertinence de notre algorithme sur des données simulées. Nous avons, dans ce but, tout d'abord lancé une simulation dans un contexte totalement statique, c'est-à-dire que chaque mission est connue au tout début de la simulation et que les ressources (chariots cavaliers) sont toujours disponibles. Dans un second temps, nous avons inséré des événements dynamiques comme l'arrivée de nouvelles missions. Pour chaque simulation, nous avons mesuré le temps total nécessaire pour effectuer toutes les missions (en nombre d'itérations), le nombre de fenêtres de temps dépassées et le temps total de ces dépassements. Nous avons considéré qu'une fenêtre de temps a été dépassée si le non respect de cette fenêtre de temps fait l'objet d'une pénalité pour le terminal. En effet, si la fenêtre de temps d'une mission dans laquelle il faut charger ou décharger un camion est dépassée alors le transporteur est en droit de demander une compensation.
La figure 4 montre les résultats de trois instances contenant 12 missions et 3 chariots cavaliers chacune. La seule différence entre ces trois problèmes réside dans leur degré de dynamicité. En effet, nous avons simplement changé la date d'arrivée de ces missions dans le système afin de les rendre plus ou moins dynamiques. Comme nous pouvons le voir sur la Fig. 4, notre algorithme semble agir comme attendu. Plus les missions sont connues à l'avance et meilleures sont les performances. Le pire des cas correspond à l'instance totalement dynamique ($\mathrm{dod} = 1$ et $\mathrm{edod} = 1$) où les missions entrent dans le système à la date du début de leur fenêtre de temps. Ces résultats ne sont que des résultats préliminaires et nous n'avons pas encore testé tous les paramètres de l'algorithme de fourmis colorées.

7. Conclusion

Le problème considéré dans cet article appartient à la classe des problèmes dynamiques de collecte et de livraison avec fenêtres de temps. Cependant, il ne correspond pas exactement aux sous-problèmes déjà étudiés de cette classe. Il s'agit donc d'un problème original et non résolu. Nous proposons de le résoudre en utilisant une approche par intelligence collective. Un algorithme fourmi est en cours de développement. Il utilise des fourmis colorées et un graphe dynamique dans le but d'obtenir une planification. De plus, nous essayons de minimiser le nombre de véhicules dans la flotte du terminal afin de réduire les coûts de fonctionnement tout en maintenant une qualité de service suffisante. Un simulateur capable de reproduire le comportement d'un tel système et de gérer des événements dynamiques est également en cours de développement. Les résultats préliminaires confirment que notre algorithme est capable de s'adapter aux conditions changeantes et nous sommes actuellement en train de collecter des données afin de comparer les performances de notre système sur un terminal à conteneurs avec les méthodes de planification utilisées par le Port Autonome du Havre, France.

Bibliographie

1. C. Bertelle, A. Dutot, F. Guinand, et D. Olivier. Distribution of agent based simulation with colored ant algorihm. In *14th European Simulation Symposium*, 2002.
2. L. Bianchi. Notes on dynamic vehicle routing - the state of the art. *Technical Report, IDSIA*, 2000.
3. B. Bullnheimer, R.F. Hartl, et C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 1999.
4. M. Dorigo. *Learning and Natural Algorithms*. Thèse de doctorat, Politecnico di Milano, Italie, 1992.
5. Antoine Dutot, Frédéric Guinand, Damien Olivier, et Yoann Pigné. Graphstream : A tool for bridging the gap between complex systems and dynamic graphs. In *EPNACS : Emergent Properties in Natural and Artificial Complex Systems*, 2007.
6. L. Henesey, F. Wernstedt, et P. Davidsson. Market-driven control in container terminal management. In *Second International EuroConference on computer applications and information technology in the maritime industries*, Hamburg, Germany, 2003.
7. Chun Jin, Xinlu Liu, et Peng Gao. An intelligent simulation method based on artificial neural network for container yard operation. *Lecture Notes in Computer Science*, 3174 :904–911, 2004.
8. A. Larsen. *The Vehicle Routing Problem*. Thèse de doctorat, Department of Mathematical Modelling, Technical University of Denmark, 2000.
9. Gaetan Lesauvage. Gestion dynamique des activités des chariots cavaliers sur un terminal à conteneurs en environnement incertain. Thèse de Master, Université du Havre, 2008.
10. Karsten Lund, Oli B. G. Madsen, et Jens M. Rygaard. Vehicle routing problems with varying degrees of dynamism. *Technical report, IMM, The Department of Mathematical Modelling, Technical University of Denmark*, 1996.
11. Snezana Mitrovic-Minic. *The dynamic pickup and delivery problem with time windows*. Thèse de doctorat, Simon Fraser University, 2001.
12. R. Montemanni, L.M. Gambardella, A.E. Rizzoli, et A.V. Donati. A new algorithm for a dynamic vehicle routing problem based on ant colony system. *Technical Report, TR-23-02, IDSIA, Galleria 2, Manno, 6928, Switzerland*, 2004.
13. Katta G. Murty, Jiyin Liu, Yat-wah Wan, et Richard Linn. A decision support system for operations in a container terminal. *Decis. Support Syst.*, 39(3) :309–332, 2005.
14. Jaap A. Ottjes, Hans P. M. Veeke, Mark B. Duinkerken, Joan C. Rijsenbrij, et Gabriel Lodewijks. Simulation of a multiterminal system for container handling. *Container Terminals and Cargo Systems*, 2 :15–36, 2006.
15. Tom Thurston et Huosheng Hu. Distributed agent architecture for port automation. In *COMPSAC '02 : Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life : Development and Redevelopment*, pages 81–90, Washington, DC, USA, 2002. IEEE Computer Society.