UNIVERSITÉ DE MONTRÉAL

AFFECTATION D'ACTIVITÉS ET DE TÂCHES À DES QUARTS DE TRAVAIL FIXÉS

QUENTIN LEQUY
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION DU DIPLÔME DE
PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES ET GÉNIE INDUSTRIEL)
MAI 2011

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

AFFECTATION D'ACTIVITÉS ET DE TÂCHES À DES QUARTS DE TRAVAIL FIXÉS

présentée par : M. LEQUY Quentin, Ing.
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury constitué de :

M. ROUSSEAU Louis-Martin, Ph.D., président.
M. DESAULNIERS Guy, Ph.D., membre et directeur de recherche.
M. SOUMIS François, Ph.D., membre et co-directeur de recherche.
M. GENDREAU Michel, Ph.D., membre.
M. MAHEY Philippe, Doctorat, membre externe.

À ma famille, tenue à l'écart par un océan.

Remerciements

Je remercie vivement M. Guy Desaulniers, mon directeur de recherche, pour sa patience, ses conseils et sa disponibilité qui ont grandement contribué à l'accomplissement de mon travail.

Je remercie également M. François Soumis, mon codirecteur pour sa disponibilité et ses bonnes idées, mais aussi pour avoir accepter ma demande de stage de fin d'études, sans quoi, cette thèse n'aurait pas vu le jour.

Je remercie l'entreprise Kronos et l'École Polytechnique de Montréal pour leur financement et la mise à ma disposition des moyens techniques nécessaires à la réalisation de mon doctorat, ainsi que le personnel du GERAD.

Je remercie M. Philippe Mahey pour sa participation à mon jury, ainsi que MM. Louis-Martin Rousseau et Michel Gendreau.

Un grand merci aussi à toutes les personnes que j'ai croisées au GERAD et qui m'ont soutenu, aidé ou simplement accompagé pendant tout ce temps. Je pense particulièrement à Anthony Guillou et Sébastien Le Digabel, ainsi qu'à tous ceux que j'ai pu croiser autour d'une des tables de la cuisine.

Bien que je ne puisse pas tous les citer, j'aimerais aussi remercier tous les gens qui ont partagé un bout de ma vie à Montréal, et qui m'ont apporté tout ce qu'il peut y avoir autour des études.

Finalement, je termine en remerciant ma famille, qui a vécu loin de moi aussi longtemps, tout en continuant à m'encourager.

Résumé

Pour les moyennes ou grandes entreprises, construire des horaires de travail réalisables à faible coût pour leurs employés n'est pas chose facile. Selon le type d'entreprises, trouver seulement un horaire réalisable peut relever du défi. Néanmoins, les techniques modernes de modélisation et d'optimisation permettent de proposer des méthodes efficaces pour plusieurs classes de problèmes d'horaires de personnel.

Dans cette thèse, nous nous concentrons sur trois problèmes d'horaires de personnel dans lesquels les employés travaillent durant des quarts, par exemple dans des magasins à grande surface. Ces quarts doivent répondre à une demande en clients variable, à satisfaire le plus rapidement possible. Cette demande n'est pas exactement connue lorsque les quarts sont créés et a aussi un caractère périssable. Cela signifie que les compagnies peuvent perdre leurs clients immédiatement s'ils ne sont pas satisfaits du service offert. L'horaire d'un employé est composé de jours ouvrés et de jours vacants. Pour chaque jour ouvré, l'employé est affecté à un quart de travail défini par des temps de début et des temps de fin, et pouvant aussi contenir une ou plusieurs pauses. Dans notre cas d'étude, les temps de début et de fin, ainsi que le nombre idéal d'employés pour satisfaire la demande en clients, sont déterminés à l'avance par un module de prévision.

Le travail d'un employé est catégorisé entre les tâches et les activités. Les activités correspondent au service des clients en temps réel, alors que les tâches n'impliquent pas un rapport direct avec les clients, bien que celles-ci soient en général cruciales. Dans les faits, une tâche est un travail non interruptible, tel que mettre en place une vitrine, tandis qu'une activité est facilement interruptible à tout moment, en remplaçant un employé par un autre, par exemple pour les caissiers. Les tâches ont une nature unique, isolée, sont effectuées par un employé qualifié avant une date limite, et requièrent éventuellement la complétion d'autres tâches avant leur temps de début. Les activités ont un caractère récurrent et continu. Leur contexte est le cas général d'activités multiples, dans lequel il est possible d'affecter plusieurs activités dans le quart d'un employé.

Finalement, chaque quart de travail doit être rempli avec des activités ou des tâches à accomplir. Les conventions collectives restreignent, en général, la réalisabilité

d'un horaire. De plus, les employés possèdent des qualifications et ne peuvent pas être affectés à des activités ou des tâches pour lesquelles ils ne sont pas qualifiés. Lors de la construction des horaires des employés, l'objectif principal de l'entreprise est de réduire ses coûts de personnel, tout en maintenant une satisfaction élevée chez sa clientèle. Un objectif secondaire est d'offrir à ses employés des horaires plaisants à travailler.

Cette thèse traite en premier du problème d'affectation d'activités multiples, qui implique uniquement des activités. Trois modèles en nombres entiers et plusieurs méthodes de résolution sont proposées, mais la plus efficace reste une heuristique de type horizon fuyant basée sur un programme en nombres entiers. Le modèle sous-jacent est résolu par un algorithme de génération de colonnes encapsulé dans un schéma heuristique d'énumération implicite. Une comparaison avec les autres méthodes de résolution a été effectuée sur des instances générées aléatoirement, simulant des instances réalistes.

Le deuxième sujet traité est le problème d'affectation de tâches et d'activités multiples. Le concept de tâche individuelle — qui est effectuée par un seul employé pendant une durée fixe — a été introduit dans la définition du problème précédent. Une décomposition en deux étapes est proposée pour s'attaquer aux instances de grande taille, sur lesquelles la résolution directe du modèle global proposé échoue. La première étape de la décomposition consiste à affecter les tâches a priori grâce à un modèle d'approximation, résolu heuristiquement. La deuxième étape produit alors une solution complète pour l'instance en affectant les activités, tout en s'autorisant à réaffecter les tâches. Des tests numériques ont montré que les quatre stratégies de réaffectation proposées sont surclassées par l'une d'entre elles. Ces tests valident également notre méthode et suggèrent que la réaffectation des tâches améliore la qualité des solutions obtenues au coût raisonnable d'un petit temps de calcul supplémentaire.

Le dernier problème est une extension du précédent, par l'introduction des tâches en équipe, ce qui conduit au problème d'affectation de tâches en équipe et d'activités multiples. Le concept des tâches en équipe est une extension naturelle de celui des tâches individuelles et permet de considérer des pièces de travail non interruptibles, effectuées par plusieurs employés simultanément. Les relations entre le nombre d'employés contribuant à une tâche en équipe et le temps qu'ils y contribuent sont appelés patrons d'équipes. Inspirée de l'approche en deux étapes utilisée dans l'affectation de tâches individuelles, la méthode de résolution est basée sur deux modules d'affectation

impliqués dans une descente à voisinages variables. Le premier module est un modèle d'approximation pour affecter les tâches en ne tenant que partiellement compte des activités. Le deuxième module consiste à affecter les activités en réaffectant les tâches. Des expérimentations numériques ont été réalisées et montrent que, premièrement, scinder les tâches peut améliorer la qualité des horaires, et deuxièmement, la méthode de résolution peut affecter des tâches en équipe, même si elles n'apparaissent pas dans la solution initiale de la descente. De plus, l'heuristique d'amélioration confirme l'efficacité de la décomposition en deux étapes pour l'affectation des tâches individuelles uniquement.

Abstract

For most medium- to large-sized companies, building a low-cost feasible working schedule for their employees is not an easy task. Depending on the business type, just finding a feasible schedule might even be a challenge. Nevertheless, modern modeling and optimization techniques allow to propose efficient solution methods for several personnel scheduling problems.

In this thesis, we focus on three planning scheduling problems that arise in companies where the employees work shifts, such as retail stores. Those shifts should respond to a varying customer demand that must be fulfilled as soon as possible. The demand for each so-called activity is not exactly known when shifts are built and is also perishable. That means companies might lose clients immediately if they are not satisfied of the service. The schedule of an employee is composed of working days and days off. For each working day, the employee is assigned to a work shift that is defined by a start and an end time. It may also contain one or several specified break periods. For our concern, those start and end times are pre-determined by previous modules, as well as the ideal number of employees required to satisfy the customer demand, which is determined by a forecasting algorithm.

The work of an employee is categorized between tasks and activities. Activities correspond to servicing customers in real-time, whereas tasks do not involve a direct contact with them, though they are crucial work. Actually, a task is an uninterruptible piece of work, such as setting up a display, whereas an activity can easily be interrupted at any time by replacing one employee with another, such as cashiers. The former has an isolated and unique nature and is performed by a skilled employee before a due date and may require the completion of some other tasks prior to its start. The latter has a continuous and recurrent nature. The context is the general multi-activity case, where it is possible to assign several activities to an employee shift.

Finally, each work shift must be filled with activities or tasks to accomplish. Labor and collective agreement rules restrict the feasibility of a schedule. Furthermore, the employees possess skills and cannot be assigned to an activity or a task for which they are not qualified. When building the employee schedules, the primary objective

of the company is to reduce personnel costs while maintaining customer satisfaction high. A secondary objective might be to offer as much as possible pleasant schedules to the employees.

The thesis deals first with the multi-activity assignment problem, which implies only activities. Three integer models and several solution methods are proposed, but the best one is a rolling horizon heuristic based on an integer programming model. This model is solved by a column generation algorithm embedded in a heuristic branch-and-bound framework. Comparison with several other methods have been performed on randomly generated instances mimicing real-life instances.

The next subject studied is the multi-activity and task assignment problem. Single tasks — which are performed each by only one employee during a fixed time length — are introduced in the definition of the previous problem. A two-stage decomposition is proposed to tackle large-sized instances on which the direct solving of the global model fails. This first stage of the decomposition consists of assigning tasks a priori thanks to an approximation model, solved by a heuristic, and then the second stage produces a complete solution by assigning activities with possible task reassignments. Numerical experiments show that the four proposed strategies for reassigning tasks are outclassed by one of them and validate our method. The results suggest also that reassigning tasks improve the solution quality a lot at the reasonable cost of a small extra computational time.

The last problem extends the previous one by introducing team tasks to yield the multi-activity and team task assignment problem. Introducing team tasks is a natural extension to single tasks and allows one considering uninterruptible pieces of work done by several employees simultaneously. The couples number of employees contributing to them and time they spend are called team patterns. For each team task, such a pattern must be selected. Inspired by the two-stage approach used for assigning single tasks, the solution method is based on two modules involved in a variable neighborhood descent. The first module is an approximation model for assigning tasks without activities. The second module consists of assigning activities with task reassignments during the process. Computational experiments show that slicing tasks may improve the quality of the schedules and that the proposed solution method is able to assign team tasks even if they do not appear in the initial solution of the descent. Moreover the improving heuristic confirms the efficiency of the two-stage decomposition for assigning only single tasks.

Table des matières

Liste des tableaux

Liste des figures

Chapitre 1

INTRODUCTION

Déterminer les horaires de travail des employés est un enjeu important pour les entreprises ou les organisations de grosse taille. La planification des horaires conditionne la satifaction des besoins des clients et la gestion de la masse salariale. Celle-ci représente en général la plus grosse partie des coûts de fonctionnement d'une organisation. Dans cette optique, la planification d'horaires de personnel consiste donc à faire travailler les bons employés au bon moment. Les personnes chargées de la gestion du personnel ont ainsi besoin d'outils performants pour trouver cette bonne combinaison d'employés, garantissant à la fois la qualité de service et le bien-être du personnel, et ne mettront pas en péril la santé financière de l'entreprise.

1.1 Contexte de l'étude

La planification d'horaires de personnel donne naissance à des problèmes difficiles à résoudre de par leur complexité, provenant autant des différentes règles de travail que de la variété des employés. Il est alors difficile de trouver des solutions respectant toutes les contraintes de ces problèmes de grande taille, en général très contraints. Il est encore plus difficile de trouver une solution optimale lorsqu'un critère doit être minimisé, par exemple des coûts salariaux.

La planification a de nombreux domaines d'applications, que ce soit dans les transports (aérien notamment), les centres d'appels, les établissements médicaux (hôpitaux, cliniques privées), les services d'urgence (police, ambulance, pompiers), les services administratifs, les événements d'envergure (compétitions sportives), les services financiers (banques), l'hôtellerie, le tourisme, et l'industrie manufacturière. Même en tentant de regrouper les entreprises par type, à chaque entreprise correspondent pratiquement des besoins ou des règles spécifiques et cette diversité renforce encore plus l'intérêt croissant porté à la planification.

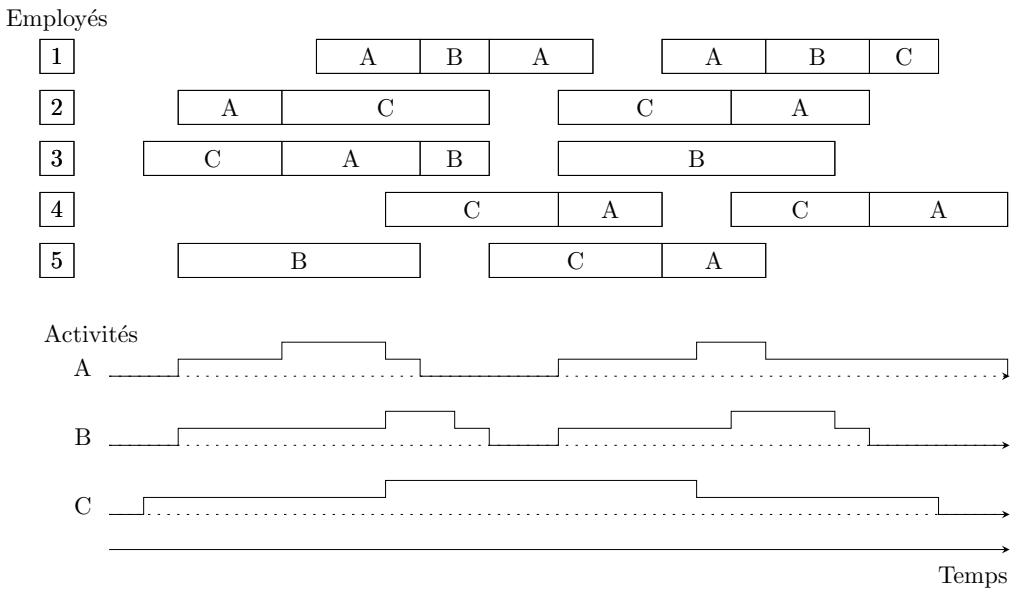

FIGURE 1.1 – Exemple d'emploi du temps

Malgré ces spécificités relatives aux entreprises, le processus de planification demeure semblable et implique une prévision de la demande (souvent basée sur des données historiques), des outils qui fournissent une solution de bonne qualité satisfaisant toutes les règles de travail, et des outils de visualisation et d'analyse de performance, pour qu'un opérateur puisse faire des ajustements si nécessaire. Un tel outil peut avoir un visuel similaire à celui présenté dans la figure 1.1 où les employés et leurs quarts se trouvent en haut de la figure et l'évaluation de la qualité de la solution se trouve en bas.

1.2 Motivations

De tels outils informatiques sont en général commercialisés par des compagnies spécialisées, telles que KRONOS INC.. Cette société de services propose une gestion d'effectifs dans les domaines de la fabrication, de la vente au détail, des loisirs et des services, des soins de santé, des organisations gouvernementales et de l'éducation. KRONOS se targue d'avoir plus de 1 200 clients canadiens, dont Best Buy Canada, Nestlé Canada, The Toronto Star et Woodbine Entertainment. Son engagement envers ses clients est, entre autres, d'harmoniser les effectifs d'une entreprise avec les volumes

d'affaires, de réduire leurs coûts et de rendre les employés plus satisfaits.

Pour ce faire, Kronos dispose de toute une gamme de produits dont certains sont basés sur des modèles d'optimisation. C'est le cas du produit *Workforce Scheduler with Optimization* dont le but est de concilier à la fois les besoins de l'entreprise pour ses opérations internes, les besoins des clients et ceux des employés, dans le contexte de la vente à l'étalage et d'autres environnements axés sur les services.

L'origine du présent sujet de recherche était la volonté de Kronos de rendre ce produit plus compétitif dans le cas des entreprises de vente au détail grâce à l'amélioration de l'affectation des activités et à l'ajout d'un module de gestion de tâches.

1.3 Description de la problématique

Afin de mieux cibler la problématique de cette thèse, il convient de définir sommairement certains termes de base. Catégorisons le travail des employés en deux : un travail facilement interruptible, appelé activité, et un travail difficilement interruptible, appelé tâche. Par exemple, un employé tenant une caisse d'un supermarché peut très vite interrompre son travail pour apporter son aide au service à la clientèle ; son travail appartient alors à la première catégorie. En revanche, un manutentionnaire déchargeant un camion, dont la tournée est sous contraintes temporelles, n'interrompra pas son travail pour aller faire l'inventaire, ce qui place son travail dans la deuxième catégorie. La distinction entre les deux catégories de travail se situe donc entre le caractère continuel des activités et le caractère ponctuel des tâches.

En considérant la demande fournie par un outil de prévision et les quarts de travail fixés par un module au préalable, notre mandat était d'étudier la possibilité d'intégrer l'affectation de tâches dans le processus d'affectation d'activités aux quarts de travail.

Pour faciliter cette étude, trois sujets de recherche ont été déterminés afin de diviser notre démarche scientifique en étapes imbriquées. Le premier sujet de recherche est l'élaboration d'une méthode heuristique pour l'affectation d'activités à des quarts de travail fixés dans un contexte multi-activités. Le deuxième sujet consiste à intégrer l'affectation de tâches déplaçables avec relations de préséance dans l'affectation d'activités. Le troisième sujet concerne une extension du deuxième à des tâches plus complexes pouvant être réalisées en équipe.

1.4 Objectifs

Ces trois sujets de recherche ont chacun amené les contributions de cette thèse. Après avoir présenté une revue de littérature concernant les problèmes d'horaires de personnel dans le chapitre 2, la démarche sous-jacente à cette thèse est synthétisée dans le chapitre 3. Ce chapitre expose les méthodes développées pour les trois sujets de recherche. Il s'agit principalement d'une heuristique de type horizon fuyant pour le premier sujet de recherche. Celle-ci résout des restrictions du problème à des tranches de temps successives, à l'aide d'un branchement heuristique encapsulant un procédé de génération de colonnes. Pour explorer le deuxième sujet, les tâches sont ensuite intégrées dans l'heuristique d'horizon fuyant, grâce à une approche de décomposition en deux étapes. La première étape met en jeu un modèle d'approximation tandis que la deuxième est une modification de l'heuristique d'affectation d'activités du premier sujet, dans laquelle la réaffectation des tâches devient possible pendant la résolution des tranches temporelles. Les tâches sont effectuées par un seul employé à la fois, celui doit être qualifié. Une fenêtre de temps pour leur complétion et des relations de préséance complétent la définition des tâches. Cette définition ne comprend donc pas le travail en équipe qui constitue le troisième sujet. À la définition des tâches est donc ajoutée la notion de patrons d'équipes et de synchronicité entre employés. Une méthode de recherche locale basée sur une approche similaire au traitement des tâches individuelles a été développée. Pour compléter cette thèse, une discussion générale dans le chapitre 5 précède la conclusion du document. Les trois articles résultant de ce travail se trouvent dans les annexes.

Chapitre 2

REVUE DE LITTÉRATURE

La planification d'horaires de personnel est un domaine de recherche actif et prolifique. Cette revue de littérature se veut un aperçu des travaux relatifs aux horaires de personnel. Leur diversité et leur nombre nous obligent donc à nous concentrer sur une partie relativement restreinte de ceux-ci. Afin de se limiter au sous-ensemble des problèmes s'approchant le plus possible de notre problématique, la présentation d'une classification se révèle nécessaire.

2.1 Hiérarchie des problèmes d'horaires de personnel

Plusieurs façons de classifier les problèmes d'horaires peuvent être envisagées, mais la plus classique reste celle proposée par Ernst *et al.* (2004b). Ce classement est basé sur une décomposition du processus de construction d'horaires en six modules. Ces modules sont dans l'ordre :

1. la modélisation de la demande (*Demand modeling*) ;
2. la planification des jours de congé (*Days off scheduling*)
3. la création de quarts (*Shift scheduling*)
4. la construction d'horaires personnalisés (*Line of work assignment*)
5. l'affectation de tâches (*Task assignment*)
6. l'affectation d'employés (*Staff assignment*)

Cette décomposition provient, d'une part, de la complexité de s'occuper de tous les modules en une seule fois et, d'autre part, des limitations imposées par la puissance de calcul informatique actuelle. De plus cette décomposition s'intègre souvent bien au processus de création d'horaires des entreprises. En fonction du type d'entreprise, certains modules peuvent être fusionnés, ou même supprimés lorsqu'ils n'ont pas lieu d'être.

En ce qui concerne le module d'affectation de tâches, le mot tâche est aussi utilisé ici pour décrire les activités. En effet, il n'est pas rare dans la littérature que les deux termes soient confondus. En général, le module d'affectation de tâches ne gère un travail que d'une seule nature, continuel ou ponctuel, que les auteurs choisissent de dénommer activité ou tâche selon le jargon de l'entreprise. L'affectation d'activités regroupe deux cas. Le cas le plus simple consiste à considérer que les employés effectuent tous un même travail – ou des travaux différents de nature équivalente – défini comme le seul type d'activité : c'est le cas mono-activité qui s'apparente à la construction de quarts de travail. Dans un cas plus complexe, ou pour affiner le modèle de l'entreprise, le contexte multi-activités permet aux employés d'effectuer plusieurs types de travaux différents au sein de leurs quarts de travail.

Dans le cadre spécifique des entreprises de vente au détail, la démarche généralement adoptée pour établir les horaires de personnel dans une entrepise passe par la détermination :

- de la demande en activités ;
- des jours de travail de chaque employé et de ses jours de congé ;
- des quarts de travail, y compris la position des pauses durant le quart, pour chaque jour de travail et l'affectation de ces quarts aux employés ;
- des tâches et des activités à faire pour chaque quart de travail.

Cette décomposition correspond donc au module 1, au module 2, à la fusion des modules 4 et 6 et au module 5 de la décomposition de Ernst *et al.* (2004b).

En ce qui concerne la résolution, la nature des problèmes traités conditionne le plus souvent les méthodes proposées. Ces différentes méthodes emploient quasiment tous les moyens disponibles en recherche opérationnelle. En se référant au recensement de Ernst *et al.* (2004a) retranscrit dans le tableau 2.1 concernant les méthodes utilisées pour résoudre les problèmes d'horaires de personnel, la plupart des méthodes sont issues de la programmation mathématique et sont souvent des méthodes heuristiques pour lutter face à l'énorme combinatoire caractéristique des problèmes d'horaires.

Cette revue se concentre essentiellement sur les méthodes impliquant la programmation mathématique et particulièrement les heuristiques qui en sont issues. Dans l'ordre de la classification, passons d'abord en revue la construction de quarts de travail, puis nous nous focaliserons ensuite sur l'affectation proprement dite de travail interruptible correspondant à notre définition des activités pour finir avec l'affectation

Méthodes	**Articles**
Programmation en nombres entiers	139
Heuristique constructive	133
Modèle de partitionnement d'ensemble	72
Modèle de recouvrement d'ensemble	58
Génération de colonnes sans contraintes d'intégrité	48
Programmation logique avec contraintes	46
Recherche locale simple	39
Modèle de flots	38
Modèle de couplage	36
Programmation linéaire	35
Autres méthodes non citées	35
Relaxation lagrangienne	32
Modèle de file d'attente	32
Simulation	31
Génération de colonnes avec contraintes d'intégrité	30
Algorithmes génétiques	28
Autres méthodes de programmation mathématique	27
Recuit simulé	20
Programmation multi-objectifs	19
Programmation dynamique	17
Recherche tabou	16
Système expert	15
Séparation et évaluation spécialisées	14
Énumération	13
Autres métaheuristiques	11
Méthodes de plans coupants	9
Construction itérative aléatoire	5
Méthodes évolutives	4

TABLEAU 2.1 – Transcription du recensement de Ernst *et al.* (2004a)

de travail non interruptible pour ce qui a trait aux tâches.

2.2 Construction de quarts de travail

Historiquement, la construction de quarts de travail a été le premier problème de planification d'horaires de personnel traité dans la littérature relative à la recherche opérationnelle. La construction de quarts est intéressante, car elle peut être

vue comme un cas où l'on aurait un seul type d'activité, donc une seule courbe de demande à couvrir. Abordé en premier par Edie (1954), Dantzig (1954) a traité le sujet en le formulant pour la première fois comme un problème de recouvrement. Néanmoins la formulation de Dantzig a l'inconvénient d'être difficile à résoudre car le nombre de variables croît très rapidement en fonction de la taille des instances et, surtout, en fonction des types de quarts et du placement des pauses. Par conséquent, de nombreuses recherches ont été ensuite faites sur le sujet pour faciliter la résolution de cette formulation et éviter la combinatoire induite par les pauses en rendant leur placement implicite.

Segal (1974) a, par exemple, proposé un algorithme basé sur deux modèles de flots consistant à modifier dynamiquement la demande en employés en fonction du temps, grâce à l'affectation d'opérateurs supplémentaires. La nécessité de ces opérateurs est déterminée lors du placement des pauses dans des quarts entièrement travaillés jusqu'à date. Malheureusement, cette méthode de résolution peut difficilement être étendue au cas multi-activités car elle est basée sur une énumération des différents quarts de travail dans lesquels on ne considère pas les pauses. Une telle énumération explicite n'est pas envisageable lorsqu'on doit considérer notamment les durées minimales et maximales ainsi que des coûts de transitions entre activités distinctes.

Henderson et Berry (1976) se sont penchés sur le problème de construction journalière de quarts dans le cadre des opérateurs téléphoniques. Ils ont proposé deux heuristiques basées sur le modèle de Dantzig (1954). La première est une sélection itérative de quarts utilisés dans ce modèle grâce à la maximisation d'une distance par rapport à la couverture de la demande. La deuxième est une procédure permettant de reconstituer une solution réalisable à partir de la relaxation linéaire d'une restriction du modèle fournie par la première heuristique. Ces deux heuristiques sont comparées à des méthodes équivalentes aléatoires sur différents jeux de tests. Les performances de la méthode sont a priori bonnes sur des petits jeux de données mais deviennent plus discutables lorsque le nombre de quarts augmente. Sachant que notre horizon est d'une semaine et que les quarts contiennent aussi des activités, les chances qu'a cette méthode de bien performer pour le cas multi-activités sont minces.

Moondra (1976) a proposé une méthode de représentation implicite des débuts de quarts et leur longueur dans un programme linéaire en nombres entiers. Deux types de quarts sont considérés : des quarts pour employés à temps plein avec deux possibilités

de placement de pauses et des quarts pour employés à temps partiel ne contenant pas de pauses. Pour ne pas augmenter le nombre de variables, tout en considérant une certaine flexibilité, l'auteur fait l'hypothèse que la moitié des employés à temps plein effectuent les quarts avec la pause en première position et l'autre moitié effectuent les quarts avec la pause en deuxième position. Ce modèle fournit une grande flexibilité dans le choix des débuts de quarts et de leur longueur mais se prête mal à notre contexte étant donné la longueur variable des quarts.

Keith (1979) a développé une procédure heuristique basée sur la relaxation linéaire d'un modèle dans lequel seule une des combinaisons permises d'affectations de pauses était incluse initialement. Les ajustements sur les pauses sont faits durant la phase heuristique pour obtenir de bonnes solutions, à l'aide du concept de sous-couverture et sur-couverture. Le concept de sous-couverture est aussi utilisé pour juger la qualité de nos solutions et consiste à se laisser la possibilité d'avoir un manque en employés durant une période.

Gaballa et Pearce (1979) ont introduit une formulation en nombres entiers qui modélise la flexibilité dans le placement des pauses, par le biais d'une variable de pause indépendante pour chaque période de chaque quart pour laquelle une pause est permise. Néanmoins, le nombre de variables entières du programme linéaire est largement supérieur à celui d'un modèle de recouvrement équivalent. Pour compenser cette faiblesse, Bechtold et Jacobs (1990) ont proposé un modèle implicite avec affectation flexible des pauses d'un seul type aux quarts de travail, par le truchement de variables représentant le nombre de pauses pour chaque période. Des contraintes permettent d'assurer que chacun des quarts présents dans la solution optimale peut se voir attribuer une pause dans une fenêtre de temps intrinsèque. D'autres contraintes garantissent qu'exactement une pause est disponible pour chaque employé. Grâce à cette représentation implicite des solutions, la matrice du programme linéaire est plus petite et moins dense que celle d'un modèle de recouvrement. La généralisation au contexte multi-activités pose problème car c'est la résolution du modèle qui détermine à quel moment l'employé est en pause durant son quart, ce qui rend très difficile l'énumération des quarts par rapport aux différentes activités.

Thompson (1995) a présenté un modèle en nombres entiers doublement implicite pour la construction de quarts de travail, dans lequel la flexibilité se trouve à la fois sur les débuts de quarts, sur les longueurs de quarts et sur le placement de pauses. Ce

modèle combine celui de Moondra (1976) et de Bechtold et Jacobs (1990) en créant des variables de début de quart et de pause repas, représentant implicitement les quarts et liées par des contraintes rendant les quarts cohérents. De plus, le modèle permet la gestion des heures supplémentaires et des pauses de repos. Ce modèle est comparé avec celui de Bechtold et Jacobs (1990) et se montre plus rapide pour résoudre les 588 instances de tests et des problèmes avec plus de possibilités de quarts.

Aykin (1996) a généralisé le modèle de Dantzig dans le cas de pauses multiples définies à l'intérieur de fenêtres de temps. Le modèle proposé fait intervenir une variable pour chaque période où peut commencer une pause dans un quart, et permet de tenir compte de plusieurs types de pauses. Cette formulation n'a pas certaines restrictions du modèle de Bechtold et Jacobs (1990) et nécessite beaucoup moins de variables que la formulation de Dantzig (1954) lorsque les quarts peuvent contenir plusieurs pauses. Bechtold et Jacobs (1996) ont prouvé que les deux formulations sont équivalentes en nombres entiers sous certaines conditions. Toujours est-il que la formulation d'Aykin (1996), tout comme celle de Bechtold et Jacobs (1990) ou de Thompson (1995), est difficilement extensible au contexte multi-activités, car elle est en fait équivalente à une formulation implicite et ne permet donc pas d'énumérer facilement les quarts contenant des activités.

Rekik *et al.* (2004) ont proposé des extensions aux deux modèles, en abordant par exemple le concept de pauses décomposables. Une reformulation du modèle d'Aykin (1996), par l'introduction de variables de flot provenant d'un graphe biparti associant les pauses aux fenêtres de temps, conduit au modèle implicite de Bechtold et Jacobs (1990) et permet de prouver que les relaxations linéaires des deux modèles sont équivalentes en terme de gap d'intégrité. Au même moment, Çezik et Günlünk (2004) ont renforcé la preuve d'équivalence lorsque les pauses respectent certaines conditions. En plus des extensions au modèle de Bechtold et Jacobs (1990) ainsi qu'au modèle de Aykin (1996), Rekik *et al.* (2004) ont aussi proposé un modèle pour le problème de construction intégrée de cycles et quarts de travail, très flexible et s'adaptant facilement à différents environnements de travail. Malheureusement, comme la formulation de ce modèle est une extension du modèle implicite de Bechtold et Jacobs (1990), sa généralisation au contexte multi-activités est compromise.

A l'image des extensions des travaux de Rekik *et al.* (2004), une grande tendance s'est dessinée dans la construction de quarts : c'est l'intégration à celle-ci de la

construction des cycles de travail, c'est-à-dire, la détermination pour chaque employé de ces jours de congés, de repos et de travail. Beaucoup d'heuristiques ont été proposées que ce soit pour un horizon continu (Burns et Koop, 1987; Easton et Rossin, 1991) ou non continu (Jarrah *et al.*, 1994). Les modèles exacts sont souvent implicites, notamment avec des débuts de quarts variables (Jacobs et Brusco, 1996; Brusco et Jacobs, 2000) ou dont l'horizon d'une semaine doit forcément contenir deux jours de repos consécutifs (Topaloglu et Ozkarahan, 2000; Bard *et al.*, 2003).

2.3 Affectation de travail interruptible

Le cas d'affectation d'activités aux quarts dans un contexte multi-activités a déjà été traité par Omari (2002) qui a proposé une formulation comme un problème de réseau multi-flots avec variables de ressources et contraintes supplémentaires de recouvrement. Dans son modèle, les unités de flots représentent les quarts de travail et les variables de ressources modélisent l'incidence des restrictions spécifiques au problème sur les problèmes de flots sous-jacents. Après une décomposition de Dantzig-Wolfe, les sous-problèmes sont résolus par programmation dynamique et le problème maître est résolu grâce à une énumération implicite. L'horizon est divisé en tranches non disjointes pour faciliter l'obtention d'une bonne solution.

Vatri (2001) a étendu le modèle d'Omari (2002) afin de pouvoir construire des quarts personnalisés en même temps que l'affectation d'activités. En effet, il ajoute à ce modèle des composantes de construction de quarts et d'affectation de quarts aux employés et permet une flexibilité au niveau des débuts et des fins de quarts. La méthode de résolution reste cependant la même, à savoir une heuristique d'énumération implicite intégrant une approche de génération de colonnes, combinée à une stratégie de décomposition temporelle. Cette méthode permet d'obtenir des résultats légèrement meilleurs que celle utilisant des quarts fixés à l'avance.

Bouchard (2004) a proposé des heuristiques, notamment au niveau des sous-problèmes, pour accélérer la résolution du modèle mais aussi pour pouvoir ajouter des pauses à l'intérieur des quarts. La résolution est décomposée en trois étapes (construction de quarts avec pauses, assignation aux employés et assignation des activités). La méthode de résolution est basée sur les mêmes techniques que celle de Vatri (2001) mais laisse la possibilité, à chaque itération de génération de colonnes, de remettre en

cause les débuts et fins de quarts ainsi que les pauses grâce à un modèle approximatif sur les types d'activités. L'inconvénient de la méthode est le temps de calcul élevé qu'elle nécessite même en présence de stratégies d'accélération.

Bard et Wan (2006) ont traité un problème d'affectation d'activités dans un contexte où elles sont effectuées dans différentes stations de travail. Ces activités ne sont pas cependant sujettes aux contraintes de durées minimales et maximales des travaux cités précédemment. La combinaison entre un modèle en nombres entiers, basé sur un problème de multi-flots, et une recherche tabou produit des résultats jugés intéressants pour un banc d'instances de test provenant du centre de tri et de distribution des services postaux américains. Les auteurs classent ce problème comme un problème d'affectation de tâches mais leur définition correspond à celle de nos activités. Dans un article subséquent, Bard et Wan (2006) considèrent le cas où les mouvements entre les groupes de stations de travail sont restreints.

Côté *et al.* (2007) puis Côté *et al.* (2009) ont utilisé des techniques provenant de la programmation par contraintes pour réduire la taille de modèles mixtes en nombres entiers relatifs à l'affectation d'activités intégrée à la création de quarts de travail pour des employés anonymes. Un grand nombre de contraintes concernant les règles de pauses, de repos et d'allongement de quarts sont filtrés par des techniques de gestion de grammaires.

Finalement, Bouchard (2008) a proposé un modèle en nombres entiers satisfaisant parfaitement la définition de notre problème. Ce modèle utilise le concept de blocs d'activités, c'est-à-dire une affectation potentielle à un quart de travail d'une activité d'une durée donnée à une période donnée. Beaucoup de ces blocs sont filtrés pour permettre l'obtention d'une solution optimale relativement rapidement sur des instances d'une journée.

2.4 Affectation de travail non interruptible

Les notions de quarts et de tâches se retrouvent aussi dans les problèmes des horaires d'infirmières ou de rotations d'équipages. Dans ces problèmes, la demande est fonction de tâches. Ces tâches n'ont pas la même nature que celles définies dans l'introduction, mais présentent néanmoins des similitudes avec ces dernières. En effet, Ernst *et al.* (2004b) décrivent les tâches dans les problèmes de *rostering* comme

définies en termes de temps de départ et de durée, ou en termes de fenêtre de temps pour leur date de complétion, pouvant nécessiter des compétences pour être réalisées. Leur rôle dans le cas du *rostering* est de modéliser la demande en employés alors que dans notre problématique, elles devront être placées dans des quarts déjà définis au même titre que les activités.

Dans ces types de problèmes, l'affectation de tâches à des quarts de travail est nécessaire lorsque les tâches n'ont pas été encore affectées mais que l'on connaît tout de même certains quarts de travail. Souvent les tâches sont groupées pour faciliter la résolution. Les tâches ne sont pas affectées de la même façon si les elles sont fixées ou non, si les quarts de travail contiennent des pauses, si les heures supplémentaires sont autorisées ou encore si des qualifications sont requises. Les types de problèmes faisant intervenir des tâches sont souvent soit très contraints, les horaires d'infirmières par exemple, soit de très grande taille, comme dans le cas des rotations d'équipages. Pour ces problèmes, on cherche la plupart du temps une solution réalisable de bonne qualité plutôt que la solution optimale, souvent extrêmement difficile à atteindre.

Dans le cas des horaires d'infirmières, de nombreux articles ont été publiés et les méthodes proposées sont assez diversifiées, utilisant la programmation logique avec contraintes, la programmation par contraintes ou des méthodes heuristiques basées sur la programmation mathématique. Ernst *et al.* (2004b) ont recensé 103 articles sur le sujet et ont constaté que l'approche programmation mathématique est loin d'être dominante. De plus, la plupart des articles utilisant la programmation mathématique considèrent les tâches comme base de la demande en infirmières et certains articles comme Warner et Prawda (1972) font référence aux tâches seulement pour introduire la notion de qualifications. En revanche, dans les articles traitant le problème avec la programmation par contraintes, les tâches sont très similaires à celles de notre problème. Malheureusement, ces articles exploitent beaucoup le caractère sur-contraint de ces problèmes ce qui n'est pas le cas dans notre problématique.

Dans le cas des rotations d'équipages, des modèles de couverture de tâches sont souvent utilisés pour construire les horaires des pilotes d'avion, des agents de bord, des conducteurs de bus ou autres. Il s'agit de couvrir à coût minimum un ensemble de tâches avec des chemins réalisables provenant de l'ensemble des rotations possibles. Ce nombre étant immense, des solutions sont obtenues à partir d'heuristiques considérant un sous-ensemble de bonnes rotations, comme le décrit Bornemann (1970),

ou grâce à des réoptimisations séquentielles de parties du problème, comme le décrit Rubin (1973). Lavoie *et al.* (1988) ont utilisé une approche de type génération de colonnes adaptée de celle des travaux de Desrosiers *et al.* (1984), qui l'avaient appliquée au problème de tournées de véhicules. Les instances réelles étant toujours trop grandes pour être résolues rapidement, des heuristiques basées sur la réoptimisation sont toujours actuellement utilisées. Des techniques d'agrégation ont aussi été développées par Elhallaoui *et al.* (2005). Cependant, les tâches de ce type de problème ont un caractère fixe qui ne se retrouve pas dans notre problème en plus d'impliquer un déplacement dans l'espace des employés.

Une autre classe de problèmes qui fait intervenir des tâches ressemblant à celles que nous avons définies est la classe des problèmes d'ordonnancement avec contraintes de ressources. Dans ces problèmes, les tâches sont des travaux élémentaires qui ont une durée fixée et qui doivent être effectués dans une fenêtre de temps. Ici, les tâches ne sont pas interruptibles et il existe des relations de préséance entre les différentes tâches. Une tâche peut nécessiter des ressources pendant son exécution, les ressources pouvant être renouvelables ou non. Les ressources pouvant être à nouveau disponibles après l'exécution d'une tâche sont, par exemple, des travailleurs et des machines. Sans contraintes de ressources, pour de la gestion de projet par exemple, le problème devient facile car il peut être modélisé comme un problème de flot à coût minimum.

Le problème de gestion de projet avec contraintes de ressources est le problème de la classe précédente se rapprochant le plus de l'affectation de tâches dans un contexte multi-activités. Ce qui est intéressant de retenir de ces problèmes est le traitement des préséances et des ressources. La partie portant sur la minimisation du temps de complétion maximum ne se retrouve pas dans notre problématique. Des revues des problèmes d'ordonnancement avec contraintes de ressources sont données dans Özdamar et Ulusoy (1995), Brucker *et al.* (1999) et Demeulemeester et Herroelen (2002).

Davis et Heidorn (1971) proposent un algorithme de résolution basé sur la découpe des tâches en tâches de durée unitaire et sur l'ajout de contraintes de préséance immédiate en plus des contraintes habituelles de préséance. Pour sa résolution, un réseau est construit et la solution est obtenue par le calcul d'un plus court chemin dans ce réseau, grâce à la programmation dynamique et à des règles de dominance pour éliminer des arcs et des noeuds du réseau. Cette méthode de résolution a l'avantage

de permettre la consommation d'une quantité variable de ressources au cours de la tâche, mais son inconvénient majeur est l'explosion du nombre de noeuds dans le réseau lorsque les tâches sont longues par rapport à la durée unitaire.

Gorenstein (1972) utilise un graphe disjonctif pour gérer les ressources. Dans ce graphe, chaque noeud représente le début d'une tâche et les arcs représentent les préséances et les durées des tâches. Des arcs empêchent qu'une ressource identique soit utilisée par deux tâches en même temps et représentent aussi les temps de mise en place. L'algorithme proposé élimine successivement des paires d'arcs disjonctifs grâce à une relaxation des contraintes de ressources et la réalisabilité de l'horaire est vérifiée grâce à une procédure d'étiquetage similaire à l'algorithme de Ford et Fulkerson (1962) pour le problème de flot maximum. La possibilité d'avoir comme objectif la minimisation du coût de la consommation de ressources avec un temps de complétion maximal borné est discutée. Ce dernier objectif ressemble à celui associé aux tâches de notre problème. En revanche, le problème traité ne dispose pas de contraintes temporelles sur les temps de complétion.

Fisher (1973) formule le problème comme un problème de recouvrement de ressources dont les variables sont des horaires réalisables. Plusieurs projets sont traités à la fois et la disponibilité des ressources varie dans le temps. Une relaxation lagrangienne est ensuite appliquée sur les contraintes de ressources puis une reformulation en sous-problème conduit à l'obtention d'une borne inférieure pour le problème d'ordonnancement. Cette borne est ensuite améliorée par une répétition de l'ajustement des multiplicateurs de Lagrange et de la résolution des sous-problèmes correspondant à chaque projet. Une procédure d'énumération implicite permet d'obtenir une solution réalisable en branchant sur des conflits au niveau d'une ressource donnée à une période donnée, branchement modifiant le sous-problème correspondant au projet concerné par le conflit. Le fait de voir le problème comme un problème de recouvrement correspond bien à l'esprit de notre problème si l'on considère les employés qualifiés pour des tâches comme une ressource. Néanmoins, l'absence de contraintes temporelles sur les temps de complétion et l'hypothèse sur l'absence de préséance entre tâches de différents projets réduisent l'attrait de cette méthode pour notre problématique.

Patterson et Huber (1974) proposent une approche originale basée sur une formulation binaire du problème. Cette formulation fait intervenir un modèle différent de celui de Fisher (1973) puisqu'un ensemble de variables du modèle décide si une

tâche se termine à une certaine période et un autre ensemble de variables permet de déterminer le temps de complétion maximal. Dans un premier temps, une borne inférieure T au problème est calculée grâce au chemin critique et aux consommations de ressources. Ensuite, une solution réalisable de temps de complétion maximal T est cherchée. Si une telle solution existe, elle est optimale pour le problème, sinon T est augmenté d'une unité et le processus est répété. Une approche similaire, par diminution d'une borne supérieure quand le problème est réalisable, est aussi étudiée. Finalement, un algorithme de recherche binaire à l'intérieur de l'intervalle créé par les deux bornes est proposé et cette procédure est apparemment plus rapide que la résolution directe de la formulation binaire complète. L'intérêt de cette formulation est que toutes les contraintes temporelles sur notre version des tâches sont présentes. Ce modèle peut être généralisé à l'affectation de tâches aux quarts par l'ajout de variables de décision quant à l'affectation d'une tâche à un employé et par l'ajout de contraintes forçant la date de complétion d'une tâche à se situer dans le quart auquel elle a été affectée. Les contraintes de ressources peuvent être aussi modifiées pour correspondre aux qualifications des employés. L'inconvénient de cette formulation est le très grand nombre de variables si la durée unitaire est petite par rapport à l'horizon.

Talbot et Patterson (1978) proposent une formulation en nombres entiers équivalente à la précédente et une procédure d'énumération implicite pour la résoudre. L'avantage de cette nouvelle formulation est le faible nombre de variables qu'elle contient par rapport à la formulation binaire. La procédure utilise le fait que les variables soient des entiers naturels, que les tâches soient numérotées dans un ordre respectant les relations de préséance, et exploite la structure du problème. L'énumération place au plus tôt les tâches dans l'ordre de leur numérotation en respectant les contraintes de ressources. Dans le cas de non-réalisabilité de la solution en cours, la procédure revient en arrière et essaie de placer les tâches précédentes plus tard dans l'horaire. La formulation implicite est donc, elle aussi, généralisable à la présence de quarts de travail avec une formulation compacte, grâce à l'ajout de contraintes de ressources spécifiques. Les activités peuvent être modélisées par des tâches de durée unitaire, à heures fixées et sans préséance, et l'objectif pourrait être raffiné pour obtenir des problèmes plus flexibles. L'inconvénient de cette modélisation est un problème que l'on retrouve dans beaucoup de modèles concernant la gestion de projet, à savoir le grand gap d'intégrité.

Alvarez-Valdès et Tamarit (1993) ont proposé une formulation mixte avec des

variables binaires associées aux arcs d'un certain graphe et des variables entières pour les temps de début des tâches. Cette formulation est basée d'une part sur la fermeture du graphe des préséances et sur les ensembles de paires d'arcs incompatibles. Cette formulation contient des contraintes sur les préséances qui se traduisent par de simples affectations de variables, des contraintes d'élimination de cycles et des contraintes sur les paires incompatibles garantissant qu'un seul arc au moins de chaque ensemble de paires incompatibles soit dans la solution, ainsi qu'un seul arc de chaque paire incompatible. Les temps de commencement des tâches sont contraints en fonction des variables binaires grâce à la présence d'une constante ayant une grande valeur. Alvarez-Valdès et Tamarit (1993) ont calculé la dimension de l'enveloppe convexe de leur formulation et ont réalisé une étude sur ses différentes contraintes implicites d'élimination de cycles pour savoir lesquelles induisaient des facettes, afin d'améliorer la relaxation linéaire par ces contraintes et de réduire le gap d'intégrité. Les récentes recherches essaient d'obtenir des bornes inférieures encore plus serrées, en utilisant une approche de programmation mathématique ou de propagation par contraintes, ou encore en mélangeant les deux approches. Diverses extensions ont été introduites faisant intervenir d'autres types de fonctions objectif et la notion de modes multiples. Une synthèse est disponible dans Baptiste *et al.* (2005).

2.5 Contributions

Cette thèse soulève trois sujets de recherche qui ont chacun abouti à un article publié dans une revue scientifique internationale. Après avoir présenté la démarche sous-jacente à cette thèse dans la section 3.1, la première section 3.3 expose les méthodes développées pour le premier sujet de recherche. Trois modèles en nombres entiers sont proposés et des heuristiques basées sur ces modèles sont décrits. Des tests numériques ont mis en évidence la meilleure méthode de résolution. Il s'agit principalement d'une heuristique de type horizon fuyant qui résout des restrictions du problème à des tranches de temps successives, à l'aide du procédé de génération de colonnes. Les tâches individuelles sont considérées dans la section 3.4 et intégrées dans l'heuristique précédente d'horizon fuyant, grâce à une approche de décomposition en deux étapes. La première étape fait intervenir un modèle d'approximation, tandis que la deuxième est une modification de l'heuristique d'affectation d'activités dans laquelle la réaffectation des tâches devient possible pendant la résolution des

tranches temporelles. La gestion des relations de préséance au sein de cette heuristique conduit à quatre stratégies de réaffectation de tâches, qui sont comparées lors de tests numériques. La définition des tâches individuelles ne comprend pas le travail en équipe qui sera traité dans le section 3.5. Le travail en équipe ajoute la notion de patrons d'équipes et de synchronicité entre employés à la définition des tâches. Une méthode de recherche locale basée sur une approche similaire au traitement des tâches simples a été développée. La gestion de la synchronicité ainsi que celle des relations de préséance pour les tâches en équipe a été intégrée dans l'heuristique de type horizon fuyant avec génération de colonnes. La méthode de recherche locale à voisinage utilise cette méthode ainsi qu'une extension du modèle d'approximation précédent pour améliorer une solution initiale quelconque, ne contenant pas forcément des tâches en équipe déjà affectées. Une version plus efficace de la recherche à voisinage a aussi été mise en évidence dans des tests numériques.

Chapitre 3

SYNTHÈSE DU TRAVAIL

Le coeur du travail réalisé durant ce projet de doctorat a mené à la rédaction de trois articles, acceptés ou soumis pour publication dans des revues scientifiques internationales. Ces articles rédigés en anglais se retrouvent en annexe. L'objet de ce chapitre est de détailler la démarche scientifique derrière ces trois articles et d'en résumer leur contenu. Nous décrivons dans un premier temps le mandat du projet dans la section 3.1, puis revenons dans la section 3.2 sur les différentes définitions nécessaires à la formulation de chaque problème relié aux sujets de recherche. Finalement, nous présentons les travaux réalisés pour ces trois sujets dans les sections 3.3, 3.4 et 3.5 respectivement.

3.1 Projet

Le projet initial se résumait à affecter des tâches dans un contexte multi-activités. Les tâches étaient considérées comme des événements relativement sporadiques, représentant moins de 20 pour cent de la charge totale du travail. Dans notre contexte, la démarche adoptée pour établir les horaires de personnel passe par la détermination :

- de la prévision de la demande ;
- des jours de travail de chaque employé et de ses jours de congé ;
- des quarts de travail, comprenant la position des pauses durant le quart, pour chaque jour de travail et de l'affectation de ces quarts aux employés ;
- des tâches et des activités à effectuer pendant chaque quart de travail.

C'est donc durant cette dernière phase de création d'horaires de personnel que le projet s'inscrivait. Durant cette phase, les quarts des employés sont entièrement déterminés, c'est-à-dire que le début, la fin et les pauses de chaque quart sont connus, ainsi que la demande en activités et les différentes tâches à effectuer durant l'horizon. Au lieu de commencer à traiter directement le problème d'affectation de tâches en

équipe, nous avons préféré commencer par concevoir une méthode pour affecter uniquement les activités aux quarts. Cette méthode de résolution se devait de pouvoir aussi affecter les activités à des quarts où se trouvaient déjà des tâches. Ainsi, cela nous donnait la possibilité de séparer le problème en deux et les tâches pouvaient être affectées dans un module séparé. D'un point de vue pratique, notre méthode de résolution ne changeait pas du tout au tout, s'il y avait des tâches à affecter ou non. Les tâches en équipe étant sensées être plus compliquées et plus rares que les tâches individuelles, nous avons décidé de nous consacrer dans un premier temps à l'affectation de ces dernières, puis d'essayer de généraliser la méthode aux tâches en équipe. Finalement, ce découpage correspond aux problèmes traités dans les trois articles.

3.2 Définitions

Pour décrire les différents problèmes pour lesquels nous proposons une méthode de résolution, il est nécessaire de donner quelques définitions préliminaires.

Segment de travail : Un quart de travail est composé de pauses et de segments de travail. Un segment de travail est donc une période continue de temps de travail effectif de l'employé. Quitte à améliorer le service à un endroit où ce n'est pas forcément nécessaire de le faire, un employé doit toujours être affecté à une activité ou une tâche et ne peut pas être inoccupé.

Temps consacré à une activité : Période de temps d'un quart pendant laquelle l'employé effectue une activité. Par abus de langage, ce temps est aussi appelé activité d'un employé lorsqu'il n'y a pas ambiguïté. Un employé peut consacrer à une activité un temps consécutif compris entre une durée minimale et une durée maximale. Ces durées sont considérées comme des contraintes dures du problème car elles proviennent des conventions collectives.

Qualifications : Un employé possède des qualifications qui lui permettent d'effectuer certaines activités ou certaines tâches. Lors d'un quart, un employé peut seulement consacrer du temps aux activités ou aux tâches pour lesquelles il est qualifié.

Durée de base : Pour faciliter la résolution du problème, le temps est fractionné en intervalles de taille égale à la durée de base. Toutes les données temporelles sont ainsi des multiples de cette durée de base.

Période de sur- ou sous-couverture : Pour une activité donnée, si l'horaire proposé ne comporte pas assez d'employés par rapport à la demande de cette activité pendant une période de temps, celle-ci est appelée période de sous-couverture. Chaque employé qui manque pour cette activité compte pour une période de sous-couverture. Pour assurer un service correct à la clientèle, il faudra réduire au mieux le nombre total de périodes de sous-couverture. En revanche, si le nombre d'employés apporté par l'horaire est trop élevé pendant une période de temps, cette période est appellée période de sur-couverture, et le nombre de périodes de sur-couverture est compté de façon similaire à la sous-couverture. Pour avoir des coûts de personnel convenables, l'horaire doit contenir le moins possible d'employés travaillant inutilement pour une activité.

Transition : Dans un quart de travail, un employé peut en général contribuer à plusieurs activités. Une transition est définie comme l'instant où l'employé change de type d'occupation, que ce soit une activité ou une tâche. Si un employé change trop souvent de type d'occupation, ses conditions de travail ne seront plus correctes et l'employé perdra autant en confort de travail que l'entreprise y perdra en productivité.

Alors que toutes les activités sont considérées de la même façon, les tâches ont deux types de défintions :

- une tâche individuelle a une durée fixe, elle doit se terminer dans une fenêtre de temps et des relations de préséance peuvent lier son affectation avec celle d'autres tâches. Bien que la notion de temps de complétion soit présente, l'aspect d'optimisation sur les temps de complétion tel qu'il se retrouve dans les problèmes classiques d'ordonnancenement n'intervient pas ici ;
- une tâche en équipe a une durée qui dépend du nombre d'employés affectés simultanément à cette tâche. Deux employés travaillant sur une même tâche qui requiert deux heures de temps de travail ne travailleront qu'une heure chacun. Une notion de synergie est aussi introduite, à savoir que si plusieurs employés travaillent en même temps, le temps total de la tâche va décroître légèrement grâce à l'entraide entre employés. Quoiqu'il en soit, à chaque tâche est affectée différents patrons d'équipes, un patron d'équipe déterminant à la fois le nombre d'employés qui peuvent contribuer simultanément à la tâche et la durée que chaque employé doit investir dans cette tâche.

Pour chaque activité, un coût de sous-couverture (respectivement sur-couverture) est associé au nombre de périodes de sous-couverture (respectivement sur-couverture). Similairement aux activités, les tâches ont aussi des coûts de sous-couverture qui leur sont associés si elles ne sont pas affectées.

Chaque problème est défini sur un horizon d'au plus une semaine, car en général, les prévisions de la demande ne sont plus fiables au-delà. La période de discrétisation est d'une quinzaine de minutes en général. L'objectif principal des problèmes est de trouver une solution réalisable qui minimise le plus possible les coûts de sous-couverture et de sur-couverture des activités et de sous-couverture des tâches. Ensuite, le nombre de transitions doit lui aussi être minimisé. Étant donné que les deux objectifs ne sont pas réellement conflictuels et que le deuxième est secondaire, nous nous contenterons de minimiser constamment une somme pondérée de ces objectifs au lieu d'étudier leur frontière de Pareto.

3.3 Premier article : affectation d'activités

Nous résumons dans cette section les méthodes et les principaux résultats de l'article intitulé "Assigning multiple activities to work shifts" accepté pour publication dans la revue *Journal of Scheduling* et présenté dans la première annexe.

Fort des définitions de la section précédente, le problème d'affection d'activités multiples (PAAM) est énoncé de la façon suivante :

Étant donné
- un horizon divisé en périodes de temps ;
- les quarts de travail que chaque employé doit effectuer pendant l'horizon ;
- la liste des qualifications pour chaque employé concernant les activités ;
- la courbe de demande de chaque activité ; et
- les durées minimale et maximale d'affectation pour chaque activité ;

remplir chaque segment de chaque quart avec une séquence d'activités telle que
- une activité soit affectée à un segment si et seulement si l'employé travaillant ce segment est qualifié pour cette activité ;
- les durées minimale et maximale d'affectation soient respectées ;
- le coût total des sous-couvertures d'activité soit minimal ;
- le coût total des sur-couvertures d'activité soit minimal ;

– le nombre total de transitions soit minimal.

Comme dit précédemment, ces objectifs rendent le problème multi-objectifs mais nous le traiterons comme un problème de minimisation simple en hiérarchisant les objectifs par le truchement d'une somme pondérée. Nous allons tout d'abord proposer un modèle mathématique global, puis deux variations de celui-ci. Ces trois modèles ont des notations communes qu'il convient de détailler immédiatement.

Soit T l'ensemble des périodes contenues dans l'horizon. Les périodes sont toutes de même durée et sont indexées dans l'ordre chronologique. Comme les quarts de travail sont connus, nous travaillerons directement avec les segments. Ces segments sont regroupés dans l'ensemble P et chaque segment $p \in P$ s'étend sur les périodes contenues dans le sous-ensemble $T_p \subset T$. Soit A l'ensemble des activités et $d_{a,t}$ la demande en employés pour l'activité $a \in A$ pour chaque période $t \in T$. La demande en employés est connue et considérée comme constante sur chacune des périodes de T. La durée minimale d'affectation pour l'activité $a \in A$ est notée δ_a^{min} et la durée maximale est notée δ_a^{max}.

Les modèles pour le PAAM proposés dans ce chapitre sont basés sur des réseaux associés aux différents segments contenus dans l'horizon temporel. À chaque segment p correspond un réseau $\mathcal{N}^p$ qui représente implicitement les affectations potentielles d'activités à ce segment ainsi que leurs transitions. Un réseau $\mathcal{N}^p$ est composé d'une source α_p, d'un puits β_p et d'une paire de noeuds ($N^E_{a,t}$,$N^B_{a,t}$) pour chaque activité $a \in A$ pour chaque période $t \in T_p$ [1]. Pour chaque paire, le noeud $N^E_{a,t}$ représente la fin d'une affectation potentielle et le noeud $N^B_{a,t}$ représente le début d'une affectation potentielle. Il existe quatre types d'arcs dans un réseau $\mathcal{N}^p$. Les arcs d'activité relient un noeud N^B_{a,t_i} à un noeud N^E_{a,t_j} de telle sorte que $t_j - t_i \in [\delta_a^{min}, \delta_a^{max}]$. Un arc de transition relie un noeud N^E_{a,t_i} à un noeud N^B_{a,t_j}, t_i et t_j étant des périodes successives. Finalement, les arcs de début de segment relient la source à tous les noeuds N^B_{a,t_1} au puits si t_1 est la première période du segment p et les arcs de fin de segment relient tous les noeuds $N^E_{a,t_{q_w}}$ si t_{q_w} est la dernière période du segment p. Chaque chemin allant de la source α_p au puits β_p correspond à une séquence d'activités réalisable. Les arcs n'appartenant pas à un tel chemin sont retirés du réseau par un algorithme d'étiquetage approprié.

1. La notation utilisée dans les trois articles n'est pas uniforme. Pour faciliter la lecture de ce chapitre, nous avons choisi d'utiliser une notation uniforme qui peut différer avec celle des articles. Par exemple, les noeuds $N^E_{a,t}$ et $N^B_{a,t}$ sont notés $B^w_{a,t}$ et $E^w_{a,t}$, avec $w \in P$

En considérant ces types d'arcs, la notation pour le premier modèle repose sur les ensembles suivants :

$\mathcal{V}^p$: ensemble de tous les sommets de $\mathcal{N}^p$;

$\mathcal{V}_B^{p,t}$: sous-ensemble des sommets représentant un début d'activité à la période $t \in T_p$;

$\mathcal{V}_E^{p,t}$: sous-ensemble des sommets représentant une fin d'activité à la période $t \in T_p$;

$\mathcal{B}^p$: ensemble des arcs de $\mathcal{N}^p$;

$\mathcal{B}_A^p$: sous-ensemble des arcs d'activité de $\mathcal{N}^p$;

$\mathcal{B}_\tau^p$: sous-ensemble des arcs de transition de $\mathcal{N}^p$;

$\mathcal{B}_+^p(v)$: sous-ensemble des arcs ayant $v \in \mathcal{V}^p$ pour origine ;

$\mathcal{B}_-^p(v)$: sous-ensemble des arcs ayant $v \in \mathcal{V}^p$ pour destination.

3.3.1 Modèle multi-flots avec contraintes additionnelles

Le premier modèle s'apparente à un modèle multi-flots avec contraintes additionnelles. Pour chaque période $t \in T$ et chaque activité $a \in A$, la variable positive $U_{a,t}$ (resp. $O_{a,t}$) de coût $\underline{c}_{a,t}$ (resp. $\overline{c}_{a,t}$) représente le nombre de sous-couvertures (resp. sur-couvertures) pour l'activité a à la période t. De plus, le coefficient $\kappa_{a,t}^b$ vaut 1 si l'arc d'activité $b \in \mathcal{B}_A^p$ pour le segment p contribue à l'activité a à la période t. A chaque arc d'activité $b \in \mathcal{B}_A^p$ est d'ailleurs associée une variable binaire X_b^p indiquant le flot circulant sur b et à chaque arc $b \in \mathcal{B}^p \setminus \mathcal{B}_A^p$ est associée une variable de flot binaire Y_b^p. Le coût d'une transition est c_τ.

En considérant cette notation, le problème d'affectation d'activités multiples peut se formuler comme le modèle multi-flots suivant :

$$\underset{U,O,X,Y}{\text{Minimiser}} \quad \sum_{a\in A}\sum_{t\in T}(\overline{c}_a O_{a,t} + \underline{c}_a U_{a,t}) + \sum_{p\in P}\sum_{b\in\mathcal{B}_\tau^p} c_\tau Y_b^p \tag{3.1}$$

sujet à :

$$\sum_{p\in P}\sum_{b\in\mathcal{B}_A^p} \kappa_{a,t}^b X_b^p - O_{a,t} + U_{a,t} = d_{a,t}, \quad \forall a \in A,\ t \in T \tag{3.2}$$

$$\sum_{b\in\mathcal{B}_+^p(\alpha_p)} Y_b^p = 1, \quad \forall p \in P \tag{3.3}$$

$$\sum_{b\in\mathcal{B}^p_+(v)} Y^p_b - \sum_{b\in\mathcal{B}^p_-(v)} X^p_b = 0, \quad \forall p \in P,\ t \in T_p,\ v \in \mathcal{V}^{p,t}_E \tag{3.4}$$

$$\sum_{b\in\mathcal{B}^p_+(v)} X^p_b - \sum_{b\in\mathcal{B}^p_-(v)} Y^p_b = 0, \quad \forall p \in P,\ t \in T_p,\ v \in \mathcal{V}^{p,t}_B \tag{3.5}$$

$$\sum_{b\in\mathcal{B}^p_-(\beta_p)} Y^p_b = 1 \quad \forall p \in P \tag{3.6}$$

$$Y^p_b \in \{0,1\}, \quad \forall p \in P,\ b \in \mathcal{B}^p \setminus \mathcal{B}^p_A \tag{3.7}$$

$$X^p_b \in \{0,1\}, \quad \forall p \in P,\ b \in \mathcal{B}^p_A \tag{3.8}$$

$$O_{a,t},\ U_{a,t} \geq 0, \quad \forall a \in A, t \in T \tag{3.9}$$

où chaque flot dans un réseau représente une séquence d'affectations pour le segment correspondant.

L'objectif (3.1) tend à minimiser la somme pondérée des coûts de sous-couverture, de sur-couverture d'activités et de transition. Les égalités (3.2) permettent de calculer le nombre de sous-couvertures et de sur-couvertures. Les contraintes (3.3)–(3.6) sont des contraintes de chemin dans les réseaux $\mathcal{N}^p$. Finalement, les contraintes (3.7)–(3.9) définissent les domaines des variables de décisions.

La résolution de ce premier modèle donne une solution optimale au problème d'affectation d'activités multiples. Cependant, le nombre d'arcs de transition devient très grand lorsque le nombre d'activités augmente. Ainsi le modèle grossit et obtenir une solution optimale de la relaxation de ce modèle prend de plus en plus de temps. Pour accélérer la résolution de la relaxation linéaire, la décomposition de Dantzig-Wolfe (Dantzig et Wolfe, 1960) est appliquée à ce modèle et sépare le problème sous la forme d'un problème maître et de sous-problèmes, un par segment. Malgré l'énorme nombre théorique de variables, la technique de génération de colonnes donne une solution optimale pour la relaxation linéaire de façon rapide et efficace. Cette technique sera détaillée après la présentation du nouveau modèle.

3.3.2 Décomposition de Dantzig-Wolfe

La décomposition proposée repose sur les notations supplémentaires suivantes. Soit Ω_p l'ensemble total des séquences d'activités affectables au segment $p \in P$. Cet ensemble comprend un très grand nombre d'éléments. Pour chaque segment $p \in P$ et chaque séquence d'activités $q \in \Omega_p$, soit $c_{p,q}$ la somme des coûts de transitions associée

à cette séquence et soit $\kappa_{a,t}^{p,q}$ le coefficient booléen indiquant si la séquence q contient une affectation pour l'activité $a \in A$ à la période $t \in T_p$ ou non. Finalement, soit $\theta_{p,q}$ la variable de décision binaire qui indique si la séquence d'activités q sera sélectionnée dans la solution ou non. Le problème maître de la décomposition se formule alors comme le modèle suivant :

$$\underset{U,O,\theta}{\text{Minimiser}} \qquad \sum_{a\in A}\sum_{t\in T}(\overline{c}_a O_{a,t} + \underline{c}_a U_{a,t}) + \sum_{p\in P}\sum_{q\in\Omega_p} c_{p,q}\theta_{p,q} \tag{3.10}$$

sujet à

$$\sum_{p\in P}\sum_{q\in\Omega_p} \kappa_{a,t}^{p,q}\theta_{p,q} - O_{a,t} + U_{a,t} = d_{a,t}, \quad \forall a \in A, t \in T \tag{3.11}$$

$$\sum_{q\in\Omega_p} \theta_{p,q} = 1, \quad \forall p \in P \tag{3.12}$$

$$\theta_{p,q} \in \{0,1\}, \quad \forall p \in P, q \in \Omega_p \tag{3.13}$$

$$O_{a,t},\ U_{a,t} \geq 0, \quad \forall a \in A, t \in T. \tag{3.14}$$

L'objectif (3.10) consiste toujours à minimiser les coûts de sous-couverture, de sur-couverture et de transitions. Les égalités (3.11) sont équivalentes aux égalités (3.2) et servent à calculer la sur-couverture et la sous-couverture présentes dans la solution. Les contraintes (3.12) forcent l'affectation d'une et une seule séquence d'activités par segment. Les domaines des variables de décisions sont stipulés par les contraintes (3.13) et (3.14).

Le modèle (3.10)–(3.14) tel quel ne peut pas être résolu directement par un solveur de programme linéaire, sauf pour de très petites instances. Pour obtenir des solutions à la relaxation linéaire de ce modèle, la technique de génération de colonnes est donc employée. Cette technique itérative consiste à résoudre les relaxations linéaires des restrictions successives du problème maître et les instances successives des sous-problèmes, où les coûts sont modifiés à chaque itération. En ne considérant qu'un petit sous-ensembe de variables de séquences, la première étape d'une itération consiste à résoudre cette partie de modèle, appelé problème maître restreint, à l'optimalité et à en récupérer les valeurs des variables duales. La deuxième étape consiste premièrement à mettre à jour les coûts dans les sous-problèmes par rapport à la valeur de ces variables duales. Ensuite, elle consiste à chercher pour chaque sous-problème, c'est-

à-dire pour chaque segment, une variable de coût réduit négatif, dont la présence en base pourrait améliorer la valeur de la fonction objectif par rapport à celle de la solution courante du problème maître restreint. Si une telle variable existe, elle est ajoutée au sous-ensemble de variables du problème maître restreint pour la prochaine itération. Si de telles variables existent, la prochaine itération démarre avec le nouveau problème maître restreint obtenu par l'ajout de ces variables. Si aucune variable de coût réduit négatif n'est trouvée, alors la condition d'optimalité de la relaxation linéaire du problème maître global est satisfaite et la génération de colonnes se termine à cette itération. Dans notre cas, le problème maître est le modèle (3.10)–(3.14) et les sous-problèmes sont des problèmes de plus court chemin décrits ultérieurement.

La technique de génération de colonnes nécessite la résolution de nombreux problèmes maîtres restreints. Comme ceux-ci sont de petite taille, le temps de résolution total est en général inférieur à celui du modèle original pour les instances de grande taille. La résolution de chaque problème maître restreint est rendue encore plus efficace si la densité de la matrice des contraintes est diminuée par une reformulation simple, comme décrit dans Haase *et al.* (2001) et Rekik *et al.* (2009), où cette reformulation a été appliquée avec succès. Pour l'effectuer, il convient de regrouper les contraintes (3.11) par activité identique puis de les trier dans l'ordre chronologique de la demande. De longues suites consécutives de 1 apparaissent alors dans la matrice des contraintes. Remplacer chaque contrainte par la soustraction entre cette contrainte et la précédente, du moment qu'elles concernent la même activité, réduit considérablement le nombre de coefficients non nuls par variable tout en conservant l'équivalence des formulations. La nouvelle formulation obtenue est le programme linéaire mixte suivant :

$$\underset{U,O,\theta}{\text{Minimiser}} \qquad \sum_{a\in A}\sum_{t\in T}\left(\bar{c}_a O_{a,t} + \underline{c}_a U_{a,t}\right) + \sum_{p\in P}\sum_{q\in\Omega_p} c_{p,q}\theta_{p,q} \tag{3.15}$$

sujet à

$$\sum_{p\in P}\sum_{q\in\Omega_p}\left(\kappa_{a,t}^{p,q} - \kappa_{a,t-1}^{p,q}\right)\theta_{p,q} - O_{a,t} + U_{a,t} + O_{a,t-1} - U_{a,t-1} = d_{a,t} - d_{a,t-1}, \quad \forall a \in A, t \in T \tag{3.16}$$

$$\sum_{q\in\Omega_p}\theta_{p,q} = 1, \quad \forall p \in P \tag{3.17}$$

$$\theta_{p,q} \in \{0,1\}, \quad \forall p \in P, q \in \Omega_p \qquad (3.18)$$

$$O_{a,t},\, U_{a,t} \geq 0, \quad \forall a \in A, t \in T, \qquad (3.19)$$

où $\kappa_{a,0}^{p,q} = d_{a,0} = 0$ et où il n'y a pas de variable $O_{a,0}$ et $U_{a,0}$ pour les contraintes (3.16).

À partir de cette nouvelle formulation, les variables duales des problèmes maîtres restreints sont notées $\pi_{a,t}$, $a \in A$, $t \in T$ pour les contraintes (3.16) et σ_p, $p \in P$ et pour les contraintes (3.17). Le coût réduit $\tilde{c}_{p,q}$ d'une variable $\theta_{p,q}$ est alors donné par

$$\tilde{c}_{p,q} = c_\tau h_q - \sum_{a \in A} \sum_{t \in T_p} (\kappa_{a,t}^{p,q} - \kappa_{a,t-1}^{p,q}) \pi_{a,t} - \sigma_p, \qquad (3.20)$$

où h_q est le nombre de transitions dans la séquence q.

En notant a_{qr}, $r = 1, 2, \ldots, h_q + 1$, la r-ième activité effectuée dans la séquence ainsi que t_{qr}^B et t_{qr}^E les périodes, possiblement la même, auxquelles commence et finit cette activité, le coût réduit peut se reformuler sous la forme :

$$\tilde{c}_{p,q} = c_\tau h_q - \sum_{r=1}^{h_q+1} (\pi_{a_{qr}, t_{qr}^B} - \pi_{a_{qr}, t_{qr}^E + 1}) - \sigma_p, \qquad (3.21)$$

où $\pi_{a,t+1} = 0$ lorsque $t = n$ pour l'activité $a \in A$.

En se rappellant que chaque chemin entre la source α_p et le puits β_p dans le réseau $\mathcal{N}^p$ correspond à une séquence réalisable pour le segment p, trouver la séquence q correspondant à la variable de plus petit coût réduit dans Ω_p revient à trouver le plus court chemin de α_p à β_p dans le réseau $\mathcal{N}^p$ avec les coûts suivants sur les arcs :

$$\tilde{c}_b = \begin{cases} c_\tau & \text{si } b \in \mathcal{B}_\tau^p \text{ est un arc de transition} \\ -\sigma_p & \text{si } b \in \mathcal{B}_+^p(\alpha_p) \text{ est un arc de début de segment} \\ 0 & \text{si } b \in \mathcal{B}_-^p(\beta_p) \text{ est un arc de fin de segment} \\ \pi_{a,t'+1} - \pi_{a,t} & \text{si } b = (B_{a,t}^p, E_{a,t'}^p) \in \mathcal{B}_A^p \text{ est un arc d'activité.} \end{cases}$$

Comme chaque réseau $\mathcal{N}^p$ est sans cycle, ce plus court chemin est obtenable par un algorithme de propagation d'étiquettes (voir Ahuja *et al.*, 1993, Chapitre 4). À chaque itération de génération de colonnes, le plus court chemin est calculé par cette méthode et le chemin résultant est transformé en coefficients pour la nouvelle variable

de séquence s'il y a lieu.

3.3.3 Heuristiques de résolution en nombres entiers

Bien que capable d'obtenir une solution optimale à la relaxation linéaire du problème maître, la génération de colonnes ne permet pas d'obtenir une solution optimale entière sans être intégrée dans une procédure d'énumération implicite. La priorité étant le temps de calcul final, la procédure de génération de colonnes est intégrée dans des heuristiques ayant pour but d'obtenir une solution entière de bonne qualité en un temps de calcul court. Deux heuristiques ont été mise en place, la deuxième étant lancée seulement en cas d'échec de la première. La première heuristique consiste simplement à résoudre le dernier problème maître restreint obtenu lors de la résolution de la relaxation linéaire du modèle (3.10)–(3.14). Les contraintes d'intégrité sur les variables de séquence sont activées et aucune autre variable n'est ajoutée durant la procédure de séparation-évaluation. Un temps limite de calcul est imposé et si aucune solution entière de bonne qualité n'a été trouvée durant cette recherche, la deuxième heuristique prend le relais.

La deuxième heuristique est une procédure qui fixe itérativement la variable $\theta_{p,q}$ de valeur fractionnaire la plus élevée à 1. À chaque itération, une nouvelle variable de séquence est fixée à 1 et des nouvelles colonnes sont générées pour obtenir la nouvelle solution linéaire du problème. Lorsque toutes les variables ont des valeurs entières, la procédure s'arrête. Les contraintes (3.17) garantissent qu'il faudra au plus $|P|$ itérations pour obtenir une solution entière et les contraintes (3.16) assurent que la solution entière obtenue est toujours réalisable quelques soient les variables de séquence fixées, du moment qu'il n'y en ait au plus qu'une par segment.

Cette façon d'obtenir une solution entière se montre efficace sur des instances de taille dite moyenne, à savoir un jour, cinquante employés et une dizaine d'activités. Toutefois elle se montre beaucoup moins efficace sur des instances de plusieurs jours étant donné que la taille des bases du problème maître restreint grandit beaucoup plus avec le nombre de périodes ou d'activités qu'avec le nombre d'employés. Pour pallier à cette difficulté, une dernière heuristique est utilisée : il s'agit d'une méthode de type horizon fuyant. Cette méthode consiste à résoudre chronologiquement un ensemble de restrictions du modèle à des tranches de temps qui se chevauchent. Cette stratégie de décomposition temporelle est définie par deux paramètres qui sont la largeur d'une

tranche de temps et la largeur de chevauchement. Les valeurs utilisées ici sont 600 minutes pour la tranche et 150 minutes pour le chevauchement.

Chaque version du modèle, restreinte à une tranche de temps, ne met en jeu que les variables de séquence associées à des segments ayant une intersection temporelle avec la tranche, et, les variables de couverture dont la période associée est comprise dans la tranche. Toutes les affectations d'activités qui se situent dans les tranches de temps précédentes et qui se finissent avant la tranche de temps courante sont conservées, les autres sont remises en question. Chaque modèle ainsi obtenu est résolu à l'aide de la double heuristique décrite précédemment. Cette dernière heuristique constitue notre méthode finale de résolution du PAAM.

Il faut aussi noter qu'un quatrième modèle a été proposé par Bouchard (2008). Il peut être vu comme une version du modèle multi-flots dans lequel les variables d'arcs $b \in \mathcal{B}^p \setminus \mathcal{B}^p_A$ ont été supprimées. Des contraintes supplémentaires concernant les affectations contiguës d'arcs de la même activité garantissent la validité des solutions. Ce modèle est appelé modèle par blocs, car chaque affectation potentielle d'activités est appellée bloc d'activité. Cette dénomination sera reprise par la suite lorsque l'affectation des tâches sera traitée. Pour comparer de façon raisonnable la résolution de ce modèle avec les méthodes de résolutions présentées précédemment, une heuristique basée sur une réduction du nombre de variables d'activités a été implémentée de telle sorte à pouvoir mettre en regard l'horizon fuyant avec génération de colonnes à un horizon fuyant basé sur ce modèle.

3.3.4 Résultats numériques

Des tests numériques ont été réalisés sur des instances générées aléatoirement. N'ayant pas d'instances réelles à disposition, ces instances ont été créées de façon à ressembler dans la mesure du possible aux horaires d'employés d'un magasin à grande surface. Plusieurs jeux de tests ont été créés pour évaluer la performance de la méthode à plusieurs échelles. Les caractérisques principales des quarts de travail dans ces instances sont les suivantes. La longueur moyenne des quarts est de 8 heures et ceux-ci comportent une pause d'une demi-heure pour les repas. Chaque jour, environ 80 pour cent des employés travaillent de jour et le reste de nuit. La durée minimale d'affectation est d'environ une heure pour toutes les activités et la durée maximale d'affectation est de trois heures. Les employés sont qualifiés en moyenne pour 75 pour

cent des activités. Pour toutes les instances, une transition coûte $c_\tau = 15$ et les coûts de sous-couverture sont $\underline{c}_a = 1500$ par période et ceux de sur-couverture sont $\bar{c}_a = 150$ par période, quelque soit l'activité. Cinq classes d'instances ont été définies avec les caractérisques décrites dans le tableau 3.1.

Classe	Horizon(jours)	Quarts par jour	Activités
1	7	50	5
2	1	50	10
3	7	50	7
4	2	75	12
5	7	100	15

TABLEAU 3.1 – Caractéristiques des classes d'instances de tests pour le PAAM

Dans un premier temps, les instances de la classe 1 et de la classe 2 ont été résolues à l'optimalité, grâce au modèle multi-flots (*Modèle MF*) et au modèle par blocs (*Modèle Blocs*), et, de façon heuristique grâce à l'heuristique basée sur la génération de colonnes (*Gén. Col.*) et celle basée sur le modèle par blocs (*Fixe Zéro*), sans qu'elles soient intégrées dans un horizon fuyant. Les tailles des modèles utilisés sont rapportées dans le tableau 3.2.

	Modèle MF		**Modèle Blocs**		**Gén. Col.**
Instances	Variables	Contraintes	Variables	Contraintes	Contraintes
Classe 1	89,534	28,264	48,688	16,942	3,553
Classe 2	27,748	9,675	14,622	5,903	986

TABLEAU 3.2 – Taille moyenne des modèles pour les deux premières classes d'instances

Les résultats obtenus sont présentés dans le tableau 3.3 où pour chaque méthode, les moyennes pour la valeur de la fonction objectif (accompagnée par le nombre de sous-couvertures entre parenthèses) et le temps total de calcul sont indiqués. Dans ce tableau, seule l'heuristique basée sur la génération de colonnes n'obtient pas de résultats optimaux. Néanmoins ces temps de calculs sont bas pour des solutions de très bonne qualité. Comme le laissait supposer le comparatif entre les tailles des modèles, le modèle par blocs se montre plus rapide que le modèle multi-flots.

Les deux heuristiques ne sont pas arrivées à obtenir des résultats suffisamment bons sur les classes 3, 4 et 5 en des temps de calculs raisonnables. Ainsi, nous avons

Instances	Résolutions exactes				Heuristiques			
	Modèle MF		Modèle Blocs		Gén. Col.		Fixe Zéro	
	Objectif	Temps	Objectif	Temps	Objectif	Temps	Objectif	Temps
Classe 1	38400 *(20.4)*	117	38400 *(20.4)*	45	38406 *(20.4)*	52	38400 *(20.4)*	31
Classe 2	12603 *(6.4)*	900	12603 *(6.4)*	399	13605 *(7)*	60	12603 *(6.4)*	446

TABLEAU 3.3 – Résultats moyens pour les deux premières classes d'instances

utilisé la méthode d'horizon fuyant dans laquelle chaque tranche de temps était résolue soit à l'aide du modèle par blocs, par une méthode de séparation-évaluation en limitant le temps de calcul (*Horizon MB*), soit par une heuristique (*Horizon FZ*) basée sur ce même modèle ou soit par l'heuristique basée sur la génération de colonnes (*Horizon GC*). Les résultats sont présentés sous la forme de deux profils de performance en échelle semi-logarithmique. La figure 3.1 concerne la valeur de la fonction objectif et la figure 3.2 concerne les temps de calculs

Les profils de performance ont été introduits dans Dolan et Moré (2002) et la façon de les lire est la suivante. Supposons qu'un algorithme $\mathcal{A}_i$ appartenant à un ensemble d'algorithme $\mathcal{A}$ produise une statistique $u_{i,j} \geq 0$ pour le problème j d'un banc de test $\mathcal{S}$, dans le contexte où le meilleur des algorithmes parmi $\mathcal{A}$ produit la statistique la plus petite. Soit la fonction

$$\omega\left(u, u^*, \rho\right) = \begin{cases} 1 & \text{si } u \leq \rho u^* \\ 0 & \text{sinon} \end{cases}$$

définie pour n'importe quels nombres réels $u \geq 0$, $u^* \geq 0$ et $\rho \geq 1$. Le profil de performance pour l'algorithme $\mathcal{A}_i$ est la fonction

$$\pi_i(\rho) = \frac{\sum_{j \in \mathcal{S}} \omega\left(u_{i,j}, u_j^*, \rho\right)}{|\mathcal{S}|}$$

où $u_j^* = \min_{i \in \mathcal{A}} u_{i,j}$. Ainsi $\pi_i(1)$ donne la proportion d'instances pour laquelle $\mathcal{A}_i$ a été le plus efficace parmi $\mathcal{A}$ et $\pi_i(2)$ donne la proportion d'instances pour laquelle $\mathcal{A}_i$ a produit un résultat inférieur à deux fois celui du meilleur. Finalement $\lim_{\rho \to +\infty} \pi_i(\rho)$ donne la proportion d'instances que $\mathcal{A}_i$ a résolus. Afin d'améliorer la lisibilité pour de grandes plages de ρ, une échelle semi-logarithmique est utilisée. Plus une courbe $\pi_i(\rho)$

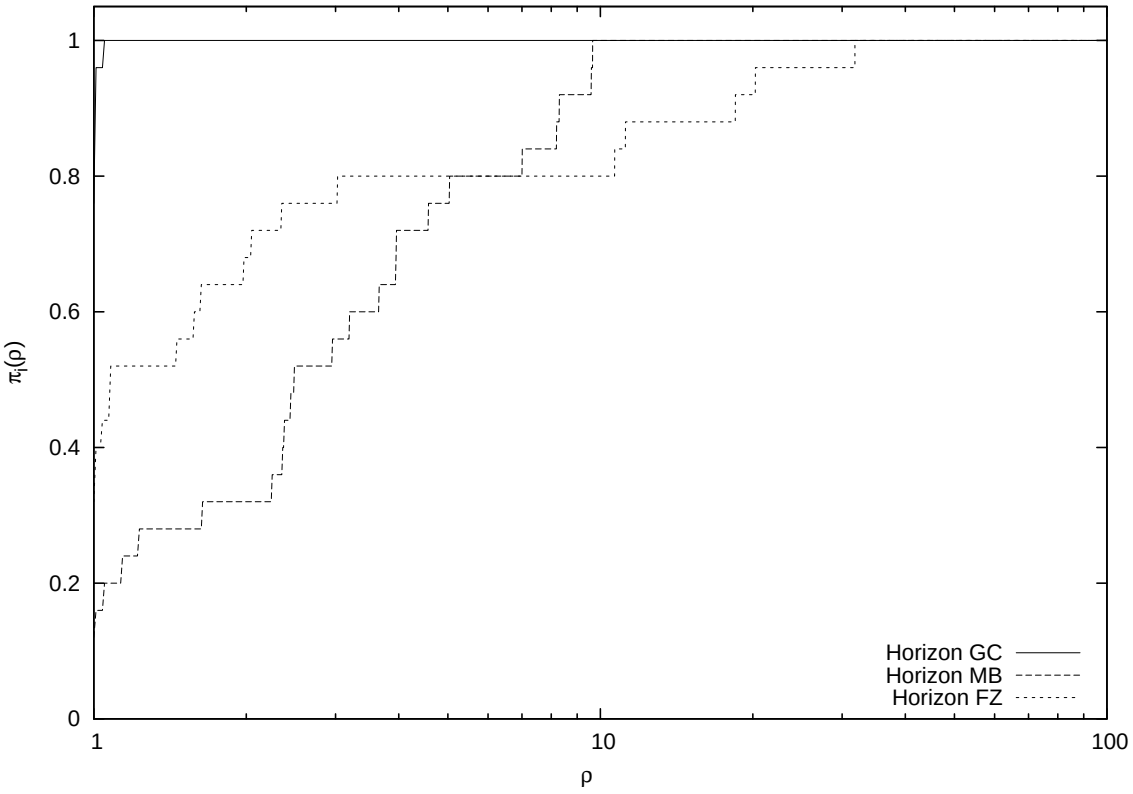

FIGURE 3.1 – Profil de performance pour la valeur de la fonction objectif

est proche du coin supérieur gauche, meilleur est l'algorithme $\mathcal{A}_i$. Sur les profils de performance des figures 3.1 et 3.2, la méthode d'horizon fuyant intégrant l'heuristique de génération de colonnes se démarque que ce soit au niveau de la valeur de la fonction objectif ou des temps de calculs.

Dans l'article décrit dans cette section, trois modèles en nombres entiers ont été proposés afin de modéliser le problème d'affectation d'activités multiples. Une heuristique pour obtenir une solution entière après la décomposition de Dantzig-Wolfe du premier modèle a été intégrée dans une procédure d'horizon fuyant et produit de meilleurs résultats que les autres méthodes de résolution intégrées dans cette même procédure.

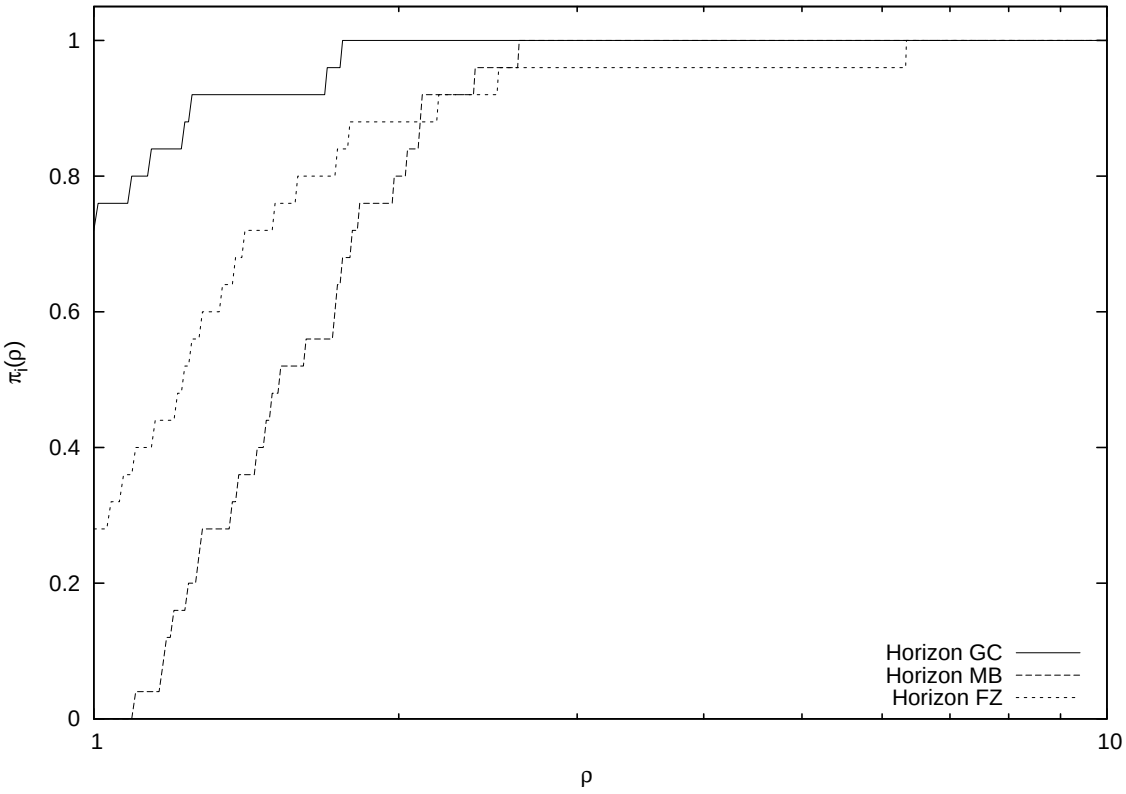

FIGURE 3.2 – Profil de performance pour le temps de calcul

3.4 Deuxième article : affectation de tâches

Dans cette section, nous présentons les principaux résultats de l'article intitulé "A two-stage heuristic for multi-activity and task assignment to work shifts", soumis pour publication dans la revue *Naval Research Logistics*, et présenté dans la deuxième annexe.

Pour traiter la notion de tâche individuelle du deuxième article, seules les définitions supplémentaires suivantes doivent être ajoutées. Rappelons qu'une tâche est un type de travail qui ne peut être interrompu et qui doit être fait par un seul employé. Elle est caractérisée par une longueur fixe et une fenêtre de temps durant laquelle elle doit être affectée. Elle peut cependant ne pas être affectée mais il s'en suit un coût élevé de sous-couverture dépendant de cette tâche. Deux tâches peuvent être reliées par une relation de préséance, formant alors un couple. Cette relation stipule

que la deuxième tâche du couple ne peut débuter que si la première est totalement complétée. Deux types de préséance peuvent être distingués. Les préseances fortes impliquent que, soit les deux tâches sont affectées, soit aucune des deux tâches n'est affectée. Une relation de préséance faible impose seulement l'affectation de la première tâche pour affecter la deuxième tâche.

La définiton du problème d'affectation de tâches et d'activités multiples (PATAM) est une extension du PAAM avec les conditions supplémentaires de réalisablités suivantes :

- à chaque période de chaque segment doit être affectée soit une activité soit une tâche ;
- un employé ne peut pas être affecté à une tâche s'il n'est pas qualifié pour l'effectuer ;
- une tâche ne peut pas être affectée en dehors de sa fenêtre de temps ;
- toutes les relations de préséance, qu'elles soient faibles ou fortes, doivent être respectées.

L'objectif est alors de minimiser les coûts du PAAM et celui des affectations de tâches. Comme montré en section 4.1, le PAAM est NP-difficile. Rajouter l'affectation de tâches à ce problème le rend encore plus difficile et nous n'avons pas trouvé de façon efficace de résoudre le problème sans le décomposer en deux phases : l'affectation des tâches a priori et l'affectation d'activités avec une réaffectation des tâches pendant le processus.

Dans l'article, nous proposons un modèle général pour le PATAM, que nous ne présentons pas ici pour des raisons de concision. Dans la version approximative de celui-ci, utilisée pour placer les tâches a priori, sont présentes des variables et des contraintes supplémentaires nécessaires à sa formulation.

3.4.1 Modèle d'approximation

Pour définir le modèle d'approximation utilisé pour affecter a priori les tâches avant d'affecter les activités, il est nécessaire de définir de nouvelles notations. Soit J l'ensemble des tâches à effectuer. Chaque tâche j a une longueur ℓ_j, un coût de sous-couverture c_j et sa première période d'affectation doit appartenir au sous-ensemble W_j de périodes contigües. Soit Γ le sous-ensemble de $J \times J$ qui représente l'ensemble

des relations de préséance entre des couples de tâches. Le sous-ensemble $\bar{\Gamma}$ contient uniquement les relations de préséance forte.

Un bloc est une affectation potentielle, que ce soit pour une tâche ou une activité. Un bloc est ainsi caractérisé par un segment, par une période de début et par une longueur, en plus de l'activité ou de la tâche qui lui correspond. L'ensemble des blocs de tâche est noté B, le sous-ensemble des blocs associés à la tâche j est noté B_j. Les blocs associés à la fois à la tâche j et au segment p appartiennent à $B_{j,p}$. Un bloc b commence à la période s_b et se termine à la période e_b.

Dans ce modèle d'approximation, les contraintes de durées minimales et maximales de contribution ont été relaxées. Pour garantir cependant qu'il est toujours possible d'affecter des activités entre deux affectations de tâches dans un même segment, soit la deuxième tâche doit débuter quand la première se finit, soit le nombre de périodes entre la fin de la première tâche et le début de la deuxième est supérieur à la plus petite durée minimale d'activités affectable au segment. Soit δ_p^{min} la durée du plus court bloc d'activité qui peut être affecté au segment p et soit P^* l'ensemble des segments p pour lesquels $\delta_p^{min} \geq 2$. Aussi, si $p \in P^*$, alors aucun bloc de tâche ou d'activité ne doit commencer à la i-ème période dans le segment p pour $i = 2, 3, \ldots, \delta_p^{min} - 1$. Un tel cas conduit à une irréalisabilité dans le processus d'affectation d'activités. La même remarque s'applique aussi à la fin du segment. Tous les blocs ne répondant pas à ces critères de réalisabilité sont exclus de l'ensemble B_p pour tout $p \in P^*$. Ces trois restrictions sur l'ensemble des blocs de tâche ne filtrent que des solutions irréalisables du point de vue des activités, mais ne les filtrent pas toutes.

La variable continue $U_{a,t}$ sert à calculer la sous-couverture pour l'activité a à la période t. La variable continue $X_{p,a,t}$ représente la fraction d'activité a affectée à la période t dans le segment p. Concernant les tâches, la variable booléenne Z_j permet de calculer la sous-couverture pour la tâche j, et pour chaque bloc $b \in B_j$, la variable Y_b indique si le bloc b fait partie de la solution ou non.

$$\underset{U,X,Y,Z}{\text{Minimiser}} \quad \sum_{j \in J} c_j Z_j + \sum_{a \in A} \sum_{t \in T} \underline{c}_a U_{a,t} \tag{3.22}$$

sujet à

$$\sum_{a\in A_p} X_{p,a,t} + \sum_{j\in J}\sum_{b\in B_{p,j}} \kappa_{b,t} Y_b = 1, \quad \forall p \in P,\, t \in T_p \tag{3.23}$$

$$\sum_{p\in P_a} X_{p,a,t} + U_{a,t} \geq d_{a,t}, \quad \forall a \in A,\, t \in T \tag{3.24}$$

$$\sum_{b\in B_j} Y_b + Z_j = 1, \quad \forall j \in J \tag{3.25}$$

$$(M + \ell_j) Z_{j'} + \sum_{b\in B_{j'}} s_b Y_b - M Z_j - \sum_{b\in B_j} s_b Y_b \geq \ell_j, \quad \forall (j,j') \in \Gamma \tag{3.26}$$

$$Z_j - Z_{j'} \geq 0, \quad \forall (j,j') \in \bar{\Gamma} \tag{3.27}$$

$$\sum_{j\in J}\sum_{\substack{b\in B_{p,j} \\ e_b = t}} Y_b + \sum_{j\in J} \sum_{\substack{b\in B_{p,j} \\ s_b \in [t+2, t+\delta_p^{min}]}} Y_b \leq 1, \quad \forall p \in P^*,\, t \in T_p \tag{3.28}$$

$$0 \leq X_{p,a,t} \leq 1, \quad \forall p \in P,\, a \in A_p,\, t \in T_p \tag{3.29}$$

$$U_{a,t} \geq 0, \quad \forall a \in A,\, t \in T \tag{3.30}$$

$$Y_b \in \{0,1\}, \quad \forall p \in P,\, j \in J,\, b \in B_{p,j} \tag{3.31}$$

$$Z_j \in \{0,1\}, \quad \forall j \in J. \tag{3.32}$$

La fonction objectif (3.22) ne concerne que la sous-couverture, que ce soit pour les activités ou les tâches. Les contraintes (3.23) assurent que les segments sont entièrement affectés. Les contraintes (3.24) servent au calcul approximatif de la sous-couverture d'activités tandis que les contraintes (3.25) servent au calcul de la sous-couverture des tâches. Le respect des relations de préséance est garanti par les contraintes (3.26) pour les préséances faibles ainsi que par les contraintes (3.27) pour les relations de préséances fortes. Pour éliminer des affectations de tâches qui conduiraient à une irréalisabilité pendant l'affectation d'activités, les contraintes (3.28) empêchent l'affectation de deux tâches entre lesquelles il n'est pas possible d'affecter un bloc. Finalement, les contraintes (3.29)–(3.32) précisent les domaines de définition des variables.

Ce modèle est dur à résoudre à l'optimalité pour des instances d'une cinquantaine de tâches. Du moins, le solveur commercial utilisé connaissait souvent des difficultés pour trouver ne serait-ce qu'une solution réalisable. En effet, les contraintes de relations de préséance sous cette forme sont très faciles à satisfaire d'un point de vue de

la relaxation linaire, mais les solutions fractionnaires obtenues durant l'arbre de branchement guident la recherche le plus souvent vers des solutions entières irréalisables. Nous avons essayé une autre forme de ces contraintes, n'utilisant pas de coefficient à grande valeur, mais l'explosion du nombre de contraintes et le faible gain au niveau du gap d'intégrité nous ont incités à conserver les contraintes sous la forme présentée dans l'article.

Pour remédier à la mauvaise orientation de la recherche d'une solution en nombres entiers, nous avons conçu une heuristique dont le but était de condenser le domaine de recherche de l'arbre de branchement. Cette heuristique consiste à favoriser certaines variables de bloc par l'intermédiaire de la modification de leurs coûts. A chaque itération, la relaxation linéaire du modèle est obtenue par une méthode dite barrière. Pour chaque tâche, les variables de bloc de valeur fractionnaire se voient attribuer un bonus. Si la valeur d'une variable de bloc reste inchangée d'une itération à l'autre alors qu'elle a reçu un bonus, son coût est remis à zéro, ainsi que celui de toutes les autres variables de bloc associées à la même tâche. Après un certain nombre d'itérations ou si la proportion de variables à valeur fractionnaire descend en dessous d'un seuil, la procédure normale d'énumération implicite prend le relais en conservant les coûts des variables de bloc issus de la dernière itération. Cette heuristique est plus efficace que la procédure normale d'énumération implicite car elle réduit rapidement le nombre de variables à valeur fractionnaire dans la relaxation linéaire et définit une instance, restreinte mollement à l'aide des coûts modifiés, plus facile à résoudre.

3.4.2 Réaffectation de tâches

Une fois ce modèle résolu, sa solution donne une affectation de tâches valide à laquelle il faut adjoindre l'affectation des activités pour obtenir une solution complète pour le PATAM. Pour chaque tâche $j \in J$ couverte dans la solution du modèle d'approximation, p_j et t_j dénotent respectivement le segment et le temps auxquels la tâche a été affectée. Réaffecter des tâches consiste à remettre en cause les décisions prises durant la résolution du modèle d'approximation et à affecter à la fois les activités et les tâches en se restreignant à un petit sous-ensemble de blocs de tâche, au lieu de considérer tous les blocs possibles. La sélection de ce sous-ensemble n'est pas unique et il est préférable qu'il possède certaines propriétés pour garantir par exemple la réalisabilité de la restriction du problème. Nous avons ainsi développé quatre stratégies

de réaffectation.

La première stratégie consiste à considérer les tâches comme définitivement affectées et affecter les activités par la méthode décrite en section 3.3. La présence de tâches déjà affectées dans des segments n'a d'influence que sur l'algorithme de génération de colonnes dans la méthode de résolution, et particulièrement, sur la structure des réseaux des sous-problèmes. La formulation du problème maître (3.10)–(3.14) reste globalement la même, à l'exception de l'ensemble Ω_p, $p \in P$ qui dénote maintenant les séquences réalisables d'activités et de tâches. Ces séquences sont générées par un réseau modifié pour chaque segment p. Soit $J_p = \{j \in J | p_j = p\}$ l'ensemble des tâches affectées au segment p. Ces affectations de tâches s'étendent sur le sous-ensemble de périodes $I_p = \bigcup_{j \in J_p} \{t_j, t_j + 1, \ldots, t_j + \ell_j - 1\}$ inclus dans T_p. Le réseau du segment p contient alors, pour toute tâche $j \in J_p$, un arc appelé arc de tâche qui relie un noeud de début de tâche vers un noeud de fin de tâche. Ces nouveaux noeuds sont reliés aux autres noeuds du réseau suivant les mêmes principes. De plus, tous les arcs d'activités chevauchant une période appartenant à I_p sont retirés du réseau.

Cette première stratégie permet de trouver une affectation d'activités et de tâches au PATAM. Cependant les décisions prises concernant les tâches dans la solution du modèle (3.22)–(3.32) sont conservées telles quelles. Cette stratégie sert donc de référence à titre de comparaison. Pour atténuer les effets des mauvaises décisions, la deuxième stratégie de réaffectation de tâches est de considérer les blocs de tâche commençant au temps t_j dans toutes les séquences des segments pour lesquels l'employé correspondant est qualifié pour réaliser la tâche j. De même que pour le cas de tâches déjà affectées, seul l'algorithme de génération de colonnes a besoin d'être modifié. Dans le modèle (3.10)–(3.14), l'ensemble Ω_p désigne de nouveau l'ensemble des séquences réalisables d'activités et de tâches et les contraintes suivantes doivent être rajoutées,

$$\sum_{p \in P} \sum_{q \in \Omega_p} \kappa_j^{p,s} \theta_{p,s} = 1, \quad \forall j \in J, \tag{3.33}$$

où $\kappa_j^{p,q}$ est un coefficient booléen indiquant si la séquence q pour le segment p contient un bloc de tâche pour la tâche j. Les réseaux de segments sont modifiés de façon similaire au cas des tâches déjà affectées, à l'exception de l'élimination des arcs d'activités. De plus, le coût d'un arc de tâches durant la résolution des sous-problèmes de génération de colonnes est égal à l'opposé de la valeur courante de la variable duale associée

à la contrainte (3.33) associée à la tâche de l'arc.

Cette deuxième stratégie permet l'échange entre employés de la réalisation d'une tâche, mais ne permet pas l'ajustement temporel de l'affectation d'une tâche. La troisième stratégie est donc basée sur ce point. Pour pouvoir réaffecter une tâche sans détruire la réalisabilité globale de l'affectation des tâches, il est nécessaire de modifier toute la méthode d'affectation d'activités. Le petit sous-ensemble de bloc de tâche considéré est celui des blocs affectés au segment p_j qui permettent de respecter les relations de préséance, tout en conservant la nature de plus court chemin des sous-problèmes. Pour que toutes les relations de préséances soient respectées quelle que soit la réaffectation des tâches, un sous-ensemble de périodes, appelé fenêtre de flexibilité, est associé à chaque tâche. Il correspond aux périodes de départ autorisées pour les blocs de chaque tâche. Ces sous-ensembles sont aussi définis tels qu'un seul bloc de tâches apparaisse pour la tâche p_j dans les séquences Ω_{p_j}. Ainsi, pour chaque tâche j, sa fenêtre de flexibilité est définie par $F_j = \left\{f_j^s, f_j^s + 1, \ldots, f_j^e\right\}$, où

$$f_j^s = \max\left\{\min_{t\in W_j} t; \max_{j'\in J|(j',j)\in\Gamma}\left\lfloor\frac{t_{j'}+\ell_{j'}-1+t_j}{2}\right\rfloor; t_j - \left\lfloor\frac{\ell_j}{2}\right\rfloor\right\} \tag{3.34}$$

$$f_j^e = \min\left\{\max_{t\in W_j} t; \min_{j'\in J|(j,j')\in\Gamma}\left\lfloor\frac{t_j+\ell_j-1+t_{j'}}{2}\right\rfloor - \ell_j; t_j + \left\lfloor\frac{\ell_j-1}{2}\right\rfloor\right\}. \tag{3.35}$$

Dans ces deux formules, W_j est le sous-ensemble de périodes telles qu'un bloc de tâche commençant à cette période respecte la fenêtre de temps de complétion de la tâche j. Le premier terme dans le membre de droite correspond au respect des fenêtres de complétion. Le second terme est le plus complexe. Chaque relation de préséance est respectée dans l'affectation de tâches proposée par le modèle d'approximation. Afin de conserver cette propriété, l'objectif pour chaque couple de tâches en relation est de considérer seulement les blocs de chacune des deux tâches jusqu'à la moitié du nombre de périodes qui les séparent, dans l'affectation de tâches proposée par le modèle d'affectation. Autrement dit, pour le seul couple de tâches j et j' tel que $(j, j') \in \Gamma$, les blocs autorisés pour j sont ceux dont la dernière période d'affectation est antérieure à la période se situant au milieu des deux blocs de tâches proposés pour j et j', c'est-à-dire le milieu entre la fin du bloc pour j, précisément $t_j + \ell_j - 1$, et le début du bloc pour j', précisément $t_{j'}$. La formule du second terme du membre de droite dans les équations (3.34) et (3.35) correspond à la généralisation de ce milieu pour l'ensemble des relations de préséance d'une tâche $j \in J$. Enfin, le troisième terme

centre la fenêtre de flexibilité autour de t_j et assure qu'elle soit strictement plus petite que ℓ_j. Ainsi, si les blocs de tâches sont représentés dans les réseaux de segments de façon similaire à celle des stratégies précédentes, un algoritme de plus court chemin ne pourra pas choisir deux fois la tâche j dans la résolution du sous-problème associé à p_j, et cela, quels que soient les coûts sur les arcs de tâches.

A ce point, seule la résolution d'une tranche de temps dans l'heuristique d'horizon fuyant a été modifiée pour tenir compte de la réaffectation temporelle des tâches. Pour la troisième stratégie, cette réaffectation temporelle perturbe aussi le processus de remise en question de l'affectation des segments. Un bloc de tâche est remis en question uniquement si ce bloc a une intersection temporelle avec la fenêtre de temps et s'il existe une solution avec un bloc associé à la même tâche dans la tranche de temps. A l'exception de cette spécificité, le processus de remise en question est le même que pour les blocs d'activités et permet à une tâche affectée dans l'affectation proposée par le modèle d'approximation d'être affectée à la fin du processus d'affectation d'activités.

Finalement la quatrième stratégie est la combinaison de la deuxième et la troisième stratégie, c'est-à-dire qu'une tâche peut être effectuée par n'importe quel employé qualifié tant que le bloc respecte la fenêtre de flexibilité. Les ajustements à apporter à la méthode d'affectation d'activités sont les mêmes que pour ceux apportés pour la troisième stratégie, mis à part pour les réseaux de segments qui doivent être modifiés en fonction de la fenêtre de flexibilité de chaque tâche pour lequel l'employé est qualifié.

Il faut aussi noter que la résolution du modèle d'approximation et celle des sous-problèmes ont été accélérés. D'une part, une heuristique a été employée pour obtenir des affectations de tâches de bonne qualité par rapport au modèle d'approximation. Cette heuristique consiste à jouer avec le coûts des variables de décisions sur les blocs de tâches afin d'alléger le travail de la procédure d'énumération implicite classique. D'autre part, concernant les sous-problèmes, une réduction du nombre d'arcs de transitions, profitant du coût constant de celles-ci, permet de réduire le nombre d'opérations de l'algorithme d'étiquetage utilisé pour obtenir les solutions aux différentes instances du problème de plus court chemin intervenant dans le processus de génération de colonnes.

3.4.3 Comparaison des stratégies

Pour déterminer laquelle des stratégies est la plus performante, nous avons effectué des tests numériques sur des jeux de données générées aléatoirement de façon similaire à celle qui nous a permis de générer des instances de test pour la méthode d'affectation d'activités. Cependant, pour la comparaison des stratégies, le nombre d'employés est fixé à cinquante, la durée l'horizon est fixée à une semaine et le nombre d'activités est fixé à cinq. Les employés travaillent 35 heures semaine avec deux jours de repos et sont qualifiés en moyenne pour 60 pour cent des activités. En revanche, les caractéristiques des tâches varient d'une instance à l'autre et une dizaine d'instances pour chaque jeu de caractéristiques ont été générées. Toutes ces instances ont été résolues avec les quatre stratégies et les résultats obtenus ont été comparés de deux façons, premièrement grâce à des tableaux comme 3.4, et deuxièmement, grâce au profil de performance de la figure 3.3.

Concernant le tableau 3.4, la première colonne représente l'identifiant du groupe d'instances, les deux colonnes suivantes indiquent la valeur moyenne des paramètres que sont le nombre de tâches et le nombre de relations de préséance, toutes fortes. Les colonnes cinq et six reportent respectivement le nombre de tâches affectées et le temps de calcul en secondes pour la résolution du modèle d'approximation. Ensuite, pour chaque stratégie, la valeur de la fonction objectif et le temps de calcul de l'affectation d'activités par l'heuristique d'horizon fuyant sont reportés.

Id	**Prop.**		**App.**		**Tout de Fixé**		**Temps Fixés**		**Employés Fixés**		**Rien de Fixé**	
	$\|J\|$	$\|\Gamma\|$	T.A.	Temps	Valeur	Temps	Valeur	Temps	Valeur	Temps	Valeur	Temps
11	20	10	20	474	42066	26	36175	30	39060	29	32203	32
12	50	25	49	522	77431	22	69036	30	69645	28	59770	36
13	70	35	70	555	113925	21	103393	32	101971	28	88525	44
14	100	50	99	590	167704	20	148011	39	142818	29	118879	52
15	150	71	149	637	254356	14	225987	43	214812	30	176655	77
16	200	99	199	675	333229	14	301204	52	279594	30	226231	116

TABLEAU 3.4 – Résultats obtenus pour différents nombres de tâches.

Comme prévu, plus le nombre de tâches est élevé, plus longue est la résolution, particulièrement pour le modèle d'approximation. Ce qui était aussi à prévoir, c'est que la quatrième stratégie soit plus performante que les autres, mais aussi soit plus

sensible au nombre de tâches quant aux temps de calculs. A l'opposé, lorsque les tâches sont fixées, plus il y a de tâches, plus les réseaux sont simples et plus les temps de calculs baissent. Le nombre de tâches influence aussi le nombre de sous-couverture d'activités qui croit avec lui. Il est aussi à noter que la plupart du temps de calcul sert à résoudre le modèle d'approximation, avec plus de 90 pour cent du temps total dans certains cas.

Pour comparer les stratégies, nous avons aussi calculé les profils de performances pour toutes les instances de tests que nous avons générées, ne se limitant pas à ceux du tableau 3.4. Les autres instances proviennent de la variation de la longueur moyenne des tâches et du nombre moyen de relations de préséance. Le fonctionnement d'un profil de performance est expliqué dans la section 3.3.4. Le profil de chaque courbe avec la valeur objectif comme statistique apparaît sur le graphique de la figure 3.3.

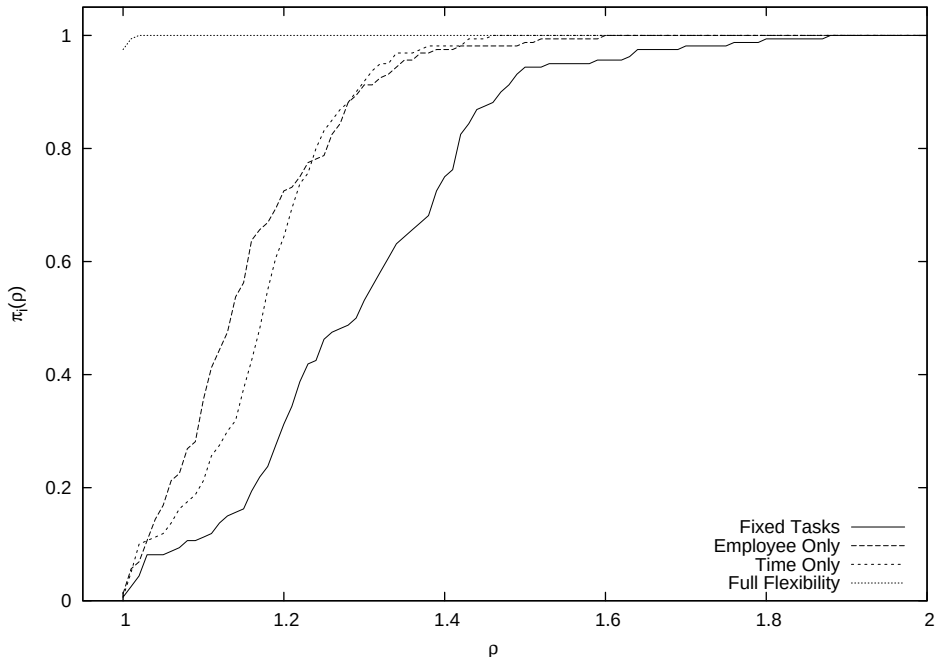

FIGURE 3.3 – Profil de performance pour les quatre stratégies.

Clairement, la quatrième stratégie se détache des autres. La première stratégie est surclassée par toutes les autres, ce qui suggère qu'il est avantageux de permettre la réaffectation de tâches pendant l'affectation d'activités pour améliorer la qualité des solutions en contrepartie d'une augmentation raisonnable des temps de calculs.

3.5 Troisième article : ajout de tâches en équipes

Dans cette section, nous présentons les principaux résultats de l'article intitulé "Assigning team tasks and multiple activities to fixed work shifts", soumis pour publication dans la revue *Journal of Heuristics*, et présenté dans la troisième annexe.

La séparation de l'affectation d'activités et de tâches en deux étapes se révèle efficace. Néanmoins, les tâches considérées ne sont effectuées que par un seul employé, ce qui n'est pas toujours le cas. Pour pallier à ce manque dans la modélisation des tâches, nous considérons un nouveau type de problème. Les tâches seront séparées en deux catégories. D'un côté, les tâches individuelles sont les tâches précédemment définies, avec un seul employé la réalisant pendant une durée fixe. D'un autre côté, les tâches en équipe peuvent être effectuées par plusieurs employés. Introduire les tâches en équipe est une extension naturelle au PATAM, mais la façon de le faire n'est pas unique. Celle que nous avons choisie est l'introduction de patrons d'équipes, notion assez générale consistant à associer à chaque tâche un ensemble de couples. Chacun est composé d'un nombre d'employés pouvant réaliser la tâche simultanément et de la durée pendant laquelle ils doivent collaborer. Le point clé ici est que les employés travaillant sur une même tâche le font simultanément. Par exemple, une tâche peut être réalisée par deux employés en trois heures ou par trois employés en deux heures. Une tâche en équipe qui ne peut pas être réalisée par un seul employé est appelée une tâche en équipe stricte.

Comme les tâches individuelles, les tâches en équipe doivent se terminer à l'intérieur d'une fenêtre de temps de complétion et doivent respecter des relations de préséance, faibles ou fortes. Pour plus de commodité, une tâche individuelle est souvent vue comme une tâche en équipe, avec comme seul patron d'équipe, le couple requièrant un employé pendant la durée totale de la tâche. Voir une tâche en équipe comme une tâche individuelle consiste à ne considérer que le patron d'équipe avec un seul employé, s'il existe.

Le nouveau problème résultant de l'introduction des tâches en équipe dans le PATAM est le problème d'affectation de tâches en équipe et d'activités multiples (PATEAM). Une solution réalisable pour ce problème doit satisfaire les contraintes des problèmes précédents avec en plus l'obligation d'affecter les tâches en équipe suivant un patron valide. Le calcul des coûts est inchangé et l'objectif reste le même.

Pour résoudre ce problème, nous avons conservé l'approche en deux étapes utilisée pour le PATAM et essayé d'intégrer cette approche en deux modules dans une heuristique d'amélioration. Le premier module consiste en une généralisation du modèle d'approximation aux tâches en équipe et le deuxième module est l'adaptation de la meilleure stratégie pour l'affectation d'activités avec réaffectation de tâches utilisée pour les tâches individuelles. Ces deux modules sont le coeur d'une heuristique à voisinages variables.

3.5.1 Premier module de l'approche

Ce nouveau type de tâches nécessite l'introduction de nouvelles notations. Le concept clé pour l'intégration des tâches en équipe dans la méthode de résolution pour les tâches individuelles est le concept de patron de tâche. Un patron de tâche est la généralisation des blocs de tâche.

Soit $\mathcal{E}$ l'ensemble des employés. Un employé $e \in \mathcal{E}$ travaille les segments appartenant à l'ensemble P_e et ainsi P est l'union sur $\mathcal{E}$ des P_e. J désigne toujours l'ensemble des tâches, qu'elles soient individuelles ou en équipe. L'ensemble des patrons d'équipe associée à une tâche $j \in J$ est noté D_j et chaque patron d'équipe $d \in D_j$ est composé d'un nombre d'employés n_j^d et d'une durée spécifique d'affectation ℓ_j^d. Etant donné que cette durée sera sûrement différente pour les patrons de D_j, l'ensemble des périodes de départ pour lesquelles la tâche j si elle est affectée suivant le patron d'équipe d finit bien dans sa fenêtre de complétion est noté W_j^d. Avec ces notations, un patron de tâche est caractérisé par :

- une tâche $j \in J$
- un patron d'équipe $d \in D_j$,
- une période de départ $t \in W_j^d$
- un sous-ensemble $E \subset \mathcal{E}$ tel que $|E| = n_j^d$ et $\forall e \in E, \exists p \in P_e, [t, t + \ell_j^d] \subset T_p$.

L'ensemble des patrons de tâche pour une tâche j et un patron d'équipe $d \in D_j$ est noté $\mathcal{H}_j^d$. Par union, $\mathcal{H}_j = \bigcup_{d \in D_j} \mathcal{H}_j^d$ est l'ensemble des patrons de tâche pour la tâche j et $\mathcal{H} = \bigcup_{j \in J} \mathcal{H}_j$ l'ensemble de tous les patrons de tâche. Pour un de ces patrons $h \in \mathcal{H}$, j_h, t_h, ℓ_h et P_h correspondent respectivement à la tâche, à la période de départ, à la durée d'affectation et au sous-ensemble de segments associés à ce patron de tâche.

Les patrons de tâche peuvent être vus comme des blocs de tâche synchronisés. Le sous-ensemble de blocs couvrant le segment $p \in P$ est noté $\mathcal{H}_p$.

Les variables de décision du nouveau modèle d'approximation sont quasiment les mêmes que pour celui des tâches individuelles. Les variables continues $U_{a,t}$ comptent la sous-couverture d'activité approximative pour l'activité a durant la période t, les variables continues $X_{p,a,t}$ remplissent les segments par des activités, les variables booléennes Z_j indiquent si une tâche n'est pas couverte et les variables booléennes Y_h indiquent si un patron de tâche h est sélectionné dans la solution. Pour un patron de tâche h, $\kappa^h_{p,t} \in \{0,1\}$ est un coefficient indiquant si le patron de tâche h est affecté au segment $p \in P$ durant la période $t \in T_p$. La formulation du modèle d'approximation pour les tâches en équipe est donc :

$$\underset{U,X,Z,Y}{\text{Minimiser}} \sum_{a\in A}\sum_{t\in T} c_a U_{a,t} + \sum_{j\in J} c_j Z_j \tag{3.36}$$

sujet à :

$$\sum_{p\in P} X_{p,a,t} + U_{a,t} \geq d_{a,t}, \quad \forall a \in A,\, \forall t \in T \tag{3.37}$$

$$\sum_{a\in A} X_{p,a,t} + \sum_{h\in\mathcal{H}_p} \kappa^h_{p,t} Y_h = 1, \quad \forall p \in P,\, t \in T_p \tag{3.38}$$

$$\sum_{h\in\mathcal{H}_j} Y_h + Z_j = 1, \quad \forall j \in J \tag{3.39}$$

$$M_{j'} Z_{j'} + \sum_{h'\in\mathcal{H}_{j'}} t_{h'} Y_{h'} - M_j Z_j - \sum_{h\in\mathcal{H}_j} (t_h + \ell_h) Y_h \geq 0, \quad \forall (j,j') \in \Gamma \tag{3.40}$$

$$Z_j - Z_{j'} \geq 0, \quad \forall (j,j') \in \bar{\Gamma} \tag{3.41}$$

$$\sum_{\substack{h\in\mathcal{H}_p \\ t_h+\ell_h=t}} Y_h + \sum_{\substack{h'\in\mathcal{H}_p \\ t_{h'}\in[t+1,t+\delta_p^{min}-1]}} Y_{h'} \leq 1, \quad \forall p \in P^*,\, t \in T_p \tag{3.42}$$

$$0 \leq X_{p,a,t} \leq 1, \quad \forall p \in P,\, a \in A,\, t \in T \tag{3.43}$$

$$U_{a,t} \geq 0, \quad \forall a \in A,\, \forall t \in T \tag{3.44}$$

$$Y_h \in \{0,1\}, \quad \forall h \in \mathcal{H} \tag{3.45}$$

$$Z_j \in \{0,1\}, \quad \forall j \in J \tag{3.46}$$

La fonction objectif tend à minimiser le coût des sous-couvertures d'activité et de tâche. Les contraintes ont les mêmes rôles que les contraintes (3.23)–(3.32). Les contraintes (3.37)–(3.39) servent aux calculs des sous-couvertures. Les contraintes (3.40) et (3.41) assurent le respect des relations de préséance. Les contraintes (3.42) filtrent des solutions irréalisables au niveau de l'affectation d'activités entre tâches. Le reste des contraintes sert à définir les domaines des variables de décision.

3.5.2 Deuxième module de l'approche

L'affectation des activités est faite par une version modifiée de l'heuristique d'horizon fuyant basée sur la génération de colonnes avec réaffectation de tâches utilisant à la fois la flexibilité sur la période de départ et sur l'employé concerné. Ce module se place dans le cadre où une affectation de tâches valide est disponible. Ce module essaie donc, à partir de cette affectation des tâches, d'affecter les activités en réaffectant les tâches, les considérant d'un point de vue des blocs de tâche. Contrairement à des tâches individuelles, les tâches en équipe doivent être synchronisées entre les employés et donc certaines modifications doivent être effectuées par rapport à la réaffectation des tâches.

Ces modifications touchent le problème maître et, dans une moindre mesure, les sous-problèmes. Ces modifications entraînent aussi un ajout au niveau de l'heuristique de branchement car de nouvelles variables binaires apparaissent. Ces variables binaires $R_{j,h}$ indiquent si le patron de tâche $h \in H_j$ est choisi pour la tâche j. Le nombre de ces variables de tâche n'était en général pas extrêmement élevé sur nos instances de tests car il était réduit par les fenêtres de flexibilité des tâches et l'influence des tranches de temps de l'horizon fuyant.

Soit $\kappa_j^{p,q}$ un coefficient binaire indiquant si une séquence q pour le segment p couvre la tâche j et $t_j^{p,q}$ un coefficient entier indiquant la période de départ de l'affectation de la tâche j à la séquence q, si elle existe, 0 sinon. En tenant compte de ces nouvelles variables et de ces nouveaux coefficients, la formulation du problème maître devient alors :

$$\underset{U,\theta}{\text{Minimiser}} \quad \sum_{a\in A}\sum_{t\in T} c_a U_{a,t} + \sum_{p\in P}\sum_{q\in\Omega_p} c_{p,q}\theta_{p,q} \tag{3.47}$$

sujet à

$$\sum_{q\in\Omega_p} \theta_{p,q} = 1, \quad \forall p \in P \tag{3.48}$$

$$\sum_{p\in P}\sum_{q\in\Omega_p} \kappa_{a,t}^{p,q}\theta_{p,q} + U_{a,t} \geq d_{a,t}, \quad \forall a \in A,\, t \in T \tag{3.49}$$

$$\sum_{h\in\mathcal{H}_j^{d_j}} R_{j,h} = 1, \quad \forall j \in J \tag{3.50}$$

$$\sum_{p\in P}\sum_{q\in\Omega_p} \kappa_j^{p,q}\theta_{p,q} = n_j^{d_j}, \quad \forall j \in J \tag{3.51}$$

$$\sum_{q\in\Omega_p} t_j^{p,q}\theta_{p,q} - \sum_{h\in\mathcal{H}_j^{d_j}:p\in P_h} t_h R_{j,h} = 0, \quad \forall j \in J, p \in P \tag{3.52}$$

$$\theta_{p,q} \in \{0,1\}, \quad \forall p \in P,\, q \in \Omega_p \tag{3.53}$$

$$U_{a,t} \geq 0, \quad \forall a \in A,\, t \in T, \tag{3.54}$$

$$R_{j,h} \in \{0,1\}, \quad \forall j \in J,\, h \in \mathcal{H}_j \tag{3.55}$$

La fonction objectif reste inchangée, de même que les contraintes (3.48) et (3.49) qui sont exactement les mêmes que les contraintes (3.12) et (3.11). Les contraintes (3.50) stipulent qu'un seul patron de tâche est sélectionné dans la solution pour chaque tâche. Les contraintes (3.51) assurent que le nombre de blocs de tâche affectés est égal au nombre d'employés du patron de tâche et ont le même rôle que les contraintes (3.33). Les contraintes (3.52) garantissent à la fois la synchronicité entre les blocs d'une même tâche et leur affectation au même sous-ensemble de segments que celui du patron de tâche sélectionné.

Les contraintes (3.50) et (3.52) introduisent de nouvelles variables duales à prendre en compte dans le sous-problème de génération de colonnes, soit ρ_j et $\mu_{j,p}$, respectivement. D'ailleurs, c'est le seul changement à effectuer et il consiste simplement à changer le coût d'un arc de tâche commençant à la période t pour la tâche j par le coût $-\rho_j - t\mu_{j,p}$.

Une fois une solution linéaire obtenue pour le problème maître grâce à l'algorithme de génération de colonnes, il reste encore à obtenir une solution entière. Dans le cas de l'affectation des activités uniquement ou celui de la réaffectation des tâches individuelles, un branchement heuristique est effectué et consiste à fixer séquentiellement la variable fractionnaire de plus haute valeur à 1, résoudre la nouvelle relaxation linéaire avec cette variable fixée et recommencer. Avec l'ajout des variables $R_{j,h}$, cette technique peut conduire à des solutions irréalisables. C'est pour cela qu'un branchement heuristique est d'abord effectué sur les variables $R_{j,h}$ puis l'ancien branchement est appliqué.

La façon de brancher sur les variables $R_{j,h}$ consiste à utiliser des bonus afin de favoriser un patron de tâche par rapport aux autres. Tant que la valeur fractionnaire de ce patron augmente, le bonus est augmenté. Si la valeur stagne, cela signifie qu'il y a de grandes chances que ce patron conduise à une irréalisabilité. Sa variable associée est donc fixée à 0, le problème maître est nettoyé des variables de séquences qui participaient à la valeur de cette variable et le sous-problème est aussi nettoyé le cas échéant. Des colonnes sont générées à chaque décision de branchement.

3.5.3 Heuristique d'amélioration

Pour obtenir de bons résultats en des temps relativement courts pour nos instances de test, nous avons dû développer une heuristique basée sur les deux modules précédents. Cette heuristique est l'adaptation de la descente à voisinages variables pour notre problème. Cette descente fait partie de la recherche à voisinages variables (Mladenović et Hansen (1997), Hansen et Mladenović (2001)) qui utilise un système de voisinages imbriqués pour améliorer séquentiellement une solution initiale. Cette métaheuristique a eu beaucoup de succès lorsqu'elle a été appliquée sur des problèmes combinatoires tels que le problème de voyage de commerce ou sur des problèmes de classification tels que le problème de la p-médiane. Récemment, cette métaheuristique a produit des résultats intéressants pour des problèmes en nombres entiers de grande taille (voir Lazić *et al.*, 2009).

L'heuristique est décrite dans l'algorithme 1 comme une fonction. Cette fonction prend comme arguments une solution initiale x au problème, une taille minimale k_{min} pour les voisinages, une taille maximale k_{max} et deux limites de temps : t_{mip} est le temps maximal accordé à la résolution des problèmes en nombres entiers et t_{vnd} est

Algorithme 1: Pseudo-code de la fonction `VND`

Entrées : x, k_{min}, k_{max}, t_{vnd}, t_{mip}
Sorties : x

```
1  t_start ⟵ cpuTime()
2  t ⟵ 0
3  v_best ⟵ ∞
4  tant que t < t_vnd et v(x) < v_best faire
5      k ⟵ k_min
6      v_best ⟵ v(x)
7      tant que t < t_vnd et k ≤ k_max faire
8          TempsLimite ⟵ min(t_mip, t_vnd − t)
9          x' ⟵ RechercheLocale( x, k, TempsLimite )
10         si v(x') < v(x) alors
11             x ⟵ x'
12             k ⟵ k_min
13         sinon
14             k ⟵ k + 1
15         fin
16         t ⟵ cpuTime() − t_start
17     fin
18 fin
```

le temps maximal accordé à la fonction complète. Une solution x a un coût noté $v(x)$ et la fonction `VND` retourne la meilleure solution apparue durant la recherche.

La fonction `VND` a besoin d'une solution initiale pour commencer la recherche. Une solution sans aucune tâche affectée peut être utilisée, mais des résultats préliminaires nous ont incités à utiliser une solution où de nombreuses tâches sont déjà affectées. La recherche bute alors moins sur les relations de préséance et produit de meilleurs résultats à temps de calcul équivalent. La détermination de cette solution comporte deux cas. Si toutes les tâches en équipe possèdent un patron d'équipe à un seul employé, alors il suffit de considérer toutes les tâches comme individuelles et utiliser l'heuristique en deux étapes développée à la section 3.4 pour trouver une bonne solution initiale. Sinon, la même heuristique est utilisée sur une instance où les tâches en équipe strictes n'apparaissent pas et la recherche à voisinages sera en charge d'affecter ces tâches.

La fonction `RechercheLocale` possède trois arguments : une solution de référence x, une taille de voisinage k et un temps limite de calcul $TempsLimite$. Elle aspire à

trouver dans le voisinage de taille k autour de x une solution qui améliore la solution courante. C'est dans cette fonction que les modules décrits dans les sections 3.5.1 et 3.5.2 sont impliqués. Le module d'affectation de tâches agit comme une fonction substitut pour tenter de trouver une meilleure affectation de tâches dans le voisinage. Cette affectation de tâches est ensuite évaluée par l'heuristique décrite dans le module d'affectation d'activités.

Donc, dans un premier temps, le modèle (3.36)–(3.46) fournit une nouvelle affectation de tâches de la façon suivante. Etant donné une solution entière correspondant à x, chaque tâche j se voit associer un poids w_j et les tâches sont triées dans l'ordre décroissant suivant ces poids. Ensuite, les variables Z_j et $Y_h, h \in \mathcal{H}_j$ sont fixées à leur valeur correspondant à x si la tâche se situe dans les $|J| - k$ premières tâches ordonnées. Les autres variables de décision sont laissées telles qu'elles sont définies dans le modèle initial. Après quoi, un solveur de programme en nombres entiers essaie de trouver la solution optimale à cette instance restreinte pendant $TempsLimite$ secondes en partant de la solution correspondant à x. La meilleure solution trouvée par le solveur est donc au pire la même que celle qu'il a reçue.

Les poids sont définis en regroupant tout d'abord les tâches par groupes selon le graphe de leur préséance. Des tâches appartiennent au même groupe si elles appartiennent à la même composante connexe de ce graphe, où chaque noeud représente une tâche et chaque arête représente une relation de préséance. Soit $\mathcal{P}$ une partition de J induite par les composantes connexes du graphe de préséance et soit $\mathcal{P}_j$ le groupe de tâches contenant la tâche j. Le poids pour la tâche j est défini par

$$w_j = \min_{j' \in \mathcal{P}_j} \left\{ \phi_{j'} \bar{T} + O_{j'} \right\} Z_j + \sum_{h \in \mathcal{H}_j} (t_h + \phi_j \bar{T}) Y_h$$

où $\bar{T} = |T| + 1$, $O_{j'}$ est la première période de départ valide pour la fenêtre de complétion de la tâche j' et ϕ_j est le nombre de fois où la tâche j a été sélectionnée pour avoir des variables de décision non fixées dans la fonction `RechercheLocale`. En suivant cette formule, le poids d'une tâche j, qui n'a pas été affectée dans x ($Z_j = 1$ et $Y_h = 0, \forall h \in \mathcal{H}_j$), est seulement fourni par le premier terme de la formule. Ce terme est égal pour toutes les tâches non affectées appartenant au même groupe que la tâche. Ainsi, il y a plus de chances de sélectionner toutes ces tâches en même temps. Au contraire, le poids d'une tâche affectée ($Z_j = 0$) est donné par le second terme de la

formule uniquement. Dans les deux cas, la formule favorise la sélection des tâches qui peuvent être effectuées le plus tôt et qui ne sont pas souvent sélectionnées pour avoir des variables libres. Pour des raisons de diversification, les nombres ϕ_j sont remis à zéro à la ligne 17 de l'algorithme 1.

L'affectation d'activités est faite à l'aide de l'heuristique de type horizon fuyant décrite dans la section 3.5.2 , avec réaffectation des tâches, qu'elles soient individuelles ou en équipe. Bien que performante pour le travail qu'elle réalise, cette heuristique n'est pas assez rapide pour le nombre de fois où elle est appelée. Pour améliorer la rapidité de la fonction `VND`, une autre fonction d'évaluation peut être choisie. Cette fonction est simplement l'affectation d'activités sans réaffectation de tâches. Le problème maître et les sous-problèmes sont grandement simplifiés et l'affectation des activités est plus rapide bien que moins efficace en terme de qualité. Cependant, plus d'itérations de la recherche peuvent être effectuées dans les mêmes temps de calcul. Un effet de bord se fait sentir. Afin d'éviter que le test d'amélioration à la ligne 9 soit trop souvent satisfait lorsque la solution est améliorée de quelques transitions, une nouvelle fonction de coût est définie. La fonction $v_2(x)$ est utilisée à la place de la fonction $v(x)$, où

$$v_2(x) = \left\lfloor \frac{v(x)}{\min\limits_{a \in A} c_a} \right\rfloor .$$

En revanche, la solution finale est obtenue par l'application de l'heuristique d'affectation d'activités avec réaffectation de tâches en équipe. Cette nouvelle mouture de la fonction `VND` est appelée FastVND.

3.5.4 Résultats

Dans un premier temps, nous avons testé comment la fonction `VND` se comportait lorsque des tâches individuelles pouvaient être scindées. Ces tests devaient mettre en évidence que scinder les tâches permet de gagner en qualité de service. Dix instances ont été utilisées pour réaliser ces tests. Dans chacune d'elles, les tâches pouvaient être effectuées soit par un seul employé, soit par deux employés. Pour le cas des tâches avec une longueur impaire, nous considérons que la synergie entre les employés réduit le nombre total de périodes de ces tâches d'une période. Par exemple, une tâche durant 6 périodes (resp. 9) lorsqu'elle est effectuée par un seul employé, dure 3

périodes (resp. 4) lorsqu'elle est effectuée par deux employés. Pour être équitable, les nombres de tâches avec et sans synergie ont été équilibrés, c'est-à-dire que la moitié des 40 tâches est considérée comme *synergétique* alors que l'autre moitié est considérée comme *régulière*. Pour ces problèmes tests, les heuristiques sont appliquées sur des instances où :

- toutes les tâches sont considérées comme individuelles(*Contrôle*) ;
- seules les tâches régulières sont considérées comme des tâches en équipe, les autres comme individuelles (*VND sans synergie*) ;
- seules les tâches synergétiques sont considérées comme des tâches en équipe, les autres comme individuelles (*VND avec synergie*).

Les paramètres de la fonction `VND` sont $t_{vnd} = 36000$ et $t_{mip} = 600$. Les paramètres k_{min} et k_{max} dépendent du solveur et du nombre de tâches. Dans notre cas, $k_{min} = 0.5\sqrt{2\,|J|} = 4$ et $k_{max} = \sqrt{2\,|J|} = 9$ dans l'optique d'incorporer les $|J|$ tâches dans la recherche si le temps limite de calcul le permet.

Le tableau 3.5 rapporte les résultats de la façon suivante. La première colonne indique l'identifiant de l'instance. Ensuite vient le nombre de sous-couvertures de la solution initiale obtenue par l'heuristique pour le PATAM ainsi que le temps de calcul utilisé pour l'obtenir. Pour chacun des trois cas détaillés ci-dessus, le nombre de sous-couvertures ainsi que le temps de calcul et le nombre d'appels à la fonction `RechercheLocale` sont donnés. De plus, le nombre de tâches scindées est aussi indiqué si des tâches sont considérées en équipe. Le meilleur nombre de sous-couverture est mis en gras.

La solution initiale a tout le temps été améliorée par les trois méthodes. Le cas *VND avec synergie* obtient de grosses améliorations par rapport à la solution initiale et sur les autres cas pour les instances 1, 2 et 6, où de nombreuses tâches ont été scindées. Le cas *VND sans synergie* obtient de bons résultats et, pour 4 instances, les meilleurs résultats en dépensant moins de temps de calcul que la version avec synergie. La version de contrôle obtient seulement une fois le meilleur résultat mais ses résultats sont assez proches des deux autres versions. Trois conclusions intéressantes peuvent être dégagées de ces résultats. Premièrement, la fonction a en général été capable d'améliorer substantiellement la solution initale, même sans tâches en équipe. Deuxièmement, autoriser des tâches scindées conduit à de meilleurs résultats, spécialement dans le cas où il y a synergie. Troisièmement, ces résultats confirment que l'heuristique pour le PATAM est très efficace par rapport à ses temps de calcul.

Id	Initiale		Contrôle			Sans synergie				Avec synergie			
	S.-c.	Temps	S.-c.	Temps	iter.	S.-c.	Temps	# Scin.	iter.	S.-c.	Time	# Scin.	iter.
1	38	128	33	2416	30	33	4136	0	28	**29**	9045	5	41
2	29	73	27	395	7	27	2285	0	22	**24**	2833	5	33
3	58	50	56	958	30	56	863	0	15	**55**	2238	1	31
4	50	53	45	1834	42	**40**	2219	4	41	46	1205	1	23
5	35	44	**31**	1078	26	32	2137	2	49	32	1667	1	33
6	67	95	53	2134	40	53	12242	2	85	**48**	24469	10	104
7	21	54	20	1909	44	16	3845	3	51	**15**	7050	2	51
8	22	59	16	437	7	**14**	3749	4	39	17	2793	2	34
9	25	72	24	391	7	**22**	2222	1	45	23	2882	3	38
10	23	72	21	893	19	**19**	3011	3	57	20	3055	3	56

TABLEAU 3.5 – Résultats pour la fonction `VND` standard

Pour voir s'il était possible d'améliorer les temps de calcul, les mêmes tests ont été effectués avec l'heuristique FastVND et les valeurs de paramètre $t_{vnd} = 1800$ et $t_{mip} = 120$. Les résultats sont présentés dans le tableau 3.6 de la même façon que dans le tableau 3.5.

Dans ce tableau, les résultats sont comparables à la version standard du VND en terme de sous-couverture. Les tendances se détachant dans le tableau 3.5 semblent être conservées et les cas avec des tâches en équipe donnent des résultats proches. Les temps de calcul ont été diminués de manière flagrante. Ces performances peuvent être expliquées par deux facteurs principaux. Le premier est que les deux méthodes sont des heuristiques d'amélioration sans processus de diversification et donc fortement conditionnées par la direction initiale de descente. Ainsi elles convergeront vers des optima locaux de façons très différentes. Deuxièmement, la fonction objectif (3.36) agit comme une fonction substitut à la fonction de coût $v(x)$ et doit lui correspondre relativement bien. Dans la méthode FastVND, la fonction d'évaluation est mieux représentée par la fonction substitut car l'influence des transitions a été grandement réduite.

Dans tous ces tests, toutes les tâches possédaient un patron avec un seul employé. D'autres tests ont été menés pour savoir si la méthode était capable de placer des tâches en équipe strictes. Ces tests ont été effectués sur des instances contenant des

Id	Initiale		Contrôle			Sans synergie				Avec synergie			
	S.-c.	Temps	S.-c.	Temps	iter.	S.-c.	Temps	# Scin.	iter.	S.-c.	Temps	# Scin.	iter.
1	38	128	33	1398	19	**32**	2049	2	22	**32**	2113	1	16
2	29	73	29	403	7	29	525	0	7	**24**	1462	4	21
3	58	50	57	516	15	**56**	427	0	7	**56**	946	0	15
4	50	53	49	720	15	**39**	1658	2	32	45	1043	2	21
5	35	44	32	1087	27	31	1268	0	31	**30**	1542	2	31
6	67	95	53	1141	21	55	1450	0	21	**52**	2023	5	24
7	21	54	21	985	22	18	1404	1	22	**17**	1641	0	19
8	22	59	15	1547	31	15	1680	1	29	**14**	2015	5	31
9	25	72	24	1219	27	23	1142	1	21	**19**	1543	3	29
10	23	72	22	858	18	**19**	1257	3	23	20	1206	3	19

TABLEAU 3.6 – Résultats pour l'heuristique FastVND

tâches pouvant être effectuées par au moins deux employés et au plus quatre employés. Dans ces instances, toutes les autres tâches sont individuelles. Dix scénarii représentés par dix instances chacun mettaient en jeu de une à dix tâches en équipe strictes parmi les 40 tâches de chaque instance, pour un total de cent instances. Les résultats sont présentés dans le tableau 3.7. La première colonne indique le nombre de tâches en équipe, puis le nombre de tâches affectées dans la solution initiale (ainsi que le temps pour l'obtenir) et le nombre de tâches affectées dans la solution obtenue par la méthode FastVND (ainsi que le temps de calcul de la méthode) sont rapportés.

Dans 98 des 100 cas, toutes les tâches ont été affectées par notre méthode dans des temps moyens de calculs inférieurs à quinze minutes. Évidemment, plus il y a de tâches en équipe, plus l'affectation prend du temps. Pour les deux autres cas, le nombre de tâches affectées par notre méthode était supérieur à celui de la solution initiale. La raison est certainement que des tâches affectées empêchent la méthode d'affecter d'autres tâches reliées par des relations de préséance. De toute façon, rien ne garantit l'existence d'une solution où toutes les tâches sont affectées durant la création des instances.

Au final, nous avons proposé une heuristique à deux modules permettant de résoudre des instances de grande taille, à savoir d'une semaine avec quarante tâches. Les tests numériques effectués ont validé la démarche et ont montré l'efficacité de la

Nombre de tâches en équipe	**Solution initiale**		**Amélioration VND**	
	Tâches	Temps	Tâches	Temps
1	37.5	37.2	40	592.6
2	33.8	29.4	38.8	603.7
3	31.3	30.2	40	677.6
4	29.2	35.6	40	735.8
5	27.2	26.0	40	802.8
6	24.8	25.8	40	748.2
7	23.5	25.2	40	827.6
8	22.1	26.5	40	810.4
9	21	25.1	39.3	800.0
10	19.7	24.7	40	852.4

TABLEAU 3.7 – Résultats pour l'affectation de tâches en équipe strictes

méthode pour affecter des tâches en équipe et pour améliorer des solutions où des tâches individuelles peuvent être scindées. Ils ont aussi validé l'efficacité de la méthode du deuxième article.

Chapitre 4

INFORMATIONS COMPLÉMENTAIRES

Deux points soulevés superficiellement dans le chapitre synthèse sont détaillés dans ce chapitre. Il s'agit de la complexité des différents problèmes abordés dans cette thèse et de l'utilisation d'une heuristique à grands voisinages.

4.1 Complexité des problèmes

Trois problèmes ont été décrits dans le chapitre 3 et un certain nombre de modèles ont été utilisés pour les résoudre. L'étude de la complexité de tous les problèmes revient à l'étude de seulement deux de ces problèmes : le problème d'affectation d'activités multiples et le problème induit par le modèle d'approximation, utilisé pour obtenir une affectation de tâches dans la section 3.4.1. Clairement, le problème d'affectation d'activités multiples est un cas particulier du problème d'affectation de tâches et d'activités multiples et a fortiori du problème d'affectation de tâches en équipe et d'activités multiples. Ainsi, si le PAAM est $\mathcal{NP}$-difficile, alors le PATAM et PATEAM le sont aussi. De même, le problème induit par le modèle d'approximation de la section 3.4.1 est un cas particulier du problème induit par l'autre modèle d'approximation pour les tâches en équipe décrit dans la section 3.5.1 et le deuxième appartiendra à la même classe de complexité que le premier. Les modèles utilisés dans le chapitre synthèse sont les suivants. Le PAAM a été formulé par le modèle multi-flots (3.1)–(3.9) et par les modèles d'énumération (3.10)–(3.14) et (3.15)–(3.19). Ce dernier modèle a été modifié et les versions apparaissant dans la section 3.4.2 formulent le PATAM, et peuvent donc formuler le PAAM puisque l'ensemble des instances du PAAM sont celles sans aucune tâche. Le deuxième problème à étudier est le problème induit par le modèle (3.22)–(3.32), dénommé problème d'affectation de tâches a priori

ou PATAP. Ce modèle est inclus dans le modèle (3.36)–(3.46), car les instances du PATAM sont les instances du PATEAM avec seulement des tâches individuelles.

4.1.1 Complexité du PAAM

Pour montrer que le PAAM est un problème $\mathcal{NP}$-difficile, il suffit de prouver qu'il est aussi difficile à résoudre qu'un problème $\mathcal{NP}$-difficile, autrement dit, il faut trouver un problème $\mathcal{NP}$-difficile qui est réductible au PAAM. Nous utilisons le problème du sac à dos non borné ($\mathcal{UKP}$) mentionné dans Kellerer *et al.* (2004). Cette variante du problème du sac à dos est $\mathcal{NP}$-difficile et s'énonce comme il suit. Étant donné un ensemble de n types d'objets numérotés de 1 à n, chaque objet de type de $i, i = 1 \dots n$ ayant un poids w_i et une valeur $v_i, i = 1 \dots n$ non négatifs, quelle est la valeur maximale d'un sac pouvant contenir un poids total de W.

Si X_i est une variable entière indiquant le nombre d'objets de type i apparaissant dans la solution optimale, le problème du sac à dos se formule par :

$$\underset{X}{\text{Minimiser}} \sum_{i=1\dots n} v_i X_i \tag{4.1}$$

sujet à

$$\sum_{i=1\dots n} w_i X_i \leq W \tag{4.2}$$

$$X_i \in \mathbb{N}, \quad i = 1 \dots n \tag{4.3}$$

Théorème 4.1. *Le problème d'affectation d'activités multiples est $\mathcal{NP}$-difficile.*

Démonstration. Montrons que n'importe quelle instance $I_{\mathcal{UKP}}$ de ce problème peut être résolue grâce à la résolution d'une instance I_{PAAM} du PAAM, obtenue en temps polynomial à partir de l'instance $I_{\mathcal{UKP}}$.

Les données de $I_{\mathcal{UKP}}$ sont le nombre n, la capacité W, les poids $w_i, i = 1 \dots n$ et les valeurs $v_i, i = 1 \dots n$.

Les données pour I_{PAAM} sont l'ensemble des périodes T, l'ensemble des segments P, l'ensemble des activités A. Pour chaque activité $a \in A$, I_{PAAM} contient aussi sa

durée minimale δ_a^{min} de contribution et sa durée maximale δ_a^{max}, ainsi que sa demande $d_{a,t}$, son coût de sous-couverture $\underline{c}_{a,t}$ et son coût de sur-couverture $\overline{c}_{a,t}$, pour toute période $t \in T$. Pour chaque segment p, I_{PAAM} contient ses qualifications $A_p \subset A$ ainsi que l'ensemble des périodes T_p sur lesquelles le segment p s'étale. A tout cela, il faut finalement ajouter le coût de transition c_τ.

Ainsi, l'instance I_{PAAM} est définie comme il suit. T contient des périodes telles que $|T| = W$, numérotées de 1 à W. A contient $n+1$ activités, numérotées de a_0 à a_n, telles que $\delta_{a_i}^{min} = \delta_{a_i}^{max} = w_i, i = 1 \dots n$ et $\delta_{a_0}^{min} = 1$ et $\delta_{a_0}^{max} = W$. Pour tout t dans T, $d_{a_i,t} = 1$, $\underline{c}_{a_i,t} = \dfrac{v_i}{\delta_{a_i}^{min}}$ et $\overline{c}_{a_i,t} = 0$ pour $i = 1 \dots n$, et $d_{a_0,t} = 0$, $\underline{c}_{a_0,t} = \overline{c}_{a_0,t} = c_\tau = 0$. P contient un seul segment de travail d'une longueur W composant le seul quart du seul employé de I_{PAAM}, qualifié pour toutes les activités. La transformation entre $I_{\mathcal{UKP}}$ et I_{PAAM} est donc facile à effectuer et peut se réaliser en temps polynomial.

Une solution optimale à cette instance du PAAM donnera une séquence d'affectations contenant des blocs d'activité. Un bloc pour l'activité a_i, $i = 1 \dots n$, représente un objet de type i, avec sa durée comme équivalent de son poids, et son impact sur la sous-couverture comme équivalent de sa valeur. La durée totale de ces blocs d'activité est inférieure à W, ce qui implique que les poids représentés par la séquence respectent la contrainte de capacité du sac. Si $u_{a,t}^*$ indique le nombre de sous-couvertures pour l'activité a à la période t et x_a^* le nombre de blocs de l'activité a apparaissant dans une solution optimale de I_{PAAM}, alors son coût est égal à :

$$
\begin{aligned}
C^* &= \sum_{a \in A} \sum_{t \in T} \underline{c}_a u_{a,t}^* \\
&= \sum_{a \in A \setminus \{a_0\}} \underline{c}_a \left(|T| - \delta_a^{min} x_a^* \right) \\
&= |T| \sum_{a \in A \setminus \{a_0\}} \underline{c}_a - \sum_{a \in A \setminus \{a_0\}} \underline{c}_a \delta_a^{min} x_a^* \\
&= |T| \sum_{a \in A \setminus \{a_0\}} \underline{c}_a - \sum_{i=1 \dots n} \frac{v_i}{\delta_{a_i}^{min}} \delta_{a_i}^{min} x_{a_i}^* = K - \sum_{i=1 \dots n} v_i x_{a_i}^*
\end{aligned}
$$

avec C^* le coût de la solution et K une constante par rapport aux variables de décisions du problème.

Comme C^* est minimal pour toutes les solutions de I_{PAAM}, $K - \sum_{i=1 \dots n} v_i x_{a_i}^*$

est minimal pour toutes les solutions de I_{PAAM} et donc $\sum_{i=1\ldots n} v_i x^*_{a_i}$ est maximal pour toutes les solutions, avec $\sum_{i=1\ldots n} \delta^{min}_{a_i} x^*_{a_i} = \sum_{i=1\ldots n} w_i x^*_{a_i} \leq W$, grâce à la durée maximale de l'unique segment de I_{PAAM}. D'où $x^*_{a_i}, i = 1 \ldots n$ est une solution optimale de $I_{\mathcal{UKP}}$.

Finalement, obtenir une solution optimale à $I_{\mathcal{UKP}}$ revient à obtenir une solution optimale de I_{PAAM}, et donc le PAAM est au moins aussi difficile que le problème du sac à dos non borné, qui $\mathcal{NP}$-difficile. □

Corollaire 4.1. *Le PATAM et le PATEA sont $\mathcal{NP}$-difficiles.*

4.1.2 Complexité du modèle d'approximation

Le problème d'affectation de tâches a priori (PATAP) peut être énoncé de la façon suivante. Étant donné

- un horizon divisé en périodes de temps ;
- les quarts de travail que chaque employé doit effectuer pendant l'horizon ;
- la liste des activités et leurs courbes de demande ;
- la durée minimale d'affectation pour chaque activité ;
- la liste des tâches avec leur longueur et leur fenêtre de complétion ;
- la liste des qualifications pour chaque employé concernant les activités et les tâches ; et
- la liste des relations de préséance ;

remplir chaque segment de chaque quart avec une séquence de tâches et de bloc d'activité de longueur unitaire telle que

- une tâche ou une activité soit affectée à un segment si et seulement si l'employé travaillant ce segment est qualifié pour celle-ci ;
- une tâche finisse dans sa fenêtre de complétion ;
- les relations de préséance soient respectées ;
- le plus court bloc d'activité affectable à un segment puisse être affecté durant les périodes où aucune tâche n'est affectée ;
- le coût total des sous-couvertures de tâche soit minimal ;
- le coût total des sous-couvertures d'activité soit minimal ;
- le nombre total de transitions soit minimal.

Comme pour le PAAM, montrer que le PATAP est un problème $\mathcal{NP}$-difficile revient à trouver un problème $\mathcal{NP}$-difficile qui est réductible au PAPP. Le problème choisi ici est le problème du sac à dos binaire ($\mathcal{BKP}$) énoncé dans Pisinger (2005). Étant donné un ensemble de n objets numérotés de 1 à n, de poids $w_i, i = 1 \dots n$ non négatifs et de valeurs $v_i, i = 1 \dots n$ non négatives également, quelle est la valeur maximale d'un sac pouvant contenir un poids total de W.

Si X_i est une variable binaire indiquant si l'objet i apparaît dans la solution optimale ou non, le problème du sac à dos se formule par :

$$\underset{X}{\text{Minimiser}} \sum_{i=1\dots n} v_i X_i \tag{4.4}$$

sujet à

$$\sum_{i=1\dots n} w_i X_i \leq W \tag{4.5}$$

$$X_i \in \{0, 1\}, \quad i = 1 \dots n \tag{4.6}$$

Ce problème est $\mathcal{NP}$-difficile et sa forme décisionnelle est même $\mathcal{NP}$-complète (voir Karp, 1972).

Théorème 4.2. *Le problème d'affectation de tâches a priori est $\mathcal{NP}$-difficile.*

Démonstration. Montrons que n'importe quelle instance $I_{\mathcal{BKP}}$ de ce problème peut être résolue grâce à la résolution d'une instance I_{PATAP} du PATAP, obtenue en temps polynomial à partir de l'instance $I_{\mathcal{BKP}}$.

Les données de $I_{\mathcal{BKP}}$ sont les mêmes que celles de $I_{\mathcal{UKP}}$.

Les données pour I_{PATAP} incluent celles de I_{PAAM} avec en plus l'ensemble J des tâches, et pour chaque tâche j, sa longueur ℓ_j, son coût de sous-couverture c_j, le sous-ensemble W_j de périodes de départ valides pour sa fenêtre de complétion et le sous-ensemble de segments où la tâche j peut être affectée.

Ainsi, l'instance I_{PATAP} est définie comme il suit. T contient des périodes telles que $|T| = W$, numérotées de 1 à W. A contient une activité a_0 telle que $\delta_{a_0}^{min} = 1$. Pour tout t dans T, $d_{a_0,t} = 0$, $\underline{c}_{a_0,t} = \overline{c}_{a_0,t} = 0$. L'ensemble J des tâches contient

n tâches, numérotées j_1 à j_n, de longueur $\ell_{j_i} = w_i, i = 1, \ldots, n$, de coût $c_{j_i} = v_i$ et de P contient un seul segment de travail d'une longueur W composant le seul quart du seul employé de I_{PATAP}, qualifié pour toutes les tâches et l'activité a_0. La transformation entre $I_{\mathcal{BKCP}}$ et I_{PATAP} est donc facile à effectuer et peut se réaliser en temps polynomial.

Une solution à cette instance du PATAP donnera une séquence d'affectations contenant des blocs d'activité et de tâche. Un bloc pour la tâche $j_i \in J$ représente l'objet i avec sa durée comme équivalent de son poids, et avec son impact sur la sous-couverture comme équivalent de sa valeur. La durée totale de ces blocs d'activité est inférieure à W ce qui implique que le poids représenté par la séquence respecte la contrainte de capacité du sac. Si z_j^* indique la sous-couverture de la tâche j apparaissant dans une solution optimale de I_{PATAP} et si x_j^* indique si la tâche j apparaît dans cette même solution, alors son coût est égal à :

$$\begin{aligned} C^* &= \sum_{j \in J} \underline{c}_j z_j^* \\ &= \sum_{i=1,\ldots,n} \underline{c}_{j_i} z_{j_i}^* \\ &= \sum_{i=1,\ldots,n} v_i \left(1 - x_{j_i}^*\right) = K - \sum_{i=1,\ldots,n} v_i x_{j_i}^* \end{aligned}$$

avec C^* le côut de la solution et K une constante par rapport aux variables de décisions du problème.

Comme C^* est minimal pour toutes les solutions de I_{PATAP}, $K - \sum_{i=1\ldots n} v_i x_{j_i}^*$ est minimal pour toutes les solutions de I_{PATAP} et donc $\sum_{i=1\ldots n} v_i x_{j_i}^*$ est maximal pour toutes les solutions, avec $\sum_{i=1\ldots n} \ell_{j_i} x_{j_i}^* = \sum_{i=1\ldots n} w_i x_{j_i}^* \leq W$ par définition de l'unique segment de I_{PATAP}. D'où $x_{a_i}^*, i = 1 \ldots n$ est une solution optimale de $I_{\mathcal{UKCP}}$.

Finalement, obtenir une solution optimale à $I_{\mathcal{BKCP}}$ revient à obtenir une solution optimale de I_{PATAP}, et donc le PATAP est au moins aussi difficile que le problème du sac à dos binaire, qui $\mathcal{NP}$-difficile. □

Corollaire 4.2. *Le problème induit par le modèle (3.36)–(3.46) est $\mathcal{NP}$-difficile.*

Dans les deux problèmes, l'étude de la compléxité s'est portée sur les activités ou les tâches. Cependant, celles-ci sont en général en nombre limité. Pour un nombre

borné d'activités, le PAAM pourrait être de nature plus facile. Toutefois, il pourrait être possible de montrer que le problème reste difficile grâce à une réduction dans un problème de *bin-packing*.

4.2 Heuristique à grands voisinages pour le PAAM

Avant de se résoudre à utiliser un simple horizon fuyant, de nombreuses méthodes d'amélioration d'une ou plusieurs solutions courantes ont été étudiées. Le choix de l'heuristique a été conditionné par plusieurs facteurs. Deux facteurs principaux sont intervenus dans le choix de l'heuristique. Le premier est que le problème se prête bien à la réoptimisation, le deuxième est qu'il se prête mal à la fusion de composantes. En effet, lorsque les composantes de deux bonnes solutions sont choisies pour en constituer une troisième, la qualité de la solution résultante est en général beaucoup moins bonne que celle des solutions originales, voire carrément mauvaise si le choix est aléatoire. Ces deux facteurs nous ont encouragé à utiliser une méthode à trajectoire. Parmi les nombreuses métaheuristiques qui existent, la recherche tabou a été choisie pour sa simplicité et sa robustesse. Elle a été adaptée à notre problème notamment pour ce qui est de la liste tabou.

4.2.1 Vers la recherche à grands voisinages

L'idée initiale pour la première méthode de résolution envisagée était une recherche tabou très simple et très rapide. Les types de mouvement considérés étaient :

- bouger une transition à droite ;
- bouger une transition à gauche ;
- supprimer une activité ; et
- insérer une nouvelle activité.

Le voisinage considéré était donc un sous-ensemble relativement grand du nombre de possibilités de remplir une pièce. La rapidité était là, la simplicité aussi, mais la convergence ne l'était pas. En effet, cette recherche ne prenait qu'une seule pièce en considération et bien que rapide, il lui fallait faire énormément d'itérations pour passer d'une bonne solution à une autre. Sur des instances considérées actuellement comme simples, cette recherche tabou arrivait loin de l'optimum. Le verdict était sans appel : les voisinages étaient de trop petite taille et manquaient donc de visibilité.

La décision d'agrandir les voisinages fut donc prise. Plusieurs options ont été étudiées, mais celle qui semblait la plus satisfaisante était de réaffecter trois segments à la fois. Pour ce faire, une énumération dynamique des possibilités, avec le calcul d'une borne inférieure pour élaguer l'arbre d'énumération, a été mise en place et la qualité des résultats a été améliorée substantiellement. En revanche, bien que la procédure implantée était assez rapide pour étudier de nombreux voisinages, elle restait relativement lente en comparaison des méthodes ultérieures. Le nombre de fois où la solution était amélioriée par rapport au nombre total d'itérations se révélait aussi insuffisant pour que l'utilisation d'une liste tabou soit pleinement efficace.

Pour améliorer plus souvent la solution, et ainsi pouvoir la réparer rapidement en termes d'itérations après une dégradation, la taille du voisinage a encore été augmentée. Chaque itération se résumait à considérer toutes les pièces dans une tranche de temps et à énumérer les triplets valides (par rapport à la liste tabou) formés parmi les segments sélectionnés dans le voisinage. Avec ces nouveaux voisinages, les dégradations étaient réparées au coût d'une augmentation énorme des temps de calculs.

Il fallait donc une procédure d'énumération plus rapide. Le nombre de possibilités pour chaque segment du triplet n'était pas très élevé (de l'ordre du millier pour les segments des instances de test) mais le nombre de possibilités pour un triplet l'était. Cette constatation aboutit à l'idée de mettre toutes les affectations possibles pour chacun des trois segments dans un modèle en nombres entiers, qui choisirait efficacement le triplet d'affectation grâce aux contraintes de convexité, qui limitent le nombre de branchements à faire pour trouver une solution entière. Cette idée a abouti à la formulation par génération de colonnes.

Les temps de calcul chutèrent effectivement, et cette version fut améliorée grâce à une liste tabou plus fine et des contraintes rajoutées pour imposer une réduction de la sous-couverture à un endroit ciblé. Des tests sur des instances plus compliquées mirent à mal cette nouvelle mouture de la recherche tabou qui prenait trop de temps à explorer les voisinages quand le nombre de possibilités dans chacun de segments augmentait, augmentation notamment due à l'allongement des durées des segments. Le nombre de variables du modèle en nombres entiers explosait et chaque évaluation de triplet prenait un temps supérieur à l'évaluation d'un voisinage sur le jeu d'instances précédent.

La résolution d'un voisinage de plusieurs segments à l'aide d'un modèle de génération de colonnes vint naturellement. Cette nouvelle formulation avait un gap d'intégrité assez petit, mais la résolution linéaire étouffait sur des instances d'une semaine lorsque le nombre d'activités était trop élevé. Cependant le modèle pouvait être utilisé pour résoudre des réductions topologiques du problème et la méthode était devenue une recherche à grands voisinages (RGV) qui donnait des solutions de bonne qualité, et de qualité encore meilleure avec une solution initiale obtenue par une méthode de type horizon fuyant. Chaque voisinage était alors constitué d'un sous-ensemble de pièces pour lesquelles une meilleure solution locale était cherchée. Des bornes sur la fonction objectif limitaient les temps de calculs et un système de voisinage fut greffé à la recherche tabou pour améliorer les temps de calcul tout en gardant une bonne qualité de résultats. Les résultats furent jugés satisfaisants.

Entre temps, le modèle par blocs vit le jour et produisit des résultats de bonne qualité en des temps comparables voire meilleurs en tous points à ceux de la RGV pour un certain nombre d'instances. Ces résultats poussèrent à améliorer la RGV, et pendant cette recherche, l'efficacité de l'horizon fuyant avec génération de colonnes fût découverte.

4.2.2 Recherche à grands voisinages

Tout comme les principales méthodes à trajectoire, notre méthode prend en entrée une solution initiale – celle obtenue par la stratégie de décomposition temporelle – et essaie de trouver une solution de meilleure qualité en un nombre imparti d'itérations et en un temps limité. À chaque itération, un meilleur voisin est cherché dans le grand voisinage courant à l'aide d'une heuristique basée sur le modèle (3.10)–(3.14). La recherche d'un meilleur voisin est restreinte au niveau du temps de calcul, quitte à ne pas trouver de solution. Le choix du voisinage, au niveau de son type et de sa localisation temporelle dans la solution courante, est aussi très important et dépend des itérations précédentes. Les types de voisinages et la façon dont ils sont choisis, sont décrits ultérieurement. L'heuristique est donc un procédé itératif partant d'une solution initiale réalisable et qui ne travaille qu'avec des solutions réalisables. Un voisinage est défini par rapport à une sous-couverture donnée (identifiée par une période et un type d'activité) et trouver le meilleur voisin consistera à résoudre une restriction topologique du problème (voir Figure 4.1).

À chaque itération, l'heuristique essaie de réduire le nombre de périodes de sous-couverture en résolvant une petite partie de l'instance. Si elle ne le réduit pas, elle essaie au moins de transformer une de ces périodes en une autre période de sous-couverture dans une autre partie de la solution, là où les chances de réduire le nombre de périodes en sous-couverture sont plus élevées. À partir d'un sous-ensemble de segments de travail, parmi ceux ayant une intersection temporelle avec la période de sous-couverture susceptible d'être réduite, l'heuristique tente d'améliorer localement la situation en cherchant une meilleure solution pour cet ensemble de segments uniquement. Même si la solution n'est pas meilleure d'une itération à l'autre en termes de valeur de la fonction objectif, le point important est que la situation change afin d'augmenter les chances d'améliorer la solution globale. Pour créer ce mouvement dans l'espace des solutions, plusieurs types de voisinages sont utilisés à la manière d'une recherche à voisinages variables ainsi qu'un mécanisme inspiré de la recherche tabou.

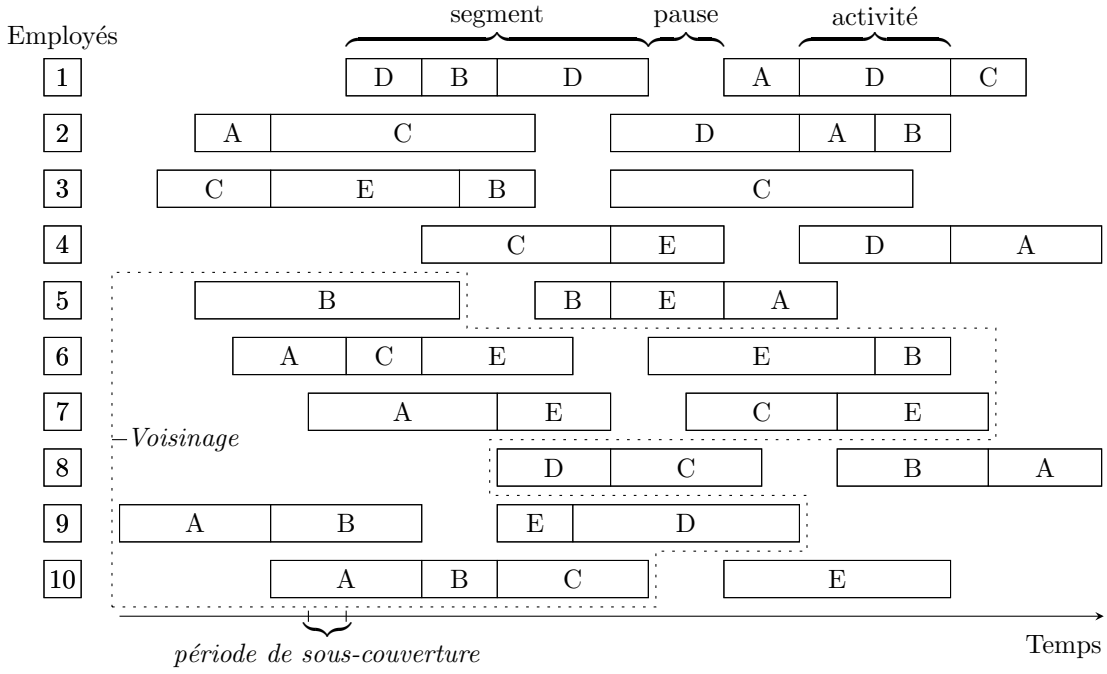

FIGURE 4.1 – Exemple de voisinage

La recherche du meilleur voisin est, elle aussi, effectuée de façon heuristique par la descente décrite dans la section précédente. Bien que cette descente ne soit pas

optimale, la résolution de la relaxation linéaire au noeud racine permet d'avoir une bonne borne inférieure sur la valeur de la fonction objectif qu'il est possible d'atteindre avec ce voisinage. Si le voisinage n'est pas supposé détériorer la solution et si la borne inférieure est trop proche de la valeur de la solution courante — autrement dit si le meilleur voisin risque d'être une solution de valeur équivalente — la recherche tabou change de voisinage en considérant l'amélioration locale de la solution comme un échec. L'ajout de contraintes sur les variables de couverture dans le problème maître peut conduire éventuellement à l'irréalisabilité du problème maître restreint. Bien que peu fréquent, ce cas est géré aussi comme un échec de la recherche locale.

Cette heuristique est inspirée de la recherche tabou dans le sens où un segment modifié durant une itération ne pourra faire partie d'un voisinage qu'au bout d'un certain temps, temps défini par le nombre de fois où le segment est considéré comme candidat pour constituer un voisinage. Il n'y aura donc pas de liste tabou globale, mais des compteurs associés à chaque segment. Couplé à une contrainte obligeant la sous-couverture choisie à une itération donnée à être réduite, ce dispositif sert à préserver la réduction de cette sous-couverture pendant un nombre d'itérations suffisamment long, dans l'espoir que la dégradation induite par la réoptimisation puisse être réparée dans un autre voisinage plus propice durant ce laps de temps. Ainsi, la solution globale aura été améliorée, ce qui n'aurait pas été le cas sans interdire de modifier de nouveau les segments directement après l'itération où leurs affectations ont été modifiées. Ce dispositif a néanmoins un inconvénient : rien ne garantit a priori que la solution pourra être réparée dans le laps de temps déterminé. Une conséquence de cet inconvénient est un cyclage local des sous-couvertures sur un nombre d'itérations supérieur au temps de protection induit par les compteurs tabou.

Une itération va donc comporter ces étapes :

1. Choix de la sous-couverture à réduire ;
2. Choix du type de voisinage ;
3. Génération du voisinage à partir d'un échantillon de segments de travail ;
4. Résolution heuristique du voisinage pour obtenir un meilleur voisin ;
5. Mise à jour de la solution et des compteurs tabou ;

Les critères d'arrêt de la recherche sont un nombre d'itérations maximal, un temps maximal de calcul, l'absence de période de sous-couverture et une non-amélioration de la meilleure valeur de la fonction objectif obtenue sur un nombre donné d'itérations.

L'algorithme 2 décrit un peu plus en détails le déroulement de la procédure de recherche à grands voisinages. Dans cet algorithme, une sous-couverture désigne le couple (période temporelle, type d'activité) définissant une période de sous-couverture. La pondération de la liste des sous-couvertures mentionnée est une pondération définie par la proximité temporelle de la dernière sous-couverture choisie et du nombre de fois où une sous-couverture a déjà été choisie lors des itérations précédentes. Cette pondération favorise l'élimination des sous-couvertures créées par les précédents voisinages. La valeur locale de la solution courante est la valeur de la fonction objectif du problème maître restreint (abrégé PMR) si les affectations des segments étaient celles de la solution courante.

La pondération mentionnée précédemment, ainsi que des critères sur les segments de travail par rapport aux qualifications, permettent de réduire le temps de calcul global, en favorisant le choix de voisinages ayant de fortes chances d'améliorer la solution courante. Nous donnons ci-dessous une brève description de notre façon de faire ces choix.

Pour choisir la sous-couverture avec laquelle travailler, nous listons toutes les sous-couvertures engendrées par la solution courante. S'il y a plusieurs sous-couvertures au même moment qu'un manque en employés, nous enlevons au hasard de la liste autant de sous-couvertures englobant cette période qu'il manque d'employés. Une fois la liste formée, nous attribuons un poids à chacun de ses éléments. Le choix d'une période de sous-couverture est ensuite effectué au hasard dans la liste avec une probabilité dépendant du poids précédemment défini.

Le poids d'une sous-couverture est défini comme une fonction linéaire de la distance temporelle avec le choix précédent u_{prec} — à la première itération, nous prenons une sous-couverture virtuelle centrée en 0 — et du nombre de fois n_u où cette sous-couverture a déjà été choisie pour définir un voisinage. La distance temporelle $d(u, v)$ entre deux sous-couvertures u et v est égale au nombre de minutes entre les centres des deux périodes de sous-couverture. La probabilité associée à une sous-couverture u est la suivante :

$$P(u) = \frac{1 + \max\limits_{w}\{poids_w\} - poids_u}{\sum\limits_{v}\left(1 + \max\limits_{w}\{poids_w\} - poids_v\right)}$$

Algorithme 2: Pseudo-code de la recherche à grands voisinages

```
initialiser la solution courante
tant que non critères de fin faire
    lister les sous-couvertures de la solution courante
    choisir une sous-couverture dans cette liste en fonction d'une pondération
    définir le type de voisinage
    choisir un ensemble de segments en respectant les compteurs tabou
    actualiser les compteurs tabou
    construire le PMR associé au voisinage
    si le voisinage le demande alors
        ajouter des contraintes additionnelles au PMR
    fin
    z^RL ← résoudre la relaxtion de PMR en générant des colonnes
    échec ← Faux
    si valeur locale de la solution courante trop proche de z^RL alors
        échec←Vrai
    sinon
        tant que solution du PMR est fractionnaire faire
            fixer à 1 la variable de segment de plus haute valeur
            z ← résoudre le PMR en générant des colonnes
            si problème irréalisable alors
                échec←Vrai ; sortir de la boucle
            fin
        fin
    fin
    si échec alors
        ne pas modifier la solution
    sinon
        modifier la solution courante
        fixer les valeurs des compteurs des segments modifiées
    fin
    actualiser le compteur d'itérations et mettre à jour le temps de calcul
fin
```

Ainsi, plus une sous-couverture est proche de la sous-couverture courante et moins de fois elle a été choisie, plus elle aura de chance d'être choisie. Ces deux critères viennent du fait que lorsqu'on déplace la situation problématique, il faut s'en occuper le plus rapidement possible.

L'ensemble de segments que nous choisissons pour construire un voisinage, appelé échantillon, a donc une constitution particulière pour faciliter la résolution en augmentant le potentiel de bonnes solutions du voisinage.

- la moitié des segments appartient à des employés qualifiés pour le type d'activité en sous-couverture
- la moitié des segments appartient à des employés contribuant à un type d'activité en sur-couverture durant la même période que celle en sous-couverture

Cet ensemble est le sous-ensemble des pièces de travail composant les quarts de travail de ces deux types d'employés. Pour ne pas avoir des voisinages trop verticaux, si un segment est mis dans l'échantillon, alors ses segments voisins dans le même quart y sont mis aussi, sauf s'ils sont tabou.

Concernant les voisinages, quatre types ont été définis. Le premier correspond à un nombre de segments relativement limité. Aucune contrainte n'est ajoutée au problème maître et la résolution d'un de ces voisinages doit être rapide. Le deuxième type a une taille identique à celle du premier type mais des contraintes sur les variables de couverture sont ajoutées. Ces contraintes sont simplement des contraintes de bornes sur les variables de sous-couverture correspondant à la sous-couverture choisie, obligeant cette dernière à être réduite d'une unité par rapport à sa valeur dans la solution courante. Le troisième type de voisinage est de taille deux fois supérieure aux tailles des deux premiers types et impose également, aux variables de sous-couverture correspondant à la période de sous-couverture et aux variables pour le même type d'activité pour des périodes adjacentes, une borne sur leurs valeurs. Finalement, le dernier voisinage est de taille deux fois supérieure au deuxième type et n'ajoute aucune contrainte au problème maître. En revanche, ce voisinage ne tient pas compte du caractère tabou des segments.

Ces voisinages ont respectivement pour rôle :

1. Assurer une descente rapide vers un optimum local,
2. Créer un petit mouvement pour déplacer la situation problématique,
3. Créer un grand mouvement pour forcer le déplacement,

4. Réparer la solution.

A chaque itération, l'heuristique détermine le type de voisinage à utiliser parmi $V = \{1, 2, 3, 4\}$. Ce type est déterminé en fonction d'un compteur d'itérations depuis la dernière amélioration de la fonction objectif et d'un critère c. A une itération i donnée, si $v_i \in V$ dénote le voisinage courant et n_i le nombre d'itérations depuis la dernière amélioration, alors :

$$v_{i+1} \leftarrow \begin{cases} v_i & \text{si } n_i < v_i c \\ v_i + 1 & \text{si } n_i \geq v_i c \\ v_i & \text{si } v_i = \max v \in V \end{cases} .$$

A chaque itération, le compteur d'itérations depuis la dernière amélioration est augmenté de 1 si la valeur de la fonction objectif est plus grande ou égale à la meilleure valeur de la fonction objectif à date, sinon il est remis à zéro.

4.2.3 Comparaison avec la méthode d'horizon fuyant

Cette comparaison a été effectuée avec le jeu d'instances le plus difficile que nous avions construit à l'époque. Toutes ces instances sont perturbées par l'augmentation artificielle des durées minimales de contribution aux types d'activités, de façon importante. Les instances ont des horizons temporels d'un jour, de deux jours ou d'une semaine. Les tests ont été lancé dix fois pour chaque instances avec des graines différentes pour le générateur pseudo-aléatoire. Les résultats de la méthode à voisinages sont résumés dans le tableau 4.1 puis comparés aux résultats de la meilleure heuristique de la section 3.3 sur le même jeu d'instances, grâce au tableau 4.2.

Instance	**Borne**	**Initial**	**5 min**	**Valeur de la fonction objectif**					**Temps**	**Iter.**
				min	méd.	max	σ	gap		
instance1	61706.29	66210	63445	62880	62895	62925	18.71	1.90	3617	289
instance2	170093.23	175215	174573	170355	173247	175155	1954.20	0.15	3158	366
instance3	671750.11	706965	705697	700320	703132	706590	2083.67	4.25	3603	204
instance4	44561.49	44640	44638	44610	44635	44640	10.12	0.11	3616	172
instance5	143314.26	158670	150343	143775	145299	147135	1204.06	0.32	2826	602
instance6	704829.97	753420	748108	738495	740628	746535	2266.96	4.78	3602	488

TABLEAU 4.1 – Résultats des tests sur des instances difficiles

Le tableau 4.1 contient les résultats sur six instances. Pour chaque instance identi-

fiée à la première colonne sont rapportés dans l'ordre des colonnes : la borne inférieure obtenue par relaxation linéaire, le coût de la solution initiale, la valeur moyenne de l'objectif après cinq minutes de temps de calcul, puis cinq statistiques sur la valeur finale de la fonction objectif pour les dix exécutions différentes par instance — le minimum, la médiane, le maximum, l'écart-type et le gap minimum en pourcentage — puis le temps de calcul en secondes, et enfin le nombre moyen d'itérations de la méthode. Les améliorations par rapport à la solution initiale sont considérables, la limite des temps de calcul à une heure est suffisante et l'écart-type assez petit : le tableau 4.1 arbore des résultats corrects. La solution initiale est obtenue par une méthode d'horizon fuyant ressemblant furieusement à la méthode avec laquelle la recherche à grands voisinages est comparée dans le tableau 4.2. Néanmoins, le paramétrage utilisé alors ne permettait pas d'exploiter son plein potentiel.

Instance	**Coût RGV**	**Temps RGV**	**Coût HFGC**	**Temps HFGC**
instance1	62925	3617	64680	161
instance2	175155	3158	173430	363
instance3	706590	3603	690195	1089
instance4	44640	3616	44655	344
instance5	147135	2826	148770	272
instance6	746535	3602	730425	1622

Tableau 4.2 – Comparaison avec l'horizon fuyant seul

Les colonnes 2 et 3 rapportent les pires résultats de la recherche à grands voisinages et les colonnes 4 et 5 rapportent les résultats pour la méthode d'horizon fuyant. La recherche à grands voisinages obtient de bien meilleurs résultats sur les instances d'un jour (instance1, instance4), des résultats équivalents pour les instances de deux jours (instance2, instance5), et de moins bons résultats pour les instances d'une semaine (instance3, instance6). Ce dernier point nous a poussé à mettre de côté la recherche à grands voisinages, avec le fait que les temps de calculs sont toujours meilleurs pour l'horizon fuyant et que la recherche à grands voisinages a de trop nombreux paramètres.

Finalement, la méthode de recherche à grands voisinages ne s'est pas révélée assez performante pour résoudre les instances les plus difficiles du premier article.

Chapitre 5

DISCUSSION GÉNÉRALE ET CONCLUSION

Les problèmes d'horaires de personnel sont une classe de problèmes combinatoires diverse et variée dépendant beaucoup du contexte dans lequel ils apparaissent. Cette thèse ne traite que trois d'entre eux qui sont reliés au contexte multi-activités dans des quarts pré-déterminés par des modules antérieurs. Ces problèmes, imbriqués les uns dans les autres, proviennent de la démarche utilisée pour traiter le dernier. Pour chacun de ces problèmes, au moins une méthode de résolution a été proposée et validée par des résultats numériques obtenus par l'application de la méthode à des instances générées aléatoirement. Les résultats de ces tests numériques ont aussi permis de tirer des conclusions quant à certains aspects généraux liés aux problèmes, décrits dans la section 5.1.

5.1 Bilan des contributions

Une formulation globale nouvelle a été proposée pour chaque problème abordé dans cette thèse, à savoir le problème d'affectation d'activités multiples, le problème d'affectation de tâches et d'activités multiples et le problème d'affectation de tâches en équipe et d'affectation d'activités multiples.

Pour le premier de ces problèmes, deux modèles en nombres entiers ainsi que diverses heuristiques basées sur ces modèles ont été présentées et comparées sur des jeux d'instances pour déterminer quelle était la plus efficace sur les instances qui apparaissent en entreprise. La plus efficace s'est révélée être une heuristique de type horizon fuyant basée sur un modèle en nombre entiers, résolu à l'aide d'un algorithme de génération de colonnes. Le sous-problème de génération de colonnes étant simple et de petite taille en général, la méthode se prête bien, à première vue, à des générali-

sations par rapport aux contraintes sur les segments, voire les quarts de travail : ajout de contraintes sur les séquences d'activités, sur les débuts et fins de segments, etc. L'utilisation de plus courts chemins avec ressources ou de grammaires dans les sous-problèmes permet l'intégration de règles de conventions collectives plus complexes sans avoir besoin de changer du tout au tout la méthode de résolution.

Le deuxième problème résulte de l'introduction des tâches individuelles au sein du précédent problème. Bien que ces dernières soient effectuées par des employés en solo, le concept de tâches individuelles est relativement large pour saisir des travaux de nature sporadique. La méthode proposée pour résoudre ce problème est une décomposition en deux étapes. Le placement des tâches individuelles par un modèle approximatif pouvant conduire à de mauvaises décisions, la réaffectation de tâches individuelles s'est révélée efficace pour améliorer la qualité des solutions, avec comme contrepartie des temps de calculs légèrement supérieurs. De plus, l'heuristique s'est montrée fiable et rapide durant les tests numériques réalisés, fournissant des solutions de bonne qualité, sans être particulièrement sensible aux diverses caractéristiques des tâches se retrouvant dans les instances de tests. Là aussi, une certaine extensibilité est garantie par le principe de réaffectation. Les contraintes liantes au niveau des tâches sont à déterminer dans le modèle approximatif, si possible d'une façon similaire aux relations de préséance, les autres contraintes ou les conséquences des contraintes précédentes peuvent se traiter au niveau des sous-problèmes tout en conservant la méthode de résolution.

L'expérience acquise avec les tâches individuelles a été utile pour proposer une méthode de résolution pour l'ultime problème. La décomposition du problème précédent en deux étapes a conduit à l'utilisation de deux modules similaires au sein d'une heuristique d'amélioration. En effet, la taille du modèle approximatif devenant trop grande pour les instances visées, l'intégration de l'approche précédente dans une heuristique à voisinages variables a permis la résolution du problème. Durant les tests numériques réalisés, la méthode s'est révélée capable de placer des tâches en équipe, à partir d'une affectation incomplète de tâches considérées comme individuelles, dans des temps de calcul raisonnables. Ces tests numériques ont aussi souligné que l'introduction des tâches en équipe par le fait de scinder des tâches individuelles permet d'obtenir un gain en qualité de solutions. Ceci peut encourager les planificateurs à scinder les tâches individuelles dans l'optique d'améliorer la qualité de service de leurs horaires. Finalement, l'heuristique d'amélioration a aussi confirmé que la réaffectation

de tâches individuelles conduisait à de solutions de très bonne qualité par rapport au temps de calcul dépensé. En effet, l'heuristique d'amélioration a fourni des solutions certes d'une meilleure qualité, mais en des temps de calculs démesurément supérieurs.

5.2 Limitations et améliorations futures

Malgré ces succès, tous les tests réalisés ont été effectués sur des instances générées de façon aléatoire. Ainsi, aucun test en conditions réelles n'a pu être effectué et les résultats obtenus ne sont que supposés généraux par nature, sans aucune garantie de l'être vraiment.

Le problème proposé par Kronos Inc. ne contenait pas l'ensemble des contraintes auxquelles leur logiciel doit faire face. Certaines contraintes globales sont susceptibles de briser la structure du problème, en particulier, les contraintes liant le contenu des segments sur plusieurs jours. Une partie de ces contraintes peut se traiter dans le modèle approximatif mais les heuristiques de type horizon fuyant nécessitent des adaptations particulières pour ce type de contraintes et pour les irréalisabilités qui apparaîtraient au niveau de la dernière tranche de temps, qui sont généralement dures à réparer.

Les différentes méthodes de résolution proposées sont efficaces sur les instances de test mais des améliorations peuvent être apportées aux différentes méthodes. Pour améliorer la qualité des résultats, autoriser des mouvements valides au niveau des débuts et des fins de segments peut apporter suffisamment de flexibilité pour diminuer le nombre de sous-couvertures. Cette première extension nécessite l'adaptation du problème maître et des sous-problèmes pour tenir compte de cette flexibilité, mais la nature de plus court chemin des sous-problèmes peut être conservée par l'introduction de noeuds et d'arcs spécifiques aux différents temps de départ.

Une deuxième extension possible est l'introduction d'une manière générique de gérer des contraintes globales sur les activités dans l'horizon fuyant. Actuellement, aucune contrainte de ce type ne peut être intégrée facilement dans l'heuristique, car les activités sont affectées de façon aveugle par rapport aux tranches postérieures. Une partie de la gestion de ces contraintes pourrait s'effectuer dans le modèle d'affectation de tâches, mais la prise en compte de ces contraintes durant l'horizon fuyant représente un bon défi.

Références

AHUJA, R., MAGNANTI, T. et ORLIN, J. (1993). *Network Flows : Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, NJ.

ALVAREZ-VALDÈS, R. et TAMARIT, J. M. (1993). The project scheduling polyhedron : dimension, facets and lifting theorems. *European Journal of Operational Research*, 67, 204–220.

AYKIN, T. (1996). Optimal shift scheduling with multiple break windows. *Management Science*, 42, 591–602.

BAPTISTE, P., GIARD, V., HAÏT, A. et SOUMIS, F. (2005). *Gestion de production et ressources humaines : méthodes de planification dans les systèmes productifs*. Presses internationales Polytechnique.

BARD, J. F., BINICI, C. et DE SILVA, A. H. (2003). Staff scheduling at the united states postal service. *Computers and Operations Research*, 30, 745–771.

BARD, J. F. et WAN, L. (2006). The task assignment problem for unrestricted movement between workstation groups. *Journal of Scheduling*, 9, 315–341.

BARD, J. F. et WAN, L. (2008). Workforce design with movement restrictions between workstation groups. *Manufacturing & Service Operations Management*, 10, 24–42.

BARNHART, C., JOHNSON, E. L., NEMHAUSER, G. L. et SAVELSBERGH, M. W. P. (1998). Branch-and-price : Column generation for solving huge integer programs. *Operations Research*, 46, 316–329.

BECHTOLD, S. E. et JACOBS, L. W. (1990). Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, 36, 1339–1351.

BECHTOLD, S. E. et JACOBS, L. W. (1996). The equivalence of general set-covering and implicit integer programming formulations for shift scheduling. *Naval Research Logistics*, 43, 233–249.

BORNEMANN, D. R. (1970). A crew planning scheduling system. *AGIFORS Symposium Proceedings*, 10, 1–19.

BOUCHARD, M. (2004). *Optimisation de pauses dans le problème de fabrication d'horaires avec quarts de travail.* Mémoire de maîtrise, École Polytechnique de Montréal, Montréal, Canada.

BOUCHARD, M. (2008). *Coloration de graphes et attribution d'activités dans des quarts de travail.* Thèse de doctorat, École Polythechnique de Montréal, Montréal, Canada.

BRUCKER, P., DREXL, A., MOHRING, R., NEUMANN, K. et PESCH, E. (1999). Resource-constrained project scheduling : notation, classification, models, and methods. *European Journal of Operational Research*, 112, 3–41.

BRUSCO, M. J. et JACOBS, L. W. (2000). Optimal models for meal-break and start-time flexibility in continuous tour scheduling. *Management Science*, 46, 1630–1641.

BURNS, R. et KOOP, G. (1987). A modular approach to optimal multiple-shift manpower scheduling. *Operations Research*, 35, 100–110.

ÇEZIK, T. et GÜNLÜNK, . (2004). Reformulating linear programs with transportation constraints - with applications to workforce scheduling. *Naval Research Logistics*, 51, 275–296.

CÔTÉ, M. C., GENDRON, B., QUIMPER, C. et ROUSSEAU, L. M. (2009). Formal languages for integer programming modeling of shift scheduling problem. *Constraints*, Forthcoming, doi :10.1007/s10601–009–9083–2.

CÔTÉ, M. C., GENDRON, B. et ROUSSEAU, L. M. (2007). Modeling the regular constraint with integer programming. *CPAIOR*. 29–43.

DANTZIG, G. B. (1954). A comment on edie's "traffic delays at toll booths". *Journal of the Operations Research Society of America*, 2, 339–341.

DANTZIG, G. B. et WOLFE, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8, 101–111.

DAVIS, E. W. et HEIDORN, G. E. (1971). An algorithm for optimal project scheduling under multiple resource constraints. *Management Science*, 17, B803–B816.

DEMEULEMEESTER, E. L. et HERROELEN, W. S. (2002). *Project scheduling : a research handbook*, vol. 42. Kluwer Academic Publishers.

DESAULNIERS, G., DESROSIERS, J. et SOLOMON, M. M. (2005). *Column generation.* Springer, New York, NY.

DESROSIERS, J., SOUMIS, F. et DESROCHERS, J. (1984). Routing with time windows by column generation. *Network*, 14, 545–565.

DOLAN, E. D. et MORÉ, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, Series A91, 201–213.

EASTON, F. F. et ROSSIN, D. F. (1991). Sufficient working subsets for the tour scheduling problem. *Management Science*, 37, 1441–1451.

EDIE, L. C. (1954). Traffic delays at toll booths. *Journal of the Operations Research Society of America*, 2, 107–138.

ELHALLAOUI, I., VILLENEUVE, D., SOUMIS, F. et DESAULNIERS, G. (2005). Dynamic aggregation of set partitioning constraints in column generation. *Operations Research*, 53, 632–645.

ERNST, A. T., JIANG, H., KRISHNAMOORTHY, M., OWENS, B. et SIER, D. (2004a). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127, 21–144.

ERNST, A. T., JIANG, H., KRISHNAMOORTHY, M. et SIER, D. (2004b). Staff scheduling and rostering : a review of applications, methods and models. *European Journal of Operational Research*, 153, 3–27.

FISHER, M. L. (1973). Optimal solution of scheduling problems using lagrange multipliers : Part i. *Operations Research*, 21, 1114–1127.

FORD, L. R. et FULKERSON, D. R. (1962). *Flows in networks*. Princeton University Press, NJ.

GABALLA, A. et PEARCE, W. (1979). Telephone sales manpower planning at qantas. *Interfaces*, 9, 1–9.

GALE, D. (1957). A theorem on flows in networks. *Pacific Journal of Mathematics*. vol. 7, 1073–1082.

GORENSTEIN, S. (1972). An algorithm for project (job) sequencing with resource constraints. *Operations Research*, 20, 835–850.

HAASE, K., DESAULNIERS, G. et DESROSIERS, J. (2001). Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35, 286–303.

HANSEN, P. et MLADENOVIĆ, N. (2001). Variable neighborhood search : Principles and applications. *European Journal of Operational Research*, 130, 449–467.

HENDERSON, W. B. et BERRY, W. L. (1976). Heuristic methods for telephone operator shift scheduling : An experimental analysis. *Management Science*, 22, 1372–1380.

JACOBS, L. W. et BRUSCO, M. J. (1996). Overlapping start-time bands in implicit tour scheduling. *Management Science*, 42, 1247–1259.

JARRAH, A., BARD, J. et DESILVA, A. (1994). Solving large-scale tour scheduling problems. *Management Science*, 40, 1124–1144.

KARP, R. M. (1972). Reducibility among combinatorial problems. R. E. Miller et E. J. W. Thatcher, éditeurs, *Complexity of Computer Computations : Proceedings of a Symposium on the Complexity of Computer Computations*. Plenum Press, New York, NY, The IBM Research Symposia Series, 85–103.

KEITH, E. G. (1979). Operator scheduling. *IIE Transactions*, 11, 37–41.

KELLERER, H., PFERSCHY, U. et PISINGER, D. (2004). *Knapsack Problems*. Springer, Berlin, Germany.

LAVOIE, S., MINOUX, M. et ODIER, E. (1988). A new approach for crew pairing problems by column generation with an application to air transportation. *European Journal of Operational Research*, 35, 45–58.

LAZIĆ, J., HANAFI, S., MLADENOVIĆ, N. et & UROŠEVIĆ, D. (2009). Variable neighborhood decomposition search for 0-1 mixed integer programs. Rapport technique, *Les Cahiers du GERAD G-2009-70* HEC.

LEQUY, Q., BOUCHARD, M., DESAULNIERS, G. et SOUMIS, F. (2009). Assigning multiple activities to work shifts. Rapport technique, *Les Cahiers du GERAD G-2009-86* HEC.

LEQUY, Q., DESAULNIERS, G. et SOLOMON, M. (2010). A two-stage heursitic for assigning multiple activities and tasks to fixed work shifts. Rapport technique, *Les Cahiers du GERAD G-2009-86* HEC.

LÜBBECKE, M. et DESROSIERS, J. (2005). Selected topics in column generation. *Operations Research*, 53, 1007–1023.

MLADENOVIĆ, N. et HANSEN, P. (1997). Variable neighborhood search. *Computers and Operations Research*, 24, 1097–1100.

MOONDRA, S. (1976). An l. p. model for work force scheduling for banks. *Journal of Bank Research*, 7, 299–301.

NEMHAUSER, G. et WOLSEY, L. (1988). *Integer and combinatorial optimization*. John Wiley & Sons, Inc.

OMARI, Z. (2002). *Attribution des activités aux employés travaillant sur des quarts*. Mémoire de maîtrise, École Polytechnique de Montréal, Montréal, Canada.

ÖZDAMAR, L. et ULUSOY, G. (1995). A survey on the resource-constrained project scheduling problem. *IIE Transactions*, 27, 574–586.

PATTERSON, J. H. et HUBER, W. D. (1974). A horizon-varying, zero-one approach to project scheduling. *Management Science*, 20, 990–998.

PISINGER, D. (2005). Where are the hard knapsack problems. *Computers & Operations Research*, 32, 271–2284.

REKIK, M., CORDEAU, J. F. et SOUMIS, F. (2004). Using benders decomposition to implicitly model tour scheduling. *Annals of Operations Research*, 128, 111–133.

REKIK, M., CORDEAU, J.-F. et SOUMIS, F. (2009). Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling (Online)*, 13, 49–75.

RUBIN, J. (1973). A technique for the solution of massive set covering problems with applications to airline crew scheduling. *Transportation Science*, 7, 34–48.

SEGAL, M. (1974). The operator-scheduling problem : a network-flow approach. *Operations Research*, 22, 808–823.

TALBOT, F. B. et PATTERSON, J. H. (1978). An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. *Management Science*, 24, 1163–1174.

THOMPSON, G. M. (1995). Improved implicit optimal modeling of the labor shift scheduling problem. *Management Science*, 41, 595–607.

TOPALOGLU, G. et OZKARAHAN, I. (2000). A goal programming approach for large-scale tour scheduling problems with flexible break assignments. *Proceedings of the 31st Annual Meeting of the Decision Science Institute, Orlando, Florida.*

VATRI, E. (2001). *Intégration de la génération de quarts de travail et de l'attribution d'activités*. Mémoire de maîtrise, École Polytechnique de Montréal, Montréal, Canada.

WARNER, M. et PRAWDA, J. (1972). A mathematical programming model for scheduling nursing personnel in a hospital. *Management Science*, 19, 411–422.

ARTICLE 1 : ASSIGNING MULTIPLE ACTIVITIES TO WORK SHIFTS

Article soumis à *Journal of scheduling* (2009) et écrit par :

QUENTIN LEQUY

École Polytechnique de Montréal et GERAD

MATHIEU BOUCHARD

Université Laval

GUY DESAULNIERS

École Polytechnique de Montréal et GERAD

FRANÇOIS SOUMIS

École Polytechnique de Montréal et GERAD

BEYIME TACHEFINE

Kronos Canadian Systems Inc.

Abstract

In some companies such as large retail stores, the employees perform different activities (e.g. cashier or clerk in a specific department) to respond to a customer demand for each activity that varies over the planning horizon and must be fulfilled as soon as possible. For a given time period this demand translates into an ideal number of employees required for the corresponding activity. During work shift, an employee can be assigned to several activities that are interruptible at any time and subject to operational constraints (required skills, minimum and maximum assignment durations).Given work shifts already assigned to employees withvarious skills, the multi-activity assignment problem (MAAP) consists of assigning activities to the work shifts such that some constraints on skills and duration are fullfilled and such that the activity demands are satisfied as best as possible over the planning horizon. In this paper, we propose three integer programming models for the MAAP and develop various heuristics based on mathematical programming techniques. Computational results obtained on randomly generated MAAP instances show that a heuristic column generation method embedded into a rolling horizon procedure provides the best results in general.

Keywords : Activity assignment, Work shifts, Multi-commodity network flow model, Column generation, Rolling horizon.

1 Introduction

For most medium- to large-sized companies, building a low-cost feasible working schedule for their employees is not an easy task. Depending on the business type, just finding a feasible schedule might even be a challenge. Nevertheless, modern modeling and optimization techniques allow one to propose efficient solution methods for several personnel scheduling problems. In this paper, we focus on a planning scheduling problem that arises in companies where the employees work shifts. Moreover those shifts should respond to a varying customer demand that must be fulfilled as soon as possible. The demand for each activity is not exactly known when shifts are built and is also perishable. That means companies might lose clients immediatly if they're not satisfied of the service. Examples of such companies are supermarkets or leisure

resorts. We assume that the work performed by an employee is interruptible at any time because an employee can be replaced by another one almost instantaneously or its absence does not diminish substantially the quality of the service offered by the company at that time. For instance, a cashier in a supermarket can rapidly interrupt her/his work to provide assistance at the customer service counter if another cashier is available to replace her/him. A type of work such as operating a cash register or providing customer service is called an *activity*.

In the simplest *mono-activity* case, the scheduling problem concerns a single activity. In the more complex *multi-activity* case, the employees can be assigned to different activities during their work shifts. With each activity is associated a demand curve that indicates the ideal number of employees required for this activity at any time of the planning horizon. Employees must, therefore, be scheduled to cover these demands as much as possible. The schedule of an employee is composed of working days and days off. For each working day, the employee is assigned to a work shift that is defined by a start and an end time. It may also contain one or several specified break periods. Finally, each work shift must be filled with activities to accomplish. Labor and collective agreement rules restrict the feasibility of a schedule. Furthermore, the employees possess skills and cannot be assigned to an activity for which they are not qualified. When building the employee schedules, the primary objective of the company is to reduce personnel costs while maintaining high customer satisfaction. A secondary objective might be to offer whenever possible pleasant schedules to the employees. Consequently, when there are many activities and the demand curves vary substantially, diversified work shifts and activity assignments must be considered, yielding a highly combinatorial optimization problem.

Globally, this scheduling problem is too complex to be tackled at once. Hence, it is usually decomposed into the following four steps that are solved sequentially :

- *Days off scheduling* determines the days off of each employee over a given horizon (typically, one month) ;
- *Shift scheduling* constructs anonymous work shifts to satisfy an aggregated demand for each day of the horizon ;
- *Shift assignment* assigns each work shift to a specific employee available to work on the corresponding day ; and
- *Activity assignment* assigns the activities to accomplish within each work shift

taking into account the skills of the employees assigned to those shifts. This last step is obviously not necessary in a mono-activity context.

Typically, the first three steps are completed a few weeks prior to the day of operations. This allows the employees to know their free time long in advance. On the other hand, because the demand curve for each activity is not very accurate too far ahead of time, the last step is usually performed only a few days prior to the day of operations.

In most companies, these four steps are interdependent. Consequently, attempts have been made to integrate some of them. In particular, the integration of the first two steps gives rise to the *tour scheduling* problem. All these problems, except the activity assignment problem, have been widely studied in the literature as shown by the numerous references given in the surveys of Ernst *et al.* (2004a,b). The *task assignment* problem, which is akin to the activity assignment problem, has also been the subject of several papers cited in those surveys. Nevertheless, the task assignment literature is not relevant for the activity assignment problem because these two problems have one fundamental difference : when assigned to an employee, a task is uninterruptible, while an activity can be interrupted. Furthermore, in many contexts, a task can start and end in different locations, while an activity is always performed at the same location.

To the best of our knowledge, the activity assignment problem has not been addressed in the literature other than in three master's theses (written in French and part of the same research project supervised by two authors of this paper) and one conference paper. Omari (2002) studied an activity assignment problem for air traffic controllers that includes several complicating issues such as the scheduling of training periods which requires synchronizing the trainees with their instructors. In a similar but simplified context, Vatri (2001) considered the integration of the shift scheduling, shift assignment, and activity assignment problems. To reduce the complexity of this integrated problem, no breaks were scheduled in the shifts. The additional treatment of breaks was addressed by Bouchard (2004). These three theses proposed a similar solution framework that applies a rolling time horizon to decompose the problem into subproblems (each restricted to a time slice of the horizon), and solves sequentially each of these subproblems using a column generation heuristic. Such a framework is also proposed in our work. Finally, Côté *et al.* (2007) proposed a mixed

integer programming model for a multi-activity shift scheduling problem for anonymous employees. In this model, certain schedule feasibility rules (concerning breaks, rest periods, and work stretches) are modeled using techniques from constraint programming to yield a more compact and easier-to-solve model.

In this paper, we concentrate on the multi-activity assignment problem (MAAP), where the work shifts (including breaks) are fixed and already assigned to the employees. The problem version considered is a particular case of the problem treated by Omari (2002) : it does not contain the issues specific to air traffic controllers. Our main goal is to develop a heuristic solution method that allows one to produce very good quality solutions in acceptable computational times (say, less than 2 hours) for practical-size instances. To do so, we propose first several models and mathematical-programming-based heuristics for the MAAP, and compare them.

Our contributions are as follows. First, we introduce the MAAP to the operations research community at large. Second, we propose three formulations for this problem : an integer multi-commodity network flow model with side constraints, an integer programming model without special structure, and a column generation model. Third, we develop heuristics based on mathematical programming tools for solving two of these models and propose to embed them into a rolling horizon procedure for tackling large-sized instances. Finally, computational results are reported to compare these different models and heuristics. They show that good-quality solutions for practical-size instances can be obtained in reasonable computational times with one of the proposed methods, namely the column generation heuristic.

The paper is organized as follows. A detailed definition of the MAAP is given in Section 2. The models are presented in Section 3, while the solution methods are described in Section 4. Computational results are reported in Section 5, and a conclusion is drawn in Section 6.

2 The multi-activity assignment problem

The MAAP is a planning problem defined over a time horizon that lasts typically between one day and one week, the activity demand curves often being not sufficiently accurate beyond one week. This horizon is discretized into periods of equal duration

(for instance, 15 minutes). The demand curves are assumed to be constant in every time period. Before stating the MAAP, let us define some terminology and introduce some concepts.

Piece of work : Part of a shift that starts either at the beginning of the shift or at the end of a break and ends either at the next start of a break or at the end of the shift. A work shift can, thus, be seen as an alternating sequence of pieces of work and breaks that starts and ends with a piece of work. In a piece of work, the employee must be assigned to a sequence of activities (possibly a single one). In such a sequence, an activity can appear more than once and no idle time is allowed.

Activity duration : The elapsed time during which an employee is assigned consecutively to an activity in a piece of work. The duration of an activity is restricted by a minimal and a maximal duration, which may vary from one activity to another. Minimal and maximal durations are usually imposed by safety or collective agreement rules, but minimal durations might also be considered for avoiding too frequent activity changes that may reduce productivity.

Qualifications : An employee can be assigned to an activity only if he possesses the skills to perform it. In this case, we say that the employee is qualified for this activity.

Over- and under-covering : An over-covering occurs for each employee assigned to an activity in a given time period that exceeds the demand for this activity in this period. Similarly, an under-covering occurs for each employee failing to reach the demand for an activity in a time period. A cost is incurred for each over- and under-covering. The under-covering cost, which can be activity dependent, should be proportional to the loss in service quality yielded by an under-covering. The over-covering cost is used to favor certain activities over others when over-covering occurs.

Transition : A transition occurs each time that an employee changes activity in a piece of work. A cost is associated with each transition. This cost should represent the loss in productivity due to a transition or reflect the inconvenience for an employee of changing activity too frequently.

With this terminology, the MAAP can be stated as follows.

Given

- a time horizon divided into time periods ;
- the work shifts that each employee must work over the horizon, including the breaks in those shifts ;
- the list of activities for which each employee is qualified ;
- the demand curve for each activity ; and
- the minimal and maximal durations for each activity ;

fill each piece of work of each shift with a sequence of activities such that

- an activity is assigned to a piece of work if and only if the employee working this piece is qualified for this activity ;
- the minimal and maximal activity durations are satisfied ;
- the total under-covering cost is minimized ;
- the total over-covering cost is minimized ; and
- the total transition cost is minimized.

The MAAP is thus a multi-criteria optimization problem. In general, there is a hierarchy between these criteria : minimize first the total under-covering cost, second the total over-covering cost, and third the total transition cost. Using weights of different magnitudes for these three criteria, the MAAP can then be modeled using a single objective function.

In practice, a typical MAAP instance with daytime and nighttime shifts considers a 1-week horizon divided into time periods of 15 minutes. It can involve 50 shifts or more per day, each with at least two pieces of work, and up to 15 different activities. On average, each employee is qualified to perform approximately 50 to 75% of the activities. Such a typical instance is, therefore, highly combinatorial.

3 Mathematical model

In this section, we propose three integer programming models for the MAAP. Because every shift is assigned to an employee that possesses its own qualifications, these models must be explicit with regard to each shift. On the other hand, each shift can be seen as a sequence of independent pieces of work and, therefore, the definition of these models can be based on pieces of work assigned to employees instead of work

shifts.

The models rely on the following common notation. Let W, A, and T be the sets of pieces of work, of activities, and of time periods, respectively. For each activity $a \in A$ and period $t \in T$, denote by $d_{a,t}$ the demand for activity a in period t and by $\bar{c}_a$ (resp., $\underline{c}_a$) the nonnegative cost for one over-covering (resp., under-covering) of activity a. Denote by c_τ the cost of a transition. For each piece of work $w \in W$, let T_w be the set of time periods it spans and A_w the set of activities for which the employee assigned to piece w is qualified. All the proposed models involve the following two types of variables. The nonnegative over-covering variables $O_{a,t}$, $a \in A$, $t \in T$, provide the number of over-coverings for each activity in each period. Similarly, the nonnegative under-covering variables $U_{a,t}$, $a \in A$, $t \in T$, give the number of under-coverings for each activity in each period.

When an employee is assigned to an activity for a certain number of consecutive periods that respects the corresponding minimum and maximum activity durations, we say that it performs an *activity block*. Thus, each piece of work can be seen as a sequence of nonoverlapping activity blocks that covers the whole piece. The first two models are based on the enumeration of all feasible activity blocks for each piece of work $w \in W$, that is, for each activity $a \in A_w$ and each period $t \in T_w$, all feasible blocks for activity a that start in period t and end in a period $t' \in T_w$ such that $\delta(t'-t+1) \in [\delta_a^{min}, \delta_a^{max}]$, where δ is the length of a time period and δ_a^{min} (resp. δ_a^{max}) is the minimum (resp. maximum) duration for activity a.

3.1 A multi-commodity network flow model

The first formulation is defined over acyclic directed networks, one for each piece of work $w \in W$ that is denoted $\mathcal{N}^w$. In these networks, every feasible activity sequence for the corresponding piece of work is represented by a path. Such a network is illustrated in Figure 6.1. For this example, we assume that there are three activities (A, B, and C), the length δ of a time period is 15 minutes, and the minimum and maximum activity durations are $\delta_A^{min} = \delta_C^{min} = 15$, $\delta_B^{min} = 30$, $\delta_A^{max} = 45$, $\delta_B^{max} = \delta_C^{max} = 30$. The work piece consists of three periods during [15,60].

In network $\mathcal{N}^w$, there are $2 + 2|T_w||A_w|$ vertices. There is a unique source vertex α_w and a unique sink vertex β_w that represent the start and the end of piece w,

respectively. Also, there is a pair of vertices $B^w_{a,t}$ and $E^w_{a,t}$ for each activity $a \in A_w$ and each time period $t \in T_w$: $B^w_{a,t}$ indicates the beginning of an assignment to activity a at the beginning of period t, while $E^w_{a,t}$ represents the end of such an assignment at the end of period t.

Table 6.1 specifies the arcs contained in this network, where $q_w = |T_w|$ is the number of periods in T_w and $t_1, t_2, \ldots, t_{q_w}$ denote these periods in chronological order. The first and second lines define the arcs representing the start and the end of the piece of work w, respectively. The third line enumerates the arcs representing all the activity blocks that can be performed in the piece of work. Finally, the last line describes the arcs representing a transition between two different activities. This network structure ensures that any path from α_w to β_w corresponds to a feasible activity sequence s for piece w and vice versa.

Head vertex	**Tail vertex**	**Condition**
α_w	$B^w_{a,t_1}, \quad \forall a \in A_w$	
$E^w_{a,t_{q_w}}, \quad \forall a \in A_w$	β_w	
$B^w_{a,t_i}, \quad \forall a \in A_w, i = 1, \ldots, q_w$	$E^w_{a,t_{i+j}}, \quad \forall j = 0, \ldots, q_w - i$	$\delta^{min}_a \leq (j+1)\delta \leq \delta^{max}_a$
$E^w_{a,t}, \quad \forall a \in A_w, t \in T_w$	$B^w_{a',t+1}, \quad \forall a' \in A_w \setminus \{a\}$	

TABLEAU 6.1 – Arcs in a network for a piece of work $w \in W$

The following notation is required for the proposed model.

$\mathcal{V}^w$: set of all the vertices in network $\mathcal{N}^w$;

$\mathcal{V}^{w,t}_B$: subset of the vertices in $\mathcal{N}^w$ representing the beginning of an activity at the beginning of period $t \in T_w$;

$\mathcal{V}^{w,t}_E$: subset of the vertices in $\mathcal{N}^w$ representing the end of an activity at the end of period $t \in T_w$;

$\mathcal{B}^w$: set of all the arcs in network $\mathcal{N}^w$;

$\mathcal{B}^w_A$: set of all activity arcs in $\mathcal{N}^w$;

$\mathcal{B}^w_\tau$: set of all transition arcs in $\mathcal{N}^w$;

$\mathcal{B}^w_+(v)$: set of arcs outgoing from vertex $v \in \mathcal{V}^w$;

$\mathcal{B}^w_-(v)$: set of arcs incoming to vertex $v \in \mathcal{V}^w$.

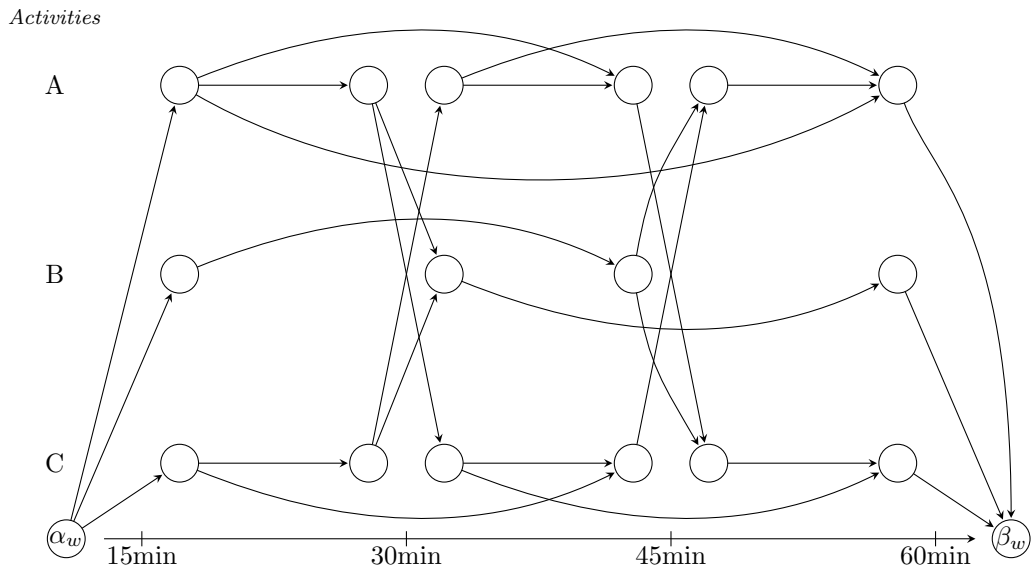

FIGURE 6.1 – Example of a network for a piece of work $w \in W$

Furthermore, with each arc $b \in \mathcal{B}_A^w$, associate a binary parameter $\kappa_{a,t}^b \in \{0,1\}$ that is equal to 1 if and only if arc b covers activity a at period t. Finally, for each piece $w \in W$, define a binary variable X_b^w indicating the flow on each activity arc $b \in \mathcal{B}_A^w$ and a binary variable Y_b^w indicating the flow on every other arc $b \in \mathcal{B}^w \setminus \mathcal{B}_A^w$.

Given this notation, the MAAP can be formulated as the following multi-commodity network flow model with additional constraints (there is one commodity for each piece of work in W) :

$$\text{Minimize} \qquad \sum_{a \in A} \sum_{t \in T} (\bar{c}_a O_{a,t} + \underline{c}_a U_{a,t}) + \sum_{w \in W} \sum_{b \in \mathcal{B}_\tau^w} c_\tau Y_b^w \tag{6.1}$$

subject to :

$$\sum_{w \in W} \sum_{b \in \mathcal{B}_A^w} \kappa_{a,t}^b X_b^w - O_{a,t} + U_{a,t} = d_{a,t}, \quad \forall a \in A,\ t \in T \tag{6.2}$$

$$\sum_{b \in \mathcal{B}_+^w(\alpha_w)} Y_b^w = 1, \quad \forall w \in W \tag{6.3}$$

$$\sum_{b \in \mathcal{B}_+^w(v)} Y_b^w - \sum_{b \in \mathcal{B}_-^w(v)} X_b^w = 0, \quad \forall w \in W,\ t \in T_w,\ v \in \mathcal{V}_E^{w,t} \tag{6.4}$$

$$\sum_{b\in\mathcal{B}^w_+(v)} X^w_b - \sum_{b\in\mathcal{B}^w_-(v)} Y^w_b = 0, \quad \forall w \in W,\ t \in T_w,\ v \in \mathcal{V}^{w,t}_B \tag{6.5}$$

$$\sum_{b\in\mathcal{B}^w_-(\beta_w)} Y^w_b = 1 \quad \forall w \in W \tag{6.6}$$

$$Y^w_b \in \{0,1\}, \quad \forall w \in W,\ b \in \mathcal{B}^w \setminus \mathcal{B}^w_A \tag{6.7}$$

$$X^w_b \in \{0,1\}, \quad \forall w \in W,\ b \in \mathcal{B}^w_A \tag{6.8}$$

$$O_{a,t},\ U_{a,t} \geq 0, \quad \forall a \in A, t \in T. \tag{6.9}$$

The objective function (6.1) aims at minimizing a sum of the weighted over-covering, under-covering, and transition costs. The demand constraints (6.2) allow one to compute the numbers of over-coverings and under-coverings for each activity and each period. Constraints (6.3)–(6.6) ensure that a path (activity sequence) is determined for each piece of work. Finally, constraints (6.7)–(6.9) restrict the feasible domain of each variable. Note that if $\bar{c}_a$ or $\underline{c}_a$ is positive for a given $a \in A$, then both variables $O_{a,t}$ and $U_{a,t}$ for each $t \in T$ cannot take a positive value at the same time in an optimal solution.

With this model and the subsequent ones, only transitions between two consecutive activities are penalized. Transitions from the start of a shift or just after a break are not penalized because their number is known a priori, yielding a constant penalty. Note also that the transitions arcs used in this model offer the possibly of defining transition costs that depend on the new activity (referred to as *activity-dependent transition costs*) or on the sequence of activities involved in this transition (referred to as *sequence-dependent transition costs*)

This multi-commodity network flow model, hereafter referred to as the *MC model*, has a major drawback : the number of transition variables can be quite high. In fact, it is quadratic with respect to the number of activities and linear with respect to the number of periods in a piece of work. In the next subsection, we propose a different formulation that does not involve transition variables. However, transition arcs allow complex costs for transitions such a cost which depends on both a current activity and the new activity that is going to be done.

3.2 Block model

In this subsection, we propose a reformulation of the previous model that reduces substantially the number of constraints and the number of variables by removing all the Y_b^w variables. This reformulation keeps the X_b^w variables from the previous formulation which specify an assignment to an activity block (when taking value 1).

Removing the Y_b^w variables requires modifying the objective function (6.1), replacing the constraints (6.3)–(6.6), and removing the constraints (6.7). The replacement constraints must ensure that each period of each piece of work is covered by a block and that no two consecutive blocks in a piece of work correspond to an assignment to the same activity. This last condition is needed to impose the maximum activity duration constraints.

The proposed block model is as follows :

$$\text{Minimize} \quad \sum_{a\in A}\sum_{t\in T}(\bar{c}_a O_{a,t} + \underline{c}_a U_{a,t}) - c_\tau |W| \; + \sum_{w\in W}\sum_{b\in\mathcal{B}_A^w} c_\tau X_b^w \tag{6.10}$$

subject to

$$\sum_{w\in W}\sum_{b\in\mathcal{B}_A^w} \kappa_{a,t}^b X_b^w - O_{a,t} + U_{a,t} = d_{a,t}, \;\; \forall a\in A, t\in T \tag{6.11}$$

$$\sum_{a\in A_w}\sum_{b\in\mathcal{B}_A^w} \kappa_{a,t}^b X_b^w = 1, \;\; \forall w\in W, \; t\in T_w \tag{6.12}$$

$$\sum_{b\in\mathcal{B}_-^w(E_{a,t_i}^w)} X_b^w - \sum_{a'\in A_w\setminus\{a\}}\sum_{b\in\mathcal{B}_+^w(B_{a',t_{i+1}}^w)} X_b^w \le 0, \;\; \forall w\in W, \; 1\le i<q_w, \; a\in A_w \tag{6.13}$$

$$X_b^w \in \{0,1\}, \;\; \forall w\in W, \; b\in\mathcal{B}_A^w \tag{6.14}$$

$$O_{a,t}, \; U_{a,t} \ge 0, \;\; \forall a\in A, t\in T. \tag{6.15}$$

Noticing that the number of transitions in a piece of work is equal to the number of blocks it contains minus one, the objective function (6.10) is equivalent to (6.1). As before, constraints (6.11) allow one to compute the numbers of over-coverings and under-coverings. The new constraints (6.12) impose that each period of each piece of work be covered by exactly one activity block. Finally, constraints (6.13) ensure that no two consecutive activity blocks in a piece of work correspond to the same activity.

This model, hereafter called the *block model*, contains much less variables and fewer constraints than the MC model. We can also prove using Gale's feasibility theorem (Gale, 1957) that the linear relaxations of both formulations yield the same bound. This allow one to consider activity-dependent transition, but not sequence-dependent transition costs.

3.3 Column generation model

For a shift scheduling problem, Dantzig (1954) proposed a set-covering-type model involving one variable for each feasible shift pattern, where a pattern is defined by a shift start time, a shift end time and a placement of breaks within the shift. Thus, this model requires the enumeration of all feasible shift patterns. In this subsection, we introduce a similar model for the MAAP. With the fact that the shifts are given and assigned to the employees, what must be enumerated is the feasible activity sequences for each piece of work in each shift.

In fact, this model can be derived by applying the well-known Dantzig-Wolfe decomposition principle (Dantzig et Wolfe, 1960) to the MC model (6.1)–(6.9). This decomposition process reformulates the problem into a master program and highlights how this program can be solved by a column generation method. As described in the next section, such a method allows one to avoid the complete enumeration of the variables.

The column generation model relies on the following additional notation. Let S_w be the set of feasible activity sequences that can be assigned to a piece of work $w \in W$. For each piece $w \in W$ and activity sequence $s \in S_w$, let $c_{w,s}$ be the sum of the transition costs incurred in this sequence and let $\kappa_{a,t}^{w,s}$ be binary parameters equal to 1 if sequence s contains an assignment to activity $a \in A_w$ at time period $t \in T_w$, and 0 otherwise. Furthermore, define binary activity sequence variables $\theta_{w,s}$, $w \in W$, $s \in S_w$, to indicate whether or not each activity sequence s for each piece of work w is selected in the solution.

Using this notation, the MAAP can be formulated as the following mixed integer program :

$$\text{Minimize} \qquad \sum_{a\in A}\sum_{t\in T}(\bar{c}_a O_{a,t} + \underline{c}_a U_{a,t}) + \sum_{w\in W}\sum_{s\in S_w} c_{w,s}\theta_{w,s} \tag{6.16}$$

subject to

$$\sum_{s\in S_w} \theta_{w,s} = 1, \qquad \forall w \in W \tag{6.17}$$

$$\sum_{w\in W}\sum_{s\in S_w} \kappa_{a,t}^{w,s}\theta_{w,s} - O_{a,t} + U_{a,t} = d_{a,t}, \qquad \forall a \in A, t \in T \tag{6.18}$$

$$\theta_{w,s} \in \{0,1\}, \qquad \forall w \in W, s \in S_w \tag{6.19}$$

$$O_{a,t},\ U_{a,t} \geq 0, \qquad \forall a \in A, t \in T. \tag{6.20}$$

The objective function (6.16) aims at minimizing the total costs, that is, the sum of the over-covering, under-covering, and transition costs. Constraints (6.17) stipulate that a feasible activity sequence must be assigned to each piece of work. Constraints (6.18) allow one to compute the number of over- and under-coverings for each activity in each period. Finally, constraints (6.19) and (6.20) restrict the domains of the variables. Using the Dantzig-Wolfe decomposition principle, it is easy to show that the linear relaxation of this model also yields the same bound as that provided by the linear relaxation of the MC model.Indeed, as shown in section 4.3, the subproblems are basic shortest path problems and then have the integrity property.

Because one of the objectives is to minimize the number of transitions, an optimal solution (and the intermediate fractional-valued solutions visited to reach it) often contains activity sequences with a small number of transitions. Hence, grouping the constraints (6.18) per activity and sorting them in each group in chronological order of their time period, the positive coefficients $\kappa_{a,t}^{w,s}$ of the positive-valued variables $\theta_{w,s}$ in an optimal solution would be seen as long sequences of consecutive ones. This observation allows one to reduce the density of the constraint matrix by replacing such a constraint (6.18) by the difference between this constraint and the preceding one (as long as both constraints are associated with the same activity). This technique has already been applied successfully in (Haase *et al.*, 2001) and (Rekik *et al.*, 2009). Hence, numbering the time periods in chronological order from 1 to n (that is, $T = \{1, \ldots, n\}$), this transformation yields the following equivalent mixed integer program for the MAAP :

$$\text{Minimize} \qquad \sum_{a\in A}\sum_{t\in T}\left(\bar{c}_a O_{a,t} + \underline{c}_a U_{a,t}\right) + \sum_{w\in W}\sum_{s\in S_w} c_{w,s}\theta_{w,s} \tag{6.21}$$

subject to

$$\sum_{s\in S_w} \theta_{w,s} = 1, \qquad \forall w \in W \tag{6.22}$$

$$\sum_{w\in W}\sum_{s\in S_w}\left(\kappa^{w,s}_{a,t} - \kappa^{w,s}_{a,t-1}\right)\theta_{w,s} - O_{a,t} + U_{a,t} + O_{a,t-1} - U_{a,t-1} = d_{a,t} - d_{a,t-1}, \qquad \forall a \in A, t \in T \tag{6.23}$$

$$\theta_{w,s} \in \{0,1\}, \qquad \forall w \in W, s \in S_w \tag{6.24}$$

$$O_{a,t},\ U_{a,t} \geq 0, \qquad \forall a \in A, t \in T, \tag{6.25}$$

where we assume that, when $t = 1$ in constraint set (6.23), $\kappa^{w,s}_{a,t-1} = d_{a,t-1} = 0$ and there are no variables $O_{a,t-1}$ and $U_{a,t-1}$.

Preliminary computational tests showed that, in practice, model (6.21)–(6.25) is much easier to solve than model (6.16)–(6.20). Consequently, model (6.16)–(6.20) will not be used for the rest of the paper. Hereafter, model (6.21)–(6.25) will be referred to as the *CG model*.

4 Solution methods

In this section, we describe several methods for solving the MAAP models introduced in the previous section. These methods are an exact branch-and-bound method, two truncated branch-and-bound methods, with and without variable fixing, and a column generation heuristic. Finally, we discuss how these last three methods can be embedded into a rolling horizon procedure to further speed up the solution process.

4.1 Exact branch-and-bound

The MC, the block, and the CG models can all be solved exactly using a traditional branch-and-bound algorithm with or without cutting planes (see Nemhauser et Wolsey, 1988, Section II.4.2). On one hand, state-of-the-art commercial mixed integer

programming (MIP) solvers, implementing such an algorithm, can be directly used for solving the MC and the block model. On the other hand, the CG model requires an ad hoc branch-and-bound algorithm, called branch-and-price (Barnhart *et al.*, 1998; Desaulniers *et al.*, 2005). Preliminary tests showed that a branch-and-price algorithm would not be competitive with a commercial MIP solver for small-sized instances, and would require too much time for larger instances. Hence, no exact branch-and-price method was developed for this paper.

4.2 Truncated branch-and-bound with and without variable fixing

We propose two acceleration strategies for speeding up the branch-and-bound algorithm applied to the block model, yielding two heuristic methods. The first strategy is very simple and consists of imposing a time limit to the solution process. When this limit is reached with at least one integer solution found, the best integer solution found is kept as the computed solution. Otherwise, if there is no integer solution found at the time limit, then the solution process continues and stops when it finds a first integer solution. This heuristic is called the *truncated branch-and-bound heuristic*.

To further speed up this first heuristic, the second strategy fixes the value of a large number of variables to zero after solving the linear relaxation of the block model. Given the computed linear relaxation solution, all zero-valued variables with a strict positive reduced cost in this solution are permanently fixed to zero for all the nodes to explore in the branch-and-bound tree. This heuristic, which is an accelerated version of the truncated branch-and-bound heuristic, is called the *zero fixing heuristic*.

4.3 Column generation heuristic

Column generation (Dantzig et Wolfe, 1960; Desaulniers *et al.*, 2005; Lübbecke et Desrosiers, 2005) can be used for solving the linear relaxation of integer programs that contain a huge number of variables such as model (6.21)–(6.25). This iterative exact method solves at each iteration a restricted master problem and one or several subproblems. The restricted master problem is the linear relaxation of the integer program restricted to a subset of its variables. In our case, it corresponds to model

(6.21)–(6.25) where the integrality requirements (6.24) are relaxed and only a subset of the sequence variables $\theta_{w,s}$ is considered. The restricted master problem is solved by a linear programming algorithm such as the primal simplex method to provide a pair of primal and dual solutions. The subproblems aim at finding negative reduced cost variables $\theta_{w,s}$ among those that are not part of the current subset, where the reduced costs are computed with respect to the dual solution of the current restricted master problem. On one hand, when such variables are found, they are added to the subset of known variables in the restricted master problem before starting another iteration. On the other hand, when no such variables exist, it means that the solution of the current restricted master problem is optimal for the original linear relaxation and the algorithm stops. In our case, the subproblems are shortest path problems and there is one such subproblem for each piece of work $w \in W$. Indeed, these subproblems can be deduced from the application of the Dantzig-Wolfe decomposition principle applied to the MC model. There feasible domains are defined by the constraint sets (6.3)–(6.8) which are separable by piece of work. Thus, the subproblem for piece $w \in W$ is a shortest path problem defined over the network $\mathcal{N}^w$ introduced in Section 3.1. In a column generation algorithm, the arc costs must, however, be replaced by arc reduced costs as follows.

Let σ_w, $w \in W$, and $\pi_{a,t}$, $a \in A$, $t \in T$, be the dual variables associated with constraints (6.22) and (6.23), respectively. The reduced cost $\tilde{c}_{w,s}$ of a variable $\theta_{w,s}$ is thus given by

$$\tilde{c}_{w,s} = c_\tau h_s - \sum_{a \in A_w} \sum_{t \in T_w} (\kappa^{w,s}_{a,t} - \kappa^{w,s}_{a,t-1}) \pi_{a,t} - \sigma_w, \tag{6.26}$$

where h_s is the number of transitions in activity sequence s. Denoting by a_{sr}, $r = 1, 2, \ldots, h_s + 1$, the r^{th} activity performed in sequence s and by t^B_{sr} and t^E_{sr} the time periods (possibly the same) in which this r^{th} activity begins and ends, respectively, this reduced cost can be rewritten as follows :

$$\tilde{c}_{w,s} = c_\tau h_s - \sum_{r=1}^{h_s+1} (\pi_{a_{sr},t^B_{sr}} - \pi_{a_{sr},t^E_{sr}+1}) - \sigma_w, \tag{6.27}$$

where we assume that $\pi_{a,t+1} = 0$ if $t = n$ for any activity $a \in A$.

Now, recall that every sequence s is represented by a path from the source node α_w to the sink node β_w in $\mathcal{N}^w$. Consequently, finding a least reduced cost variable

$\theta_{w,s}$ corresponds to finding a shortest path from α_w to β_w in $\mathcal{N}^w$ with the following arc reduced costs :

$$\tilde{c}_b = \begin{cases} c_\tau & \text{if } b \in \mathcal{B}^w_\tau \text{ is a transition arc} \\ -\sigma_w & \text{if } b \in \mathcal{B}^w_+(\alpha_w) \text{ is a start arc} \\ 0 & \text{if } b \in \mathcal{B}^w_-(\beta_w) \text{ is an end arc} \\ \pi_{a,t'+1} - \pi_{a,t} & \text{if } b = (B^w_{a,t}, E^w_{a,t'}) \in \mathcal{B}^w_A \text{ is an activity arc.} \end{cases}$$

Because the network $\mathcal{N}^w$ is acyclic, the subproblems are solved by a label-setting algorithm for acyclic networks (see Ahuja *et al.*, 1993, Chapter 4).

To obtain an integer solution for the MAAP, we use two heuristics, where the second one is executed only when the first one is not successful. The first heuristic consists of trying to solve directly with a MIP solver the model (6.21)–(6.25) restricted to the variables contained in the last restricted master problem solved (no additional columns are thus generated). A maximum time limit is imposed for this heuristic, which is deemed successful only when an integer solution is found within that time limit and the gap between the cost of that solution and the linear relaxation opimal value is less than the minimum cost sum of an under-covering and an over-covering. Integer search is stopped for avoiding losing two much time in improving tertiary objective. The second heuristic consists of the following iterative rounding procedure. Given the linear relaxation solution computed by column generation, a variable $\theta_{w,s}$ with the highest fractional value is selected and permanently fixed at one. After fixing this variable value, column generation is started over again to reoptimize the linear relaxation ensuing from this decision and rounding again another variable. This rounding procedure goes on until the column generation method provides an integer linear relaxation solution.

This two-step approach was adopted for the following reasons. The total computational time of the first heuristic (when not limited) can vary significantly from one instance to another, while that of the second heuristic is rather predictable with respect to the instance size. However, the first heuristic often produces very high-quality solutions when the total computational time is small (because of low integrality gaps), whereas the rounding heuristic can sometimes yield solutions of doubtful

quality. Consequently, the proposed approach first tries to take advantage of the first heuristic without losing too much time, and, if needed, resorts to the second heuristic for ensuring the computation of an integer solution in a reasonable amount of time.

4.4 Rolling horizon

As frequently used for tackling difficult instances over a relatively long planning horizon, we rely on a rolling horizon technique to reduce the total computational time. The principle of such a technique is simple : the overall planning horizon is divided into overlapping time slices of equal length and problems restricted to these time slices are solved sequentially in chronological order. Typically, overlapping occurs only between two consecutive time slices. This time decomposition strategy is defined by two parameters, namely, ℓ the length of a time slice and o the overlapping length. For example, for a 1440-minute horizon, the time slices for $\ell = 600$ and $o = 150$ are [0, 600], [450, 1050], and [900, 1440]. Every problem restricted to a time slice involves only the variables associated with the pieces of work that have an intersection with the time slice, besides the under- and over-covering variables. Furthermore, all activity assignments made in the previous time slices and effective for time periods prior to the current time slice are enforced. Thus, all pieces of work that begin prior to the current time slice and end after its beginning are partly fixed. We propose to combine this rolling horizon procedure with the truncated branch-and-bound heuristic, the zero fixing heuristic, and the column generation heuristic to yield three additional heuristics for the MAAP.

5 Computational experiments

To compare and evaluate the performances of the different models and solution methods presented above, we conducted a series of computational experiments on randomly generated instances. In this section, we report the results of these experiments after describing the test instances and our experimental plan.

5.1 Test instances

Our computational tests were performed on randomly generated instances that mimic the main characteristics of real-life instances thanks to our own knowledge of supermarket schedules. Each instance is determined by four parameters : the number of days in the horizon, the number of shifts per day, the number of activities, and a random seed number. The other parameters used by our instance generator have been set to produce realistic instances that are nontrivial to solve. For instances defined over multiple-day horizons, we assume that the employees work around the clock so that the problem cannot be separated by day.

The characteristics of the instances generated are as follows. The average length of a shift is around 8 hours and its standard deviation is relatively small. Every shift contains a single 30-minute break in the middle. There are around 80% daytime shifts and 20% nighttime shifts. The total demand in each time period is distributed more or less evenly among the different activities. The minimal activity duration is about 1 hour and the maximal activity duration is about 3 hours. On average, an employee is qualified for approximately 75% of the activities. For all the instances, the horizon is discretized into 15-minute time periods, the under-covering and over-covering costs are $\underline{c}_a = 1500$ and $\bar{c}_a = 150$ for all activity a, and the transition cost is $c_\tau = 15$. These costs clearly highlight a (nonstrict) hierarchy between the three objectives : minimizing first the number of under-coverings, second the number of over-coverings, and third the number of transitions. Furthermore, it is easy to prove that, in this case, minimizing the number of under-coverings also minimizes the number of over-coverings (in fact, each under-covering saved yields an over-covering saved). Therefore, the objective is rather a two-level one, where the cost of one under-covering is equal to 1500+150=1650. Note that, for this reason, we report in the results the number of under-coverings that must be minimized in priority

5.2 Experimental plan and setting

Our experimental plan is divided into two phases. The first phase concentrates on small-sized instances. Its goal is to validate the quality of the solutions produced by the column generation and the zero fixing heuristics on instances that can be solved to optimality with an exact method. It also allows one to compare the multi-

commodity network flow model and the block model. In the second phase, we compare the performance of the different rolling horizon heuristics for solving medium- to large-sized instances. This phase also shows the limits of the proposed methodologies.

All tests were performed on an *Intel®Core™2 CPU 6700* clocked at 2.66GHz with 4GB RAM. The column generation heuristic was implemented using the GENCOL library, version 4.5, a state-of-the-art software developed over the last twenty years that is now commercialized by *Kronos Inc.* We used the XPRESS-MP solver of *Fair Isaac Corporation* for solving all linear programs and integer programs. Default parameters were used for this solver except for the branch-and-bound stopping criterion used by the heuristic methods. In fact, to speed up the computational times, we chose to fix the absolute tolerance that determines whether or not the global search for an integer solution should continue, to 1650, which is equal to the sum of the cost of one under-covering and one over-covering.

5.3 Results for small-sized instances

For the first-phase experiments, we considered two classes of small-sized instances. Class 1 involves a 1-week horizon, 20 shifts per day, and 5 activities, while class 2 involves a 1-day horizon, 50 shifts per day, and 10 activities. For each class, five instances was randomly generated. Each of these ten instances were solved exactly using the MC and the block model, as well as heuristically using the column generation heuristic (*Col Gen*) and the zero fixing heuristic (*Zero Fix*).

For these instances, the average size of the models per instance class is given in Table 6.2. One can observe that the block model is between 40 to 50% smaller than the MC model. The density of the constraint matrix (not reported in the table) for the block model is, however, approximately five times higher. We also notice that the class 2 instances yield smaller model sizes for the three different models. As for the CG model, we see that its average number of constraints is much smaller for the class 2 instances because this number is $O(|T||A| + |W|)$.

The results of our experiments are reported in Tables 6.3 and 6.4. For each instance and each solution method, we provide the value of the computed solution with the number of under-coverings in parentheses (which must be minimized in priority) and the total computational time in seconds. The best value got by a heuristic is

	MC Model		Block Model		CG Model
Instances	Variables	Constraints	Variables	Constraints	Constraints
Class 1	89,534	28,264	48,688	16,942	3,553
Class 2	27,748	9,675	14,622	5,903	986

TABLEAU 6.2 – Average sizes of the models for the two instance classes

highlighted in boldface face. The average computational time per instance for each solution method is reported at the bottom line. From the results in Table 6.3, we make the following observations. All methods found an optimal solution for all instances, except the column generation heuristic, which failed to minimize the number of transitions for one of the instances. In terms of computational time, the exact branch-and-bound method performed much better with the block model than with the MC model, while the zero fixing heuristic performed better than the column generation heuristic. Overall, the computational time of all methods is satisfactory, the zero fixing heuristic achieving the best average time per instance (31 seconds).

The results in Table 6.4 are different. Again, the zero fixing heuristic succeded in obtaining an optimal solution for all instances, while the column generation heuristic could only find a solution with one additional under-covering for three of the five instances. We observe that the computational times have increased significantly (compared to those reported in Table 6.3) except for the column generation method. This is rather surprising given that the model sizes have decreased considerably, but it can be explained by an increased number of possibilities to meet the demand in every time period, yielding larger integrality gaps. On the other hand, the reduced size of the column generation model and its heuristic branching prevents a computational time increase. Comparing both exact methods, we can conclude again that the block model is superior to the MC model. This latter model will thus be omitted from the phase two tests. For the heuristic methods, the column generation heuristic is faster while the zero fixing heuristic gives better quality solutions. One can observe that the average computational time of the zero fixing heuristic is a slightly higher than that of the exact branch-and-bound method with the block model. This can be explained by the fact that the zero fixing heuristic can greatly reduce the feasible domain after fixing variables, yielding a large number of infeasible branch-and-bound nodes.

All these methods were also tested on medium-sized instances. They all failed to

find good-quality solutions in reasonable computational times.

5.4 Results for larger instances with the rolling horizon procedure

To solve medium- to large-sized instances in relatively short computational times, we combined the column generation, the truncated branch-and-bound, and the zero fixing heuristic with the rolling horizon procedure described in Section 4.4. The resulting three methods are denoted *Horizon CG*, *Horizon TBB*, and *Horizon ZF*. Recall that the block model is used for the last two methods.

Based on the results obtained from independent preliminary tests, we set the values of the rolling horizon procedure parameters to $\ell = 720$ and $o = 240$. Furthermore, the time limit used by the truncated branch-and-bound heuristic in each time slice of the Horizon TBB and Horizon ZF methods was set to 1.5 multiplied by the average time taken by the Horizon CG method in each time slice. This allows sufficient time to explore a significant number of branch-and-bound nodes, while avoiding excessive computational times in certain time slices.

First, we checked the efficiency of the rolling horizon heuristics for the small-sized instances. Tables 6.5 and 6.6 report the results obtained, which can be compared to those reported in Tables 6.3 and 6.4. These results show that the rolling horizon heuristics are well calibrated with similar computational times. The quality of the computed solutions is very good for the class 1 instances ; in fact, the optimal solution in terms of under-coverings was found 11 times out of 15 tests. For class 2, the methods delivered different results. The Horizon CG method performed best with three optimal solutions, while the Horizon TBB method could not find any optimal solution and did very poorly on the first two instances. The Horizon ZF method performed well even if it get the best solution only for one instance.

Next, we tested the rolling horizon heuristics on two classes of medium-sized instances. Class 3 involves a 1-week horizon, 50 shifts per day, and 7 activities, while class 4 considers a 2-day horizon, 75 shifts per day, and 12 activities. Again, five instances were generated for each class. The results of our tests are presented in Tables 6.7 and 6.8, which provide two additional columns. The z_{LP} column indicates the linear relaxation value of the block model, and the column *Time* 1^{st} gives the time

Instance Id	Exact B&B				Heuristic			
	MC Model		Block Model		Col Gen		Zero Fix	
	Value	Time	Value	Time	Value	Time	Value	Time
2732	33600 *(17)*	200	33600 *(17)*	21	**33600** *(17)*	88	**33600** *(17)*	23
1024	45720 *(25)*	114	45720 *(25)*	17	45750 *(25)*	39	**45720** *(25)*	12
1773	34065 *(18)*	50	34065 *(18)*	17	**34065** *(18)*	28	**34065** *(18)*	10
5553	41415 *(22)*	92	41415 *(22)*	34	**41415** *(22)*	62	**41415** *(22)*	23
4657	37200 *(20)*	129	37200 *(20)*	135	**37200** *(20)*	42	**37200** *(20)*	86
Average		117		45		52		31

TABLEAU 6.3 – Results for 7 days, 20 shifts per day, and 5 activities (class 1)

Instance Id	Exact B&B				Heuristic			
	MC Model		Block Model		Col Gen		Zero Fix	
	Value	Time	Value	Time	Value	Time	Value	Time
5226	15045 *(8)*	107	15045 *(8)*	93	**15045** *(8)*	20	**15045** *(8)*	62
5135	11820 *(6)*	155	11820 *(6)*	98	11865 *(6)*	18	**11820** *(6)*	88
1808	20190 *(11)*	2681	20190 *(11)*	1190	21780 *(12)*	48	**20190** *(11)*	1521
5066	5520 *(2)*	716	5520 *(2)*	294	7215 *(3)*	74	**5520** *(2)*	214
8854	10440 *(5)*	839	10440 *(5)*	321	12120 *(6)*	138	**10440** *(5)*	346
Average		900		399		60		446

TABLEAU 6.4 – Results for 1 day, 50 shifts per day, and 10 activities (class 2)

taken by the exact branch-and-bound method applied to the block model for finding a first integer solution.

For these results, we have the following remarks. The solutions produced by the Horizon TBB heuristic are of poor quality for both instance classes, while those computed by the Horizon ZF heuristic are relatively good for the class 3 instances, but also poor for the class 4 instances. In contrast, the solutions of the Horizon CG heuristic are all very good if not optimal with respect to the number of under-coverings (this can be seen by comparing their values with the z_{LP} lower bounds). The average computational times are acceptable (less than 18 minutes) for all methods and similar except for the Horizon TBB heuristic, which needed more time for solving the class 4 instances. This can be explained by the difficulty of obtaining a first integer solution in certain times slices when directly using a MIP solver as a black box. Notice that the

Instance Id	Horizon CG		Horizon TBB		Horizon ZF	
	Value	Time	Value	Time	Value	Time
2732	**33600** *(17)*	17	**33600** *(17)*	19	**33600** *(17)*	15
1024	45765 *(25)*	10	**45750** *(25)*	12	45780 *(25)*	11
1773	**35700** *(19)*	9	40665 *(22)*	13	**35700** *(19)*	11
5553	41430 *(22)*	15	**41415** *(22)*	14	**41415** *(22)*	12
4657	37215 *(20)*	11	38880 *(21)*	14	**37200** *(20)*	14
Average		12		14		13

TABLEAU 6.5 – Rolling horizon results for 7 days, 20 shifts per day, and 5 activities (class 1)

Instance Id	Horizon CG		Horizon TBB		Horizon ZF	
	Value	Time	Value	Time	Value	Time
5226	**15060** *(8)*	19	48045 *(28)*	28	15090 *(8)*	33
5135	**11835** *(6)*	18	59640 *(35)*	31	**11835** *(6)*	114
1808	**21795** *(12)*	86	26760 *(15)*	94	23445 *(16)*	104
5066	**7185** *(3)*	237	7215 *(3)*	184	10440 *(5)*	135
8854	**10470** *(5)*	70	17085 *(9)*	105	17055 *(9)*	153
Average		86		88		108

TABLEAU 6.6 – Rolling horizon results for 1 day, 50 shifts per day, and 10 activities (class 2)

time required to find a first integer solution (which is not necessarily of good quality) by an exact branch-and-bound method is on average 5.5 hours ; that is, approximately 30 times slower than the Horizon CG heuristic.

Finally, Table 6.9 reports results for five large-sized instances (class 5) that involve a 1-week horizon, 100 shifts per day, and 15 activities. These results clearly show that only the Horizon CG heuristic is viable, both in terms of solution quality and computational time. The other two heuristics provide very poor quality solutions. The other two heuristics provide very poor quality solutions. In particular, the Horizon ZF heuristic yields certain solutions that contain almost fifty times more under-coverings than the CG solutions. This can be explained by the fact the size of the problem solved each time slice is similar to that of the class 2 instances, which required much more computational time to be solved by the ZF heuristic than by the CG heuristic

Instance	Horizon CG		Horizon TBB		Horizon ZF		Block Model	
Id	Value	Time	Value	Time	Value	Time	z_{LP}	Time 1^{st}
237	**41310** *(18)*	333	163470 *(92)*	411	44550 *(20)*	529	41142.5	16050
1007	**46605** *(21)*	363	110790 *(60)*	586	48180 *(22)*	546	46388.3	16108
4369	41610 *(18)*	473	157140 *(88)*	828	**39870** *(17)*	650	38966.8	25029
156	25890 *(8)*	612	118215 *(64)*	850	**25830** *(8)*	816	25710.4	21271
5216	50460 *(23)*	543	184125 *(104)*	639	**50325** *(23)*	678	50150.0	13020
Average		465		663		644		18296

TABLEAU 6.7 – Rolling horizon results for 7 days, 50 shifts per day, and 7 activities (class 3)

Instance	Horizon CG		Horizon TBB		Horizon ZF		Block Model	
Id	Value	Time	Value	Time	Value	Time	z_{LP}	Time 1^{st}
4225	**21810** *(10)*	655	64545 *(36)*	995	43125 *(23)*	654	19079.1	18594
2435	**19950** *(9)*	252	49485 *(27)*	461	31410 *(16)*	632	19636.5	3511
2106	**25110** *(12)*	878	56400 *(31)*	1452	51420 *(28)*	809	24100.8	15400
1855	**16965** *(7)*	1068	41430 *(22)*	1654	51390 *(28)*	631	16590.1	18685
9863	**12195** *(4)*	388	28665 *(14)*	767	28590 *(14)*	693	11931.1	19216
Average		648		1066		684		15081

TABLEAU 6.8 – Rolling horizon results for 2 days, 75 shifts per day, and 12 activities (class 4)

as reported in Table 6.4. Hence, the computational time limit imposed per time slice for the Horizon ZF heuristic is really disadvantaging this method.

6 Conclusions

This paper has introduced the multi-activity assignment problem (MAAP) to the operations research community. For this problem, we proposed three integer programming formulations and developed three heuristics based on branch-and-bound or column generation methods. These heuristics were also embedded into a rolling horizon procedure to tackle very large-scale instances. Extensive computational tests on randomly generated instances that mimic real-life instances showed that small-sized instances can be solved to optimality using a commercial MIP solver. For solving

Instance	Horizon CG		Horizon TBB		Horizon ZF		Block Model		
Id	Value	Time	Value	Time	Value	Time	z_{LP}	Time	1^{st}
6832	**54525** *(20)*	6504	526275 *(306)*	13219	1734225 *(1038)*	9139	53742.0	>24h	
1310	**62085** *(24)*	8284	508980 *(295)*	14336	1146285 *(681)*	9490	61287.3	>24h	
3407	**37590** *(9)*	7747	263160 *(146)*	13290	760095 *(447)*	6331	36677.7	>24h	
1248	**41400** *(12)*	5480	397410 *(228)*	13010	441765 *(255)*	5658	40518.5	>24h	
4261	**60600** *(24)*	5224	502515 *(292)*	10971	679215 *(399)*	5578	59806.5	>24h	
Average		6647		12965		7239		>24h	

TABLEAU 6.9 – Rolling horizon results for 7 days, 100 shifts per day, and 15 activities (class 5)

large-sized instances, the column generation heuristic combined with a rolling horizon procedure produced the best results. In fact, it was able to compute high-quality solutions in less than 2 hours of computational time for instances involving up to 100 shifts per day and 15 activities over a 7-day horizon.

Based on our experimental results, we conclude that integrating a commercial MIP solver with basic heuristic strategies within a rolling horizon procedure is not sufficient to produce good-quality results for the MAAP. Indeed, the computational time of a MIP solver is rather unstable from one instance to another and there is a high chance that one or several problems solved in the rolling horizon procedure may yield high computational times or bad-quality solutions when the branch-and-bound algorithm is truncated. This drawback is not present in the proposed column generation heuristic implemented using the flexible GENCOL software library.

Given that this problem is new, many directions can be explored for further research. For instance, an work which is not interruptible can be introduced to cover all the type of work in a company, or setup time can be added between activities to modelize in a more subtle the way lack of productivity.

ARTICLE 2 : A TWO-STAGE HEURISTIC FOR MULTI-ACTIVITY AND TASK ASSIGNMENT TO WORK SHIFTS

Article soumis à *Computer and Industrial Engineering* (2010) et écrit par :

QUENTIN LEQUY

École Polytechnique de Montréal and GERAD

GUY DESAULNIERS

École Polytechnique de Montréal and GERAD

MARIUS M. SOLOMON

Northeastern University and GERAD

Abstract

The multi-activity assignment problem consists of assigning interruptible activities to given work shifts so as to match as much as possible for each activity a demand curve in function of time. In this paper we consider an extension to this problem, called the multi-activity and task assignment problem, that additionally considers the assignment of uninterruptible pieces of work, called tasks. These possess properties such as worker qualifications, time windows for completion, fixed lengths and precedence relationships. We propose a mixed integer programming formulation and a two-stage method to solve this problem. The first stage consists of an approximation model to assign tasks approximately taking into account the activities and the second involves a heuristic for assigning activities and reassigning tasks at the same time. We suggest four different strategies for reassigning tasks. We conducted extensive computational tests on randomly generated instances in order to validate our method and to compare the various strategies. One strategy proved universally best when compared to the other three policies.

Keywords : Activity assignment, work shifts, tasks, precedence relationships, column generation.

1 Introduction

This paper focuses on environments where employees work in shifts during which they fulfill various customer demands and/or perform some tasks. This problem belongs to the shift scheduling problem class, a classic operations research field. It is of great practical importance since it aims at building employee schedules for companies, independent of whether they are in manufacturing, service or government.

We consider the work of an employee to be divided between tasks and activities. A task is an uninterruptible piece of work, such as putting up a display, whereas an activity can easily be interrupted at any time by replacing one employee with another, such as cashiers. The former has an isolated and unique nature and is performed by a single skilled employee before a due date and may require the completion of some other tasks prior to its start. The latter has a continuous and recurrent nature. While we address the general multi-activity case where it is possible to assign several

activities to an employee, special cases of single activity employee assignments may be considered. For each activity, a separate algorithm can determine the ideal number of employees to handle it at any time during the planning horizon, providing a demand curve. The whole schedule must cover, as close as possible, the demand curve for each activity. To illustrate these concepts, we can assume that an activity corresponds to servicing customers in real-time, whereas tasks do not involve a direct contact with the customers.

Finding a schedule for each employee includes many features. It must be composed of a number of working days per month and some days off. A working day matches one shift, defined by a starting time and an ending time, and this shift may contain one or several specified break periods. Therefore, the shift consists of activities, tasks, and break periods. Each part of the schedule (days off, definition of shifts and break periods, assignment of activities and tasks) must respect labor and collective agreement rules which in general are quite complex. Companies want to design employee schedules that reduce personnel costs without affecting customer satisfaction. Furthermore, the schedules should satisfy as many of the employees preferences as possible.

Trying to build schedules with these objectives leads to highly combinatorial optimization problems, even for small problem instances. They involve multiple tasks and activities with significantly diverse demand curves which must be assigned to different work shifts belonging to skilled employees. Due to its complexity, solving the overall problem necessitates decomposing it in steps that are solved sequentially. First, the days off of each employee are scheduled over a given horizon (one year for instance). Then, unspecified shifts are built to satisfy aggregated demand of activities and tasks. Next, each shift is assigned to a specific employee which is not in day off. Finally, activities and tasks are assigned to work shifts, by taking skills into account and with a better forecast of the demand for all activities. This fourth step, which is the subject of our analysis, is usually performed less than one week prior to the week of operations.

Depending on the company, the four steps are performed independently or some of them are integrated in the same module. For instance, the *tour scheduling* problem attempts to merge the first two steps. The surveys of Ernst *et al.* (2004a,b) give many references related to *days-off scheduling*, *shift scheduling*, *shift assignment* and

task assignment, but nothing relevant for *activity assignment*. In Ernst *et al.* (2004b), *Task assignment* refers to "the process of allocating a set of tasks, with specified start and end times and skill requirements, between a group of workers who have typically already been assigned to a set of working shifts." In this paper, we adhere to this definition but combines task assignment with activity assignment. Ernst *et al.* (2004b) cite around 30 references on task assignment. All these references address problem variants that are fundamentally different from ours. For example, they consider fixed task start times, tasks performed in different locations, tasks with different starting and ending locations, assignment of a single task to each shift, dynamic task assignment, as well as combined shift scheduling and task assignment. None of these works involve simultaneous task and activity assignment.

The multi-activity assignment problem (MAAP) has been addressed in a few works. Omari (2002) considered a MAAP for air traffic controllers with complex additional rules such as synchronization between trainees and instructors. Vatri (2001) integrated shift scheduling, shift assignment and activity assignment problems in a simplified context. To reduce problem complexity breaks are not scheduled in the shifts. To alleviate this shortcoming, Bouchard (2004) propose methods that can handle breaks through accelerating heuristics. Lequy *et al.* (2009) proposed three different models and several heursitics to solve the basic MAAP problem. All these works based their solution approaches on a rolling horizon heuristic which decomposes instances into easier to solve subproblems. These subproblems are solved sequentially using a column generation heuristic embedded in a modified branch-and-price framework. In a related research, Côté *et al.* (2009) used techniques from constraint programming to create compact mixed integer programs for a combined shift scheduling and multi-activity assignment problem for unidentified employees. Constraint programming is used here for filtering certain feasiblity rules concerning breaks, rest periods and work streches.

Bard et Wan (2006) addressed a MAAP where activities are executed in different workstation groups and are not subject to constraints on minimum and maximum activity assignment durations as in the works cited above. A combination of an integer programming model based on a multi-commodity network and a tabu search technique provides results for test data from a U.S. Postal Service mail processing and distribution center. The authors classified this problem as a task assignment problem,

but its definition corresponds to our MAAP definition. In a follow-up paper, Bard et Wan (2008) consider additional restrictions on movements between the workstation groups.

In this paper we consider the multi-activity and task assignment problem (MATAP) which is an extension of the MAAP described in Lequy *et al.* (2009). The work shifts are fixed and already assigned to the employees. Activities and tasks must be assigned to these shifts. Task characteristics include due date, skills and precedence relationships. We aim at solving instances where task workload represents only up to 10 percent of the activity workload, even though, in theory, our model can handle larger percentages. To do so, we provide a model and a two-stage heuristic method. The first stage assigns only tasks based on an approximation model. The second stage allocates activities using a column generation heuristic wherein either tasks obtained at the previous stage are fixed or activity assignments include some flexibility on task assignments.

This paper offers the following contributions. First, we introduce a new kind of problem dealing with activities and tasks at the same time. Second, we propose an integer programming model that provides lower bounds and, for small instances, exact solutions. Third, for tackling large-size problems, we develop a two-stage heuristic with four strategies. Finally, we use the computational results and especially a performance profile to compare our strategies to determine the best one.

The paper is organized as follows. Section 2 provides a precise verbal definition of the problem. Next, in Section 3, we suggest a mixed integer programming formulation for the MATAP. The direct solution of this formulation is complex enough to force us to propose a two-stage heuristic method described in Sections 4 and 5. Some accelerating strategies are discussed in Section 6. Section 7 reports our computational experiments and Section 8 presents our conclusions.

2 The multi-activity and task assignment problem

In this section, we provide a detailed definition of the MATAP. First, we introduce the necessary vocabulary.

2.1 Vocabulary

Time horizon, time period : The MATAP is defined over a time horizon that can last one day to one week. This time horizon is discretized into disjoint time periods of equal length (typically, 15 or 30 minutes). In the following, we assume that all times correspond to the start or the end of a time period, and all durations are integer multiples of the time period length. Under this assumption, we express all times and durations in time period units.

Activity : An activity is a type of work that can be interrupted at any time. The assignment of an employee to an activity is restricted by a minimum and a maximum duration (number of consecutive time periods) that depend on this activity.

Demand, under-covering : Each activity requires a nonnegative number of employees in each time period of the horizon. These numbers define a demand curve. An under-covering occurs for each employee less than the demand for an activity in one time period. An activity-dependent cost is incurred for each under-covering. These under-covering costs represent the loss in productivity for the corresponding activity and one time period.

Task, time window, precedence relationships : A task is a type of work that cannot be interrupted and requires a single employee. It is defined by a duration and a time window during which it must be assigned. A task can, however, remain unassigned at the expense of a high penalty cost. There may exist precedence relationships between pairs of tasks. Each of them stipulates that the second task cannot begin before the completion of the first task. We distinguish between two types of precedence relationships. A strong precedence relationship implies that either both tasks are assigned or none of them is assigned. A weak precedence relationship implies only that the first task must be assigned to allow the second one to be assigned.

Employee qualifications : An employee possesses certain qualifications (skills). She/he can be assigned to an activity or a task only if she/he is skilled for it.

Shift, segment, and break : A shift is associated with an employee. It is defined by a pair of start and end times and by start and end times for the breaks it contains. These breaks divide the shift into segments during which the employee

must be assigned to activities or tasks. No idleness is allowed during a segment as activity over-covering is preferred.

Transition : A transition occurs when an employee switches from an activity to another or from a task to an activity (or vice versa) within the same segment. Every transition incurs a cost that represents the loss in productivity due to the transition or reflects the inconvenience perceived by an employee that changes the type of work too frequently.

2.2 MATAP definition

Given a time horizon, a set of employees with their qualifications, a set of shifts for these employees, a set of tasks, and a set of activities with their demand curves, the MATAP consists of filling every segment of every shift with activities and tasks such that the following constraints are satisfied :

- exactly one activity or one task must be assigned in each time period of each segment ;
- the assignment of each activity must respect the activity duration bounds ;
- an employee cannot be assigned to an activity or a task without being qualified for it ;
- a task cannot be assigned outside its time window ;
- all strong and weak precedence relationships between the tasks must be respected.

The objective is to minimize the total of the costs incurred by the task and activity under-coverings and the transitions. In practice, the costs are set in such a way that under-covering is minimized first and the number of transitions is minimized second. Note that task over-covering is not allowed because it is always preferable to assign employees to activities for improving customer service. Note also that no constraints nor costs link the segments of the same shift. Therefore, the shift notion is not necessary if we associate the segments directly with the employees.

3 Mathematical formulation for the MATAP

In this section, we introduce a mathematical formulation for the MATAP that is a generalization of one of the models presented in Lequy *et al.* (2009) for the MAAP. This formulation is based on the concept of a block, which represents the potential assignment of an employee to an activity or a task during consecutive time periods in a segment for this employee. A block is thus associated with a segment, a start time period, an end time period, and either an activity or a task. The duration of an activity block respects the activity duration bounds. The length of a task block is equal to the task duration. A task block is valid only if it satisfies the task time window. Finally, the employee assigned to the segment associated with a block must be qualified for the corresponding activity or task.

The proposed model relies on the following notation :

P : set of segments ;

T : set of time periods in the time horizon, numbered from 1 to $|T|$ in chronological order ;

A : set of activities ;

$d_{a,t}$: number of employees required for activity a in period t ;

$\underline{c}_a$: cost of one under-covering for activity a (per employee in shortage) ;

$B_{p,a}$: set of all activity blocks associated with segment p and activity a ;

J : set of tasks ;

ℓ_j : duration of task j ;

c_j : cost for not executing task j ;

$D_{p,j}$: set of all valid task blocks associated with segment p and task j ;

s_b : starting period of block b ;

e_b : ending period of block b ;

$s_{b,t}$: equal to 1 if block b starts at the beginning of period t, and 0 otherwise ;

$e_{b,t}$: equal to 1 if block b ends at the end of period t, and 0 otherwise ;

$\kappa_{b,t}$: equal to 1 if block b spans period t, and 0 otherwise ;

$\Gamma \subset J \times J$: set of pairs of tasks (j, j') with $j \neq j'$ subject to a precedence relationship stipulating that task j must be completed before starting task j' ;

$\bar{\Gamma} \subset \Gamma$: the subset of strong precedence relationships ;

c_τ : cost for one transition.

For each segment p, we denote by T_p, A_p and J_p the subset of time periods in segment p, the subset of activities, and the subset of tasks that can be assigned to segment p, respectively. The proposed model also involves the following variables :

$\mathbf{x}_b$: binary variable equal to 1 if activity block b is selected, and 0 otherwise ;

$\mathbf{y}_b$: binary variable equal to 1 if task block b is selected, and 0 otherwise ;

$\mathbf{z}_j$: binary variable equal to 1 if task j is unassigned, and 0 otherwise ;

$\mathbf{u}_{a,t}$: nonnegative variable indicating the number of under-coverings of activity a in time period t ;

Using this notation, the MATAP can be formulated as the following mixed-integer program (MIP), where M is a sufficiently large constant :

$$\underset{\mathbf{u},\mathbf{x},\mathbf{y},\mathbf{z}}{\text{Minimize}} \quad \sum_{j\in J} c_j \mathbf{z}_j + \sum_{a\in A}\sum_{t\in T} \underline{c}_a \mathbf{u}_{a,t} + c_\tau \Big(\sum_{p\in P}\sum_{a\in A}\sum_{b\in B_{p,a}} \mathbf{x}_b + \sum_{p\in P}\sum_{j\in J}\sum_{b\in D_{p,j}} \mathbf{y}_b - |P| \Big) \tag{7.1}$$

subject to :

$$\sum_{a\in A_p}\sum_{b\in B_{p,a}} \kappa_{b,t}\mathbf{x}_b + \sum_{j\in J}\sum_{b\in D_{p,j}} \kappa_{b,t}\mathbf{y}_b = 1, \qquad \forall p\in P,\, t\in T_p \tag{7.2}$$

$$\sum_{b\in B_{p,a}} e_{b,t-1}\mathbf{x}_b - \sum_{\substack{a'\in A_p \\ a\neq a'}}\sum_{b\in B_{p,a'}} s_{b,t}\mathbf{x}_b - \sum_{j\in J}\sum_{b\in D_{p,j}} s_{b,t}\mathbf{y}_b \le 0, \qquad \forall p\in P,\, t\in T_p,\, a\in A_p \tag{7.3}$$

$$\sum_{p\in P}\sum_{b\in B_{p,a}} \kappa_{b,t}\mathbf{x}_b + \mathbf{u}_{a,t} \ge d_{a,t}, \qquad \forall a\in A,\, t\in T \tag{7.4}$$

$$\sum_{p\in P}\sum_{b\in D_{p,j}} \mathbf{y}_b + \mathbf{z}_j = 1, \qquad \forall j\in J \tag{7.5}$$

$$(M+\ell_j)\mathbf{z}_{j'} + \sum_{p\in P}\sum_{b\in D_{p,j'}} s_b\mathbf{y}_b - M\mathbf{z}_j - \sum_{p\in P}\sum_{b\in D_{p,j}} s_b\mathbf{y}_b \ge \ell_j, \qquad \forall (j,j')\in\Gamma \tag{7.6}$$

$$\mathbf{z}_j - \mathbf{z}_{j'} \geq 0, \qquad \forall (j, j') \in \bar{\Gamma} \tag{7.7}$$

$$\mathbf{u}_{a,t} \geq 0, \qquad \forall a \in A,\, t \in T \tag{7.8}$$

$$\mathbf{x}_b \in \{0,1\}, \qquad \forall p \in P,\, a \in A,\, b \in B_{p,a} \tag{7.9}$$

$$\mathbf{y}_b \in \{0,1\}, \qquad \forall p \in P,\, j \in J,\, b \in D_{p,j} \tag{7.10}$$

$$\mathbf{z}_j \in \{0,1\}, \qquad \forall j \in J. \tag{7.11}$$

The objective function (7.1) minimizes the total cost yielded by the task and activity under-coverings and the transitions. The number of transitions is computed as the number of (activity or task) blocks minus the number of segments, since a transition occurs after every block, except the last block of a segment. Constraints (7.2) ensure that all segments are completely filled with activities or tasks. Inequalities (7.3) guarantee that two blocks of the same activity cannot be assigned contiguously in a segment (otherwise, the maximum activity duration might not be satisfied). Inequalities (7.4) are the activity demand constraints. They set the values of the under-covering variables. Similarly, equations (7.5) are the task covering constraints. Inequalities (7.6) enforce the task precedence relationships when both tasks j and j' are assigned (i.e., if $\mathbf{z}_j = \mathbf{z}_{j'} = 0$). These constraints also ensure that a successor task j' remains unassigned when its predecessor j is unassigned. Symmetrically, inequalities (7.7) forbid the assignment of a predecessor task j if its successor j' cannot be assigned for strong precedence relationships reasons. Finally, constraints (7.8)-(7.11) define the decision variable domains.

In practice, model (7.1)–(7.11) contains a very large number of activity block variables $\mathbf{x}_b$. Without tasks (that is, for the MAAP), only small one-day instances of this model can be solved to optimality in reasonable computational times using a commercial MIP solver as shown in Lequy *et al.* (2009). Clearly, when tasks are included one cannot expect better results. Therefore, we propose a two-stage heuristic for solving the MATAP. In the first stage, tasks are assigned to the employees at specific times considering the assignment of the activities but without taking into account the activity duration bounds. In the second stage, the activity assignment is performed considering fixed tasks. We also develop a second stage approach that allows some flexibility for the tasks.

4 First stage : Task assignment

The first stage of the proposed heuristic aims at assigning the tasks within the segments. To do so, we consider a simplified version of the MATAP, model it as a MIP and solve it using a commercial MIP solver. In this simplified MATAP, there are no activity duration constraints and no transition costs. In addition to the sets and coefficients introduced in the previous section, we define :

δ_p^{min} **:** minimum duration of any activity or task block that can be assigned to segment p ;

P^* **:** subset of segments for which $\delta_p^{min} \geq 2$.

If $p \in P^*$, then no activity or task block should start at the beginning of the i^{th} time period in p for $i = 2, 3, \ldots, \delta_p^{min} - 1$. Otherwise, no activity or task could be assigned in the first $i - 1$ periods, yielding infeasibility in the second stage. A similar remark applies for the end of the segment. Consequently, to avoid infeasibility, we remove from the sets $D_{p,j}$, $p \in P^*$, all task blocks that do not respect these conditions.

Furthermore, because we do not consider the activity duration bounds in the first stage, we replace the binary variables X_b for the activity blocks by continuous variables $\mathbf{x}_{p,a,t} \in [0, 1]$. If such a variable was binary, it would indicate whether or not activity a is assigned to segment p in period t.

With this additional notation, we propose the approximation model described below :

$$\underset{\mathbf{u},\mathbf{x},\mathbf{y},\mathbf{z}}{\text{Minimize}} \quad \sum_{j \in J} c_j \mathbf{z}_j + \sum_{a \in A} \sum_{t \in T} \underline{c}_a \mathbf{u}_{a,t} \tag{7.12}$$

subject to :

$$\sum_{a\in A_p} \mathbf{x}_{p,a,t} + \sum_{j\in J} \sum_{b\in D_{p,j}} \kappa_{b,t}\mathbf{y}_b = 1, \qquad \forall p \in P,\, t \in T_p \tag{7.13}$$

$$\sum_{p\in P} \mathbf{x}_{p,a,t} + \mathbf{u}_{a,t} \geq d_{a,t}, \qquad \forall a \in A,\, t \in T \tag{7.14}$$

$$\sum_{p\in P} \sum_{b\in D_{p,j}} \mathbf{y}_b + \mathbf{z}_j = 1, \qquad \forall j \in J \tag{7.15}$$

$$(M + \ell_j)\mathbf{z}_{j'} + \sum_{p\in P} \sum_{b\in D_{p,j'}} s_b\mathbf{y}_b - M\mathbf{z}_j - \sum_{p\in P} \sum_{b\in D_{p,j}} s_b\mathbf{y}_b \geq \ell_j, \qquad \forall (j,j') \in \Gamma \tag{7.16}$$

$$\mathbf{z}_j - \mathbf{z}_{j'} \geq 0, \qquad \forall (j,j') \in \bar{\Gamma} \tag{7.17}$$

$$\sum_{j\in J} \sum_{\substack{b\in D_{p,j} \\ e_b = t}} \mathbf{y}_b + \sum_{j\in J} \sum_{\substack{b\in D_{p,j} \\ s_b \in [t+2, t+\delta_p^{min}]}} \mathbf{y}_b \leq 1, \qquad \forall p \in P^*,\, t \in T_p \tag{7.18}$$

$$0 \leq \mathbf{x}_{p,a,t} \leq 1, \qquad \forall p \in P,\, a \in A_p,\, t \in T_p \tag{7.19}$$

$$\mathbf{u}_{a,t} \geq 0, \qquad \forall a \in A,\, t \in T \tag{7.20}$$

$$\mathbf{y}_b \in \{0,1\}, \qquad \forall p \in P,\, j \in J,\, b \in D_{p,j} \tag{7.21}$$

$$\mathbf{z}_j \in \{0,1\}, \qquad \forall j \in J. \tag{7.22}$$

Objective function (7.12) minimizes task and activity under-covering costs. Equations (7.13) ensure that all segments are completely assigned while inequalities (7.14) compute under-covering variables as the excess number of employees. Equations (7.15) state that every task is either performed or considered in the objective function. Inequalities (7.16) and (7.17) enforce the precedence constraints. Inequalities (7.18) guarantee that, if two tasks are assigned to the same segment, either they are consecutive or there is enough time between them to assign at least the smallest activity or task block in between. Constraints (7.19)–(7.22) define the domains of decision variables.

5 Second stage : Activity assignment

In this section we present how to assign activities to employees taking into account the task blocks selected in the first-stage solution. To do so, we propose four different strategies that are adapted versions of the rolling horizon/column generation

heuristic proposed by Lequy *et al.* (2009) for the MAAP. We begin by summarizing this heuristic method.

5.1 Activity assignment without tasks

Lequy *et al.* (2009) developed a column generation method for solving the MAAP, that is, for assigning multiple activities without considering tasks. Their formulation of the MAAP relies on the following definition. Given a segment $p \in P$, a feasible activity sequence s for p is a sequence of non-overlapping activity blocks that covers the whole segment and respects the activity duration bounds. It uses the notation previously defined and the following one :

S_p : set of feasible activity sequences for segment p ;

$c_{p,s}$: cost of sequence $s \in S_p$ (equal to the sum of the transition costs) ;

$\kappa_{a,t}^{p,s}$: equal to 1 if sequence $s \in S_p$ contains an assignment of activity $a \in A$ in time period $t \in T_p$, and 0 otherwise ;

$\theta_{p,s}$: binary variable equal to 1 if sequence $s \in S_p$ is selected for segment p, and 0 otherwise

Formally, the MAAP mixed integer programming (MIP) model is :

$$\underset{\mathbf{u},\theta}{\text{Minimize}} \quad \sum_{a \in A} \sum_{t \in T} \underline{c}_a U_{a,t} + \sum_{p \in P} \sum_{s \in S_p} c_{p,s} \theta_{p,s} \tag{7.23}$$

subject to :

$$\sum_{s \in S_p} \theta_{p,s} = 1, \qquad \forall p \in P \tag{7.24}$$

$$\sum_{p \in P} \sum_{s \in S_p} \kappa_{a,t}^{p,s} \theta_{p,s} + U_{a,t} \geq d_{a,t}, \qquad \forall a \in A, t \in T \tag{7.25}$$

$$\theta_{p,s} \in \{0,1\}, \qquad \forall p \in P, s \in S_p \tag{7.26}$$

$$U_{a,t} \geq 0, \qquad \forall a \in A, t \in T. \tag{7.27}$$

Objective function (7.23) aims at minimizing the sum of the undercovering and the transition costs. Equalities (7.24) ensure that a feasible sequence is selected for each segment, whereas inequalities (7.25) are the activity demand constraints.

The linear relaxation of model (7.23)–(7.27) is called the master problem and solved using a column generation method. Such a method solves at each iteration a restricted master problem (RMP) and several subproblems. The RMP is simply the master problem restricted to a subset of its variables $\theta_{p,s}$. There is one subproblem per segment that aims at finding, for this segment p, the variable $\theta_{p,s}$ with the least reduced cost with respect to the current RMP dual solution. If no negative reduced cost sequences can be found for all subproblems, the column generation process stops and the current RMP primal solution is optimal for the master problem. Otherwise, negative reduced cost variables (columns) are added to the RMP before starting a new iteration.

For the MAAP, each subproblem is a shortest path problem defined on an acyclic network. Figure 7.1 illustrates such a network for a segment p such that T_p contains 8 time periods (numbered from 1 to 8) and A_p three activities (a^1, a^2, and a^3). The minimum and maximum activity durations are 4 and 8 periods for a^1, 2 and 3 periods for a^2, and 2 and 4 periods for a^3. In this figure, nodes α_p and β_p represent the start and the end time of segment p, respectively. The square and round nodes, except α_p and β_p, correspond to the start and the end of an activity block at a given period, respectively. There is one arc for each activity block in $\bigcup_{a \in A_p} B_{p,a}$ that links a start of activity node to an end of activity node. Additionally, there are start of sequence, end of sequence, and transition arcs. The latter link an end of activity node to a start of activity node associated with the same time, but not the same activity. A path from node α_p to node β_p in such a network represents a feasible activity sequence for segment p. Arc costs involve the RMP dual variables to ensure that the cost of a path is equal to the reduced cost of its corresponding sequence variable. See Lequy *et al.* (2009) for additional details.

Integer solutions to model (7.23)–(7.27) are obtained using two heuristics. First, integrality requirements are imposed on the variables in the last RMP and the resulting mixed integer program is solved directly with a MIP solver (no additional variables are generated). If this heuristic fails to find a solution within a certain time limit or if the quality of the computed solution is not good enough, then a second heuristic is called. This heuristic is a variable fixing procedure that fixes at each iteration one variable $\theta_{p,s}$ to 1, modify the master problem in consequence, and reoptimizes it using column generation.

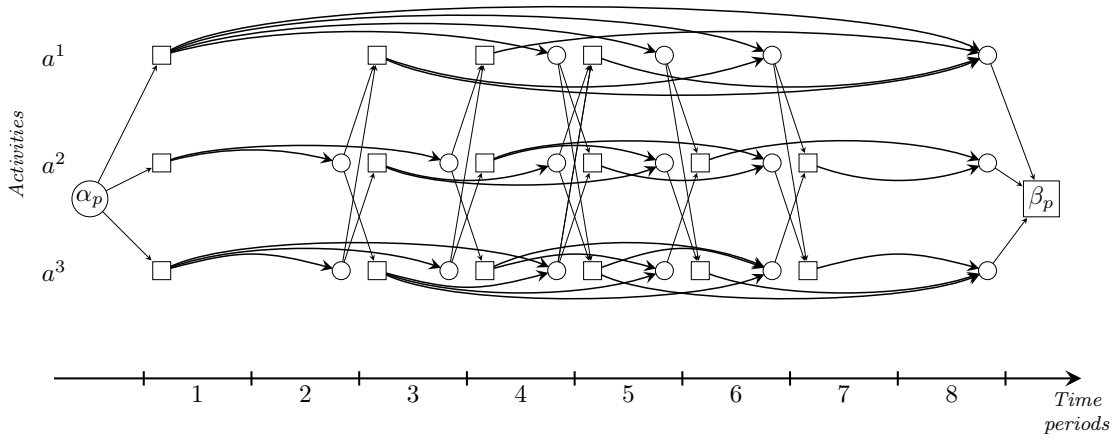

FIGURE 7.1 – Example of a network for activity assignment without tasks

To speed up the solution process, Lequy *et al.* (2009) embedded this column generation heuristic into a rolling horizon procedure. Such a procedure divides the planning horizon into overlapping time slices. Starting from the first slice, it solves sequentially a MAAP restricted to each time slice using the column generation heuristic. To ensure continuity in the overall solution, initial conditions derived from the solution of the previous time slice are imposed in each time slice except the first.

5.2 Activity assignment with fixed tasks

When task blocks are selected in the first stage of the heuristic, activity assignment is performed in the second stage while considering these blocks. A first strategy consists of fixing these task blocks, that is, assigning each task $j \in J$ covered in the first-stage solution to segment p_j starting at time t_j, where p_j and t_j are provided by this solution. The resulting multi-activity assignment problem with fixed tasks can then be solved using a modified version of the rolling horizon/column generation heuristic of Lequy *et al.* (2009).

The modifications concern only the column generation algorithm. The integer master problem (7.23)–(7.27) remains the same except that S_p, $p \in P$, denotes now the set of feasible activity and task sequences for segment p. These activity and task sequences are generated by the corresponding subproblem defined on a modified network. Let $J_p = \{j \in J | p_j = p\}$ be the (possibly empty) set of task blocks assigned

to segment p and let $I_p = \bigcup_{j \in J_p} \{t_j, t_j + 1, \ldots, t_j + \ell_j - 1\}$ be the subset of T_p where a task is assigned. The modified network contains a task block arc for each task $j \in J_p$. Such an arc links a start of task to an end of task node that play roles similar to those of their counterpart activity nodes. Furthermore, all activity block arcs that spans a period in I_p are removed from the network. Figure 7.2 presents such a modified network for an example that involves a single task j^1 whose task block spans time periods 5 and 6. Except for the addition of task j^1, this example is the same as the one used for Figure 7.1. It clearly shows that the treatment of fixed tasks can reduce the subproblem complexity.

Note that every path in a modified network for segment p includes all task block arcs associated with the tasks in J_p. Hence, there is no need for additional task covering constraints in the integer master problem (7.23)–(7.27) because constraints (7.24) ensure that one sequence is selected for each segment. This modeling option assumes that activity assignment is feasible when considering fixed tasks. In practice, this assumption often holds because the activity duration bounds usually offer enough leeway around the fixed tasks to schedule activities. Furthermore, constraints (7.18) and the definition of the sets $D_{p,j}$ ensure that there is enough time to assign an activity or a task before and after every task if needed. When this assumption is not satisfied, subproblem networks including all activity block arcs as well as task covering constraints with slack variables Z_j in the integer master problem must be considered. In the rest of this paper, we assume that activity assignment is always feasible when forcing the covering of all tasks selected in the first-stage solution.

5.3 Activity assignment with flexibility on tasks

The task assignment model (7.13)–(7.22) might lead to poor decisions concerning activities due to limited information. To correct this, task assignment may be slightly modified during the activity assignment phase. In this article, flexibility is defined as the reassignment of a task j among a very limited subset of its task blocks, denoted by $\hat{D}_j$. Those task blocks, together with activity blocks, are thus part of activity and task sequences in model (7.23)–(7.27). However this model must be modified to ensure that one and only one task block is assigned while fulfilling all constraints on tasks. Time window and qualifications constraints are already satisfied by the definition

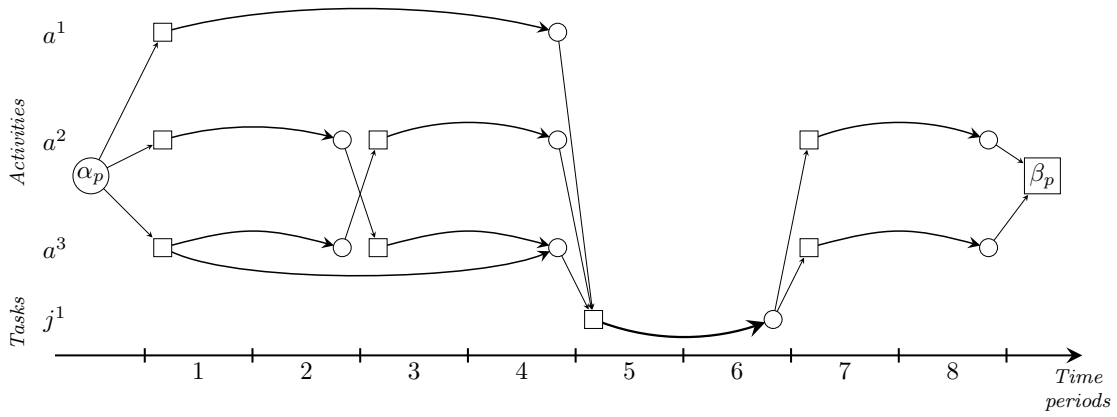

FIGURE 7.2 – Example of a network for activity assignment with fixed tasks

of the task blocks. When defining the sets $\hat{D}_j$, we just have to make certain that precedence relationships are satisfied. We introduce three types of flexibility.

The first type of flexibility consists simply in allowing the assignment of a task j to a different employee at the very same starting time t_j as the previously assigned task block. Subset $\hat{D}_j$ comprises all the valid task blocks (for different segments) with starting time t_j. Model (7.23)–(7.27) has to be modified to include the following constraint set :

$$\sum_{p \in P} \sum_{s \in S_p} \kappa_j^{p,s} \theta_{p,s} = 1, \quad \forall j \in J, \tag{7.28}$$

where $\kappa_j^{p,s}$ are binary parameters equal to 1 if sequence s for segment p contains an assignment to task $j \in J$ and 0 otherwise. Constraint (7.28) for task j stipulates that exactly one task block in $\hat{D}_j$ must be part of a sequence in the final solution.

The subproblem networks must also be changed to include the various task blocks. Each task block in $\hat{D}_j$ for all $j \in J$ is represented by an arc in the corresponding network as in Figure 7.2. However, as opposed to the fixed tasks case described in Section 5.2, all activity block arcs must be considered in all networks, even those overlapping with a task block, because every task block might not be part of the final solution.

A second type of flexibility allows to assign a task j slightly earlier or later than t_j, the start time of the previously assigned task block, but still to the same segment

p_j. Thus, the subset $\hat{D}_j$ contains only valid task blocks associated with segment p_j. They start in a contiguous subset of T called the flexibility window which is denoted by F_j. The flexibility windows of all tasks have to be defined in such a way that the final task assignment fulfills all the precedence relationships. To avoid paths that contain the same task twice or more and to preserve the shortest path structure of the subproblems, we decided to define the flexibility windows F_j shorter than the length ℓ_j of the associated task j. Such a window is given by $F_j = \left\{f_j^s, f_j^s + 1, \ldots, f_j^e\right\}$, where

$$f_j^s = \max\left\{\min_{t\in W_j} t; \max_{j'\in J|(j',j)\in\Gamma}\left\lfloor\frac{t_{j'}+\ell_{j'}-1+t_j}{2}\right\rfloor; t_j - \left\lfloor\frac{\ell_j}{2}\right\rfloor\right\} \quad (7.29)$$

$$f_j^e = \min\left\{\max_{t\in W_j} t; \min_{j'\in J|(j,j')\in\Gamma}\left\lfloor\frac{t_j+\ell_j-1+t_{j'}}{2}\right\rfloor - \ell_j; t_j + \left\lfloor\frac{\ell_j-1}{2}\right\rfloor\right\}. \quad (7.30)$$

In these formulas, W_j is the set of all feasible start time periods for task j according to its time window. The second term in the right-hand side of (7.29) restricts the flexibility for task j to about one half of the slack time between t_j, the proposed start of task j, and $t_{j'}+\ell_{j'}$, the proposed end time of any predecessor task j' of j. Similarly, the second term in the right-hand side of (7.30) restricts the flexibility with respect to successor tasks j' of task j. Finally, the last terms in the right-hand sides of (7.29) and (7.30) center the flexibility window around t_j and ensure that its length is shorter than ℓ_j.

To deal with this flexibility, constraints (7.28) are added to the integer master problem (7.23)–(7.27) to ensure that each task is assigned once. The subproblem networks are also modified. The network of subproblem for segment p includes an arc for each task block in $D_{p,j}$ associated with a task $j \in J$ such that $p = p_j$ and its start time period is in F_j. Furthermore, this network contains all arcs for the activity blocks in $B_{p,a}$ for each activity $a \in A_p$. Such a network is illustrated in Figure 7.3 for an exemple that involves the three activities of the previous examples and two tasks j^1 and j^2 with length $\ell_{j^1} = 4$ and $\ell_{j^2} = 4$. The flexibility windows for these tasks are $F_{j^1} = \{1, 2, 3\}$ and $F_{j^2} = \{4, 5\}$. In this example, there no task j^2 block arc starting in period 2 because such a block is not valid (all minimum activity durations exceed 2).

The last type of flexibility is the combination of both previous flexibilities. That means that we consider all the valid task blocks for task j starting in its flexibility

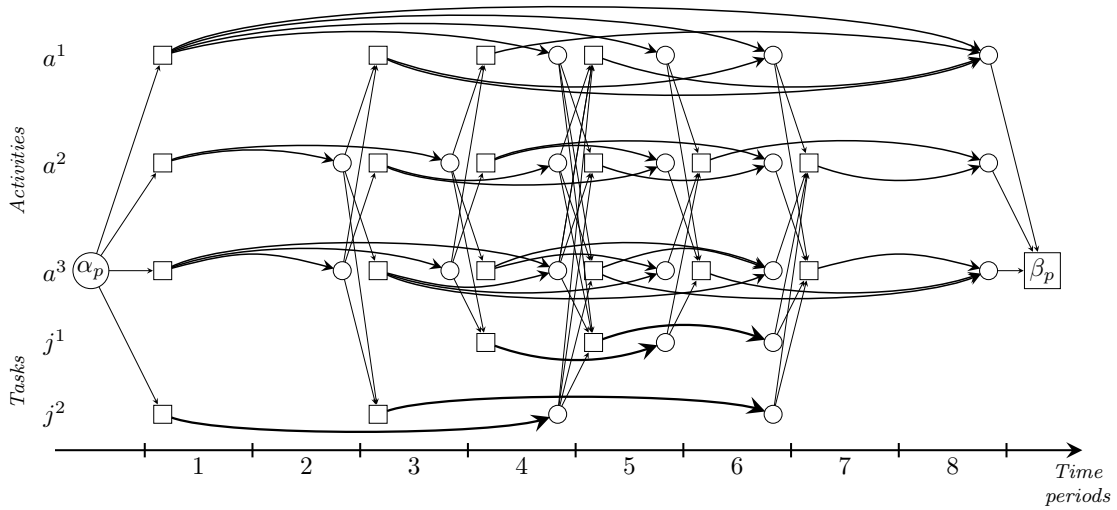

FIGURE 7.3 – Example of a network for activity assignment with time flexibility

window as part of $\hat{D}_j$, even if they are not associated with segment p_j. In this case, the integer master problem is given by (7.23)–(7.28). The networks are identical to those described for the time flexibility except that task j block arcs are considered in a network for segment p even if $p \neq p_j$.

6 Acceleration strategies

In this section, we present acceleration strategies for both task assignment and activity assignment stages of the proposed heuristic.

6.1 Task assignment

Preliminary computational tests showed that the task assignment model (7.13)–(7.22) might yield very large branch-and-bound trees because of relatively large integrality gaps induced by the precedence constraints. To speed up the approximation model solution process, we propose an alteration and a heuristic for this model. Let $D = \bigcup_{p \in P, j \in J} D_{p,j}$. The costs on the variables $\mathbf{y}_b, b \in D$, are set to νs_b with the aim of encouraging the selection of the earliest task blocks. The value of parameter ν depends on the length of the time horizon, the number of tasks and the minimum

under-covering cost, in order to keep those new costs low enough in comparison to regular costs. We chose the value $\nu = \frac{\min_{a \in A} \underline{c}_a}{|J| \max_{t \in T} t}$.

This model is solved with a dedicated branching heuristic which is more efficient than the default branching method of our commercial solver, and allows us to reduce the computation time. The method is an iterative heuristic with two steps. First, we solve with a barrier-type method the linear relaxation of model (7.12)–(7.22) also considering the costs on $\mathbf{y}_b, b \in D$. We then adjust all costs of $\mathbf{y}_b, b \in D$, by decreasing them by a small amount if the value of $\mathbf{y}_b$ is fractional. If one variable value doesn't rise from an iteration to another, after having put a bonus on it, we reset all the costs of the variables of the same task to their initial costs (νs_b, in this case). Either after a certain number of iterations or if the number of fractional-valued variables drops below a certain threshold (depending on the number of tasks), we stop this process. We then begin a normal branching process by keeping the costs like they were at the last iteration of the process.

This heuristic method allows to rapidly reduce the number of fractional-valued variables and to define a (softly-restricted) problem with modified costs that is relatively easy to solve. With this technique, the size of the branch-and-bound tree remains under control and does not really depend on the integrality gap of the original model, but rather on that of the softly-restricted problem wich is often relatively small.

6.2 Activity assignment

Due to the minimization of the number of transitions, the activity and task sequences appearing in the RMP solutions during the column gneration process may contain a small number of activity blocks. If we order the rows in the appropriate order (see Lequy *et al.* (2009)), the constraint matrix will contain many columns with long sequences of consecutive ones. A density reduction of the constraint matrix can then be performed on the activity assignment models by subtracting a constraint with the preceding one. For this, we also have to introduce over-covering variables $O_{a,t}$, for each activity a and for each time period t, and set their costs to zero.

We thus replace constraint set (7.25) in the model (7.23)–(7.27) by the following

constraint sets :

$$\sum_{p\in P}\sum_{s\in S_p}\left(\kappa_{a,t}^{p,s}-\kappa_{a,t-1}^{p,s}\right)\theta_{p,s}-O_{a,t}+U_{a,t}+$$

$$O_{a,t-1}-U_{a,t-1}=d_{a,t}-d_{a,t-1}, \qquad \forall a\in A, t\in T \quad (7.31)$$

$$O_{a,t}\geq 0, \qquad \forall a\in A, t\in T, \quad (7.32)$$

where we assume that when $t=1$ in (7.31), $\kappa_{a,t-1}^{p,s}=d_{a,t-1}=0$ and there are no variables $O_{a,t-1}$ and $U_{a,t-1}$. The impact of this transformation on the column generation subproblems is discussed in Lequy *et al.* (2009).

The subproblem networks of Section 5.3 can also be modified to speed up the column generation process. For each period, the number of transition arcs in a network grows quadratically with respect to a weighted sum of $|A_p|$ and $|J_p|$ for a segment p, where in this case J_p contains all tasks that can be assigned to p. When the number of simultaneous starting or ending task blocks becomes large, the number of transition arcs in the network may become too high to keep the computation times low for the shortest path algorithm. Fortunately, the number of arcs in the subproblems can be reduced by adding some extra nodes in the networks.

Consider the network for a segment p. Given an arbitrary order of the activities, let $n_p=|A_p|$ and $a_1,a_2,\ldots,a_{n_p}$ be the activities of A_p. For each period t of segment p except the last one, two extra types of nodes are defined for each activity $a\in A_p$, denoted by $Q_{a_i,t},\ 1\leq i\leq n_p-1$, and $R_{a_i,t},\ 2\leq i\leq n_p,$, respectively, as well as two other extra nodes, denoted by Q_t and R_t. These nodes help create two chains, one going up and one going down through the ordered activities. Figure 7.4 illustrates such a reduction for the transitions between the end of a period t to the beginning of period $t+1$. Its left part shows the original subnetwork where the nodes $K_{a,t}$ and $G_{a,t+1}$ (resp. $L_{j,t}$ and $H_{j,t+1}$) represent the end of an activity a (resp. task j) block at period t and the start of an activity a (resp. task j) block at period $t+1$, respectively. The right part of the figure presents the subnetwork after the reduction, including the extra Q and R nodes.

The arcs of the reduced subnetwork are as follows. On one hand, an arc links node Q_t to node $Q_{a_{n_p-1},t}$ and for all integers $i\in[2,n_p-1]$, an arc links node $Q_{a_i,t}$ to node $Q_{a_{i-1},t}$. Node $Q_{a_i,t}$ is also linked to node $G_{a_i,t+1}$. On the other hand, for all integers

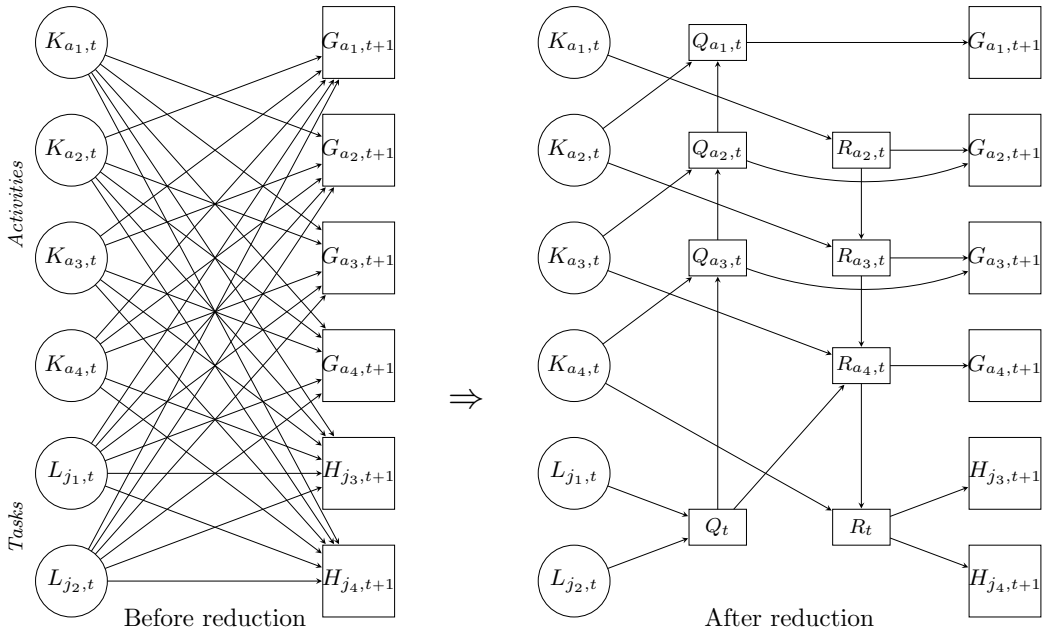

FIGURE 7.4 – Arc reduction in a piece subnetwork with four activities and four tasks

$i \in [2, n_p - 1]$, $R_{a_i,t}$ is linked to $R_{a_{i+1},t}$ and to $G_{a_i,t+1}$ by two different arcs. An arc links node $R_{a_{n_p},t}$ to R_t. For $1 \leq i \leq n_p$, each node $K_{a_i,t}$ is linked if possible to $Q_{a_{i-1},t}$ and to $R_{a_{i+1},t}$. All nodes $L_{j,t}$ are linked to node Q_t and node R_t is linked to all nodes $H_{j,t+1}$. Finally, an arc links node $K_{a_{n_p},t}$ to node R_t and an arc links node Q_t to node $R_{a_{n_p},t}$.

By adding $2|A_p|$ nodes, the growth in the number of arcs becomes linear with respect to a weighted sum of $|A_p|$ and $|J_p|$ instead of quadratic. This reduction for each period positively impacts the shortest path resolution since the efficiency of shortest path algorithms is strongly related to the number of arcs in acyclic networks.

7 Computational experiments

To compare the different methods, we performed computation experiments on randomly generated problems. The instances try to mimic shifts utilized in targeted companies, such as retail store. We also tried to duplicate the characteristics of activities and tasks, in a way to adhere as close as possible to real-world instances. In

the generating process, all parameters were fixed, except for three of them related to the number of tasks, their average length, and the number of precedence relationships between tasks. These parameters are used to study the efficiency of our different strategies.

We chose 8-hour shifts containing only one 30-minute break in the middle. We then discretized a one-week time horizon in 15 minute increments. During this horizon, each employee has a workload of 35 hours. Shifts are centered around noon following a normal distribution except for 20% of them that are night shifts. We considered five activities. The minimal duration for each activity is about one hour and the maximal duration is about three hours. The under-covering cost $\underline{c}_a$ was set at 1500 for all activities $a \in A$ per minute per employee and the transition cost c_τ at 15. Task costs were taken to equal the time length of the task multiplied by 12000. They are arbitrary high because in our context task under-coverings are more expensive than activity under-coverings. We chose all precedence relationships to be strong.

All tests were performed on an *Intel®Core™2 CPU 6700* clocked at 2.66GHz with 4GB RAM. We used XPRESS solver of *Fair Isaac Corporation* for solving linear and integer programs. The default parameters were used except for the branch-and-bound stopping criterion. The absolute tolerance determining whether the global search will continue or not was set at 1500, the cost of an under-covering.

7.1 Results on basic benchmarking instances

We first performed benchmark tests to examine the efficiency of our methods over five one-day instances. These instances have fewer employees than the regular test instances and their main characteristics are described in Table 7.1. The first column identifies the particular problem, the second reports the number of tasks, the third shows the mean task length, the fourth provides the number of precedence relationships, and the fifth gives the number of employees. Each row describes the properties of the specific problem instance.

The results obtained from running our four strategies on the five instances are reported in Table 7.2. The first column shows the method we applied. The following five columns provide the results for each benchmark instance, giving both the computed solution value and the computation time in seconds. We present, in row order,

Instance Id	$\|J\|$	ℓ	$\|\Gamma\|$	Employees
bench_1	5	96	3	20
bench_2	5	99	2	50
bench_3	15	89	6	50
bench_4	15	101	8	50
bench_5	25	99	13	50

TABLEAU 7.1 – Properties of benchmark problems

the results for the linear relaxation of the (global) model (7.1)–(7.11) and for this model when solved by an exact branch-and-bound algorithm. We also give the results of the rolling horizon heuristic without flexibility (Fixed Tasks), with the time only strategy, with the employee only strategy, and finally with the full flexibility strategy.

	bench_1		bench_2		bench_3		bench_4		bench_5	
Method	Value	Time	Value	Time	Value	Time	Value	Time	Value	Time
Global Model Relaxation	5175	1.1	5275	3.8	6648	4.0	16956	2.5	3396	19
Global Model	5175	2.4	5280	71	7215	82	17850	55	-	-
Fixed Tasks	9600	< 1	5415	8.8	10335	7.3	26985	4.7	22470	4.7
Time Only	9600	1.5	5400	8.5	8805	8.1	23970	4.6	10350	13
Employee Only	8115	< 1	5340	7.0	8790	6.1	26940	4.9	10320	11
Full Flexibility	5175	1.0	5280	10	7260	13	22410	50	7215	23

TABLEAU 7.2 – Results for the benchmark problems

Interestingly enough, the heuristics were quite effective on the benchmark instances. This was especially true for the Full Flexibility strategy, even if extra undercoverings may have appeared in the solutions for the bench_4 and bench_5 instances. We also remark that as the integrality gap grows larger, the performance of the heuristics deteriorates. For the last instance, our commercial solver failed to find the optimal solution within a day of computation time and the best objective value it gave was 4185. The Fixed Tasks strategy was always the worst because the lack of flexibility implies a search set that is smaller than all other methods. The Time Only and Employee Only strategies were in general less effective than the Full Flexibility strategy because their feasiblity sets were also smaller than that of the Full Flexibility approach.

The results for the first-stage approximation model (7.12)–(7.22) are presented in Table 7.3 where the first column gives the instance identifier, the second reports the value of the linear relaxation solution, the third shows the value of the computed integer solution, the fourth provides the number of assigned tasks in this integer solution and the last column presents the computational time in seconds.

	LP	Solution	Assigned	
Id	**Bound**	**Value**	**Tasks**	**Time**
bench_1	0	0	5	< 1
bench_2	0	0	5	< 1
bench_3	0	0	15	1.1
bench_4	12000	12000	15	1.3
bench_5	1200	3000	25	3.9

TABLEAU 7.3 – Results of the approximation model on the benchmark instances

The table shows that all tasks are assigned in all instances but unavoidable under-coverings for activities exist in the fourth and fifth instances. The first three instances don't possess such unavoidable under-coverings and this explains why our heuristics were able to get high quality results. With respect to computational times, all problems were solved very efficiently.

Another important point to be taken away from this table is that obtaining a feasible solution when most tasks are assigned becomes very hard when the number of tasks increases. The more tasks present, the more precedence relationships need to be handled. This implies a larger integrality gap in the approximation model and a larger search tree created by the increased number of binary variables.

7.2 Results on the test instances

We performed scenario analyses in order to analyze the behavior of our algorithms when certain parameters vary. In each of following tables only one parameter was allowed to take several values while the others were fixed. Ten instances were then created for each sample value and solved by all our methods. The results reported in the following three tables are mean values over those ten instances. Columns in these tables below the headings have the following meaning. The first column is the problem

identifier, columns two to four report the number of tasks, the mean task length, and the number of precedence relationships, respectively. Column five shows the number of assigned tasks (A.T.) in the solution of the first-stage model and column six gives the computational time in seconds for this first stage. Then for each strategy, the first column provides the computed solution value and the second the computational time required by the second-stage rolling horizon/column generation heuristic.

First we considered instances where we varied the mean task length. We picked values centered around 60, 90, 120 and 240 minutes, respectively. The results are reported in Table 7.4. As the table highlights, the approximation model seems to become faster as the mean length elongates. The Full Flexibility strategy proved to always be the best at the expected expense of being somewhat slower than the other approaches. The Employee Only strategy worked better than the Time Only strategy. The computational times were in the same range for all strategies and most of these times (more than 90%) is spent in the first stage of the heuristic. Furthermore, we observe that the computed solution value increases with the average task length as longer tasks take more of the employee available time yielding a higher number of activity under-coverings.

Id	**Properties**			**App.**		**Fixed Tasks**		**Employee Only**		**Time Only**		**Full Flexibility**	
	$\|J\|$	ℓ	$\|\Gamma\|$	A.T.	Time	Value	Time	Value	Time	Value	Time	Value	Time
1	50	68	25	50	531	73378	25	62343	31	66304	28	54792	35
2	50	95	24	49	522	77886	23	70828	31	71119	28	64729	34
3	50	127	25	49	509	93144	24	84784	31	91360	29	78739	36
4	50	226	24	49	440	145513	19	145372	27	144667	22	139542	28

TABLEAU 7.4 – Results for various task lengths

Next, we considered instances with various numbers of precedence relationships. We designed three type of instances : no precedence, a normal density and a high density. The results are reported in Table 7.5. The heuristic with full flexibility was once again the most effective and the precedence relationships didn't seem to affect its computational time. This is due to the fact that the first-stage heuristic is not influenced by these relationships (see Section 6.1) and that these relationships can only reduce the length of the flexibility windows in the second stage. The addition of precedence constraints doesn't seem to have a very negative impact on the number

of under-coverings.

Id	Properties			App.		Fixed Tasks		Employee Only		Time Only		Full Flexibility	
	$\|J\|$	ℓ	$\|\Gamma\|$	A.T.	Time	Value	Time	Value	Time	Value	Time	Value	Time
5	50	95	0	49	524	77922	24	69258	33	71877	30	60876	37
6	50	95	24	49	528	76926	23	66594	31	69262	26	60355	38
7	50	95	98	49	520	87156	22	76978	30	78876	27	67558	33

TABLEAU 7.5 – Results for different density precedence relationships

Finally, we considered the number of tasks which is the most important parameter because it directly impacts the size of the first-stage model and, for certain strategies, the size of the subproblem networks in the second stage. The results are reported in Table 7.6. Obviously, the higher the number of tasks, the longer the computational time, especially for the approximation model solution. Not surprisingly, the Full Flexibility strategy always got the best results, but was more affected by the number of tasks in terms of computational time. The opposite occurred for the Fixed Tasks strategy because each fixed task simplifies a subproblem whereas one task with flexibility increases the size of the network. As expected, assigning more tasks increases the number of activity under-coverings.

Id	Properties			App.		Fixed Tasks		Employee Only		Time Only		Full Flexibility	
	$\|J\|$	ℓ	$\|\Gamma\|$	A.T.	Time	Value	Time	Value	Time	Value	Time	Value	Time
11	20	93	10	20	474	42066	26	36175	30	39060	29	32203	32
12	50	95	25	49	522	77431	22	69036	30	69645	28	59770	36
13	70	93	35	70	555	113925	21	103393	32	101971	28	88525	44
14	100	93	50	99	590	167704	20	148011	39	142818	29	118879	52
15	150	94	71	149	637	254356	14	225987	43	214812	30	176655	77
16	200	94	99	199	675	333229	14	301204	52	279594	30	226231	116

TABLEAU 7.6 – Results for different numbers of tasks

7.3 Performance profiles

In order to compare all our strategies, we use performance profiles originally introduced in Dolan et Moré (2002). First, we briefly recall here how to read them.

Suppose that a given algorithm $\mathcal{A}_i$ from a competing set $\mathcal{A}$ reports a statistic $u_{i,j} \geq 0$ when run on problem j from a test set $\mathcal{S}$, and that the smaller this statistic the better the algorithm is considered. Let the function

$$\omega\left(u, u^*, \rho\right) = \begin{cases} 1 & \text{if } u \leq \rho u^* \\ 0 & \text{otherwise} \end{cases}$$

be defined for any scalars $u \geq 0$, $u^* \geq 0$ and $\rho \geq 1$. The performance profile of algorithm $\mathcal{A}_i$ is the function

$$\pi_i(\rho) = \frac{\sum_{j \in \mathcal{S}} \omega\left(u_{i,j}, u_j^*, \rho\right)}{|\mathcal{S}|}$$

where $u_j^* = \min_{i \in \mathcal{A}} u_{i,j}$. Thus, $\pi_i(1)$ gives the fraction of the number of problems for which algorithm $\mathcal{A}_i$ was the most effective, according to the statistics $u_{i,j}$, $\pi_i(2)$ gives the fraction for which algorithm $\mathcal{A}_i$ is within a factor of 2 of the best, and $\lim_{\rho \to +\infty} \pi_i(\rho)$ gives the fraction of examples for which the algorithm succeeded.

We computed performance profiles for our four strategies with the computed solution value as the targeted statistic and all test problems of the previous subsection as the test set. Profiles are displayed in Figure 7.5. Given that all our final solutions are feasible, all algorithms' profiles converge to 1. The closer to 1 a curve is, the better is the associated method. We cleary see that the proposed two-stage heuristic with the Full Flexibility strategy outclassed all the other methods. As expected, the heuristic with Fixed Tasks strategy was outperformed by the other three algorithm variants.

8 Conclusions

This paper has introduced the multi-activity and task assignment problem with skills, task time windows and precedence relationships. We proposed a mixed integer programming formulation and a two-stage heuristic involving four different strategies to tackle large-scale instances. The strategies provide certain forms of flexibility in the final task assignment after the tasks were preassigned without activities using an approximation model. We performed extensive computational tests on randomly

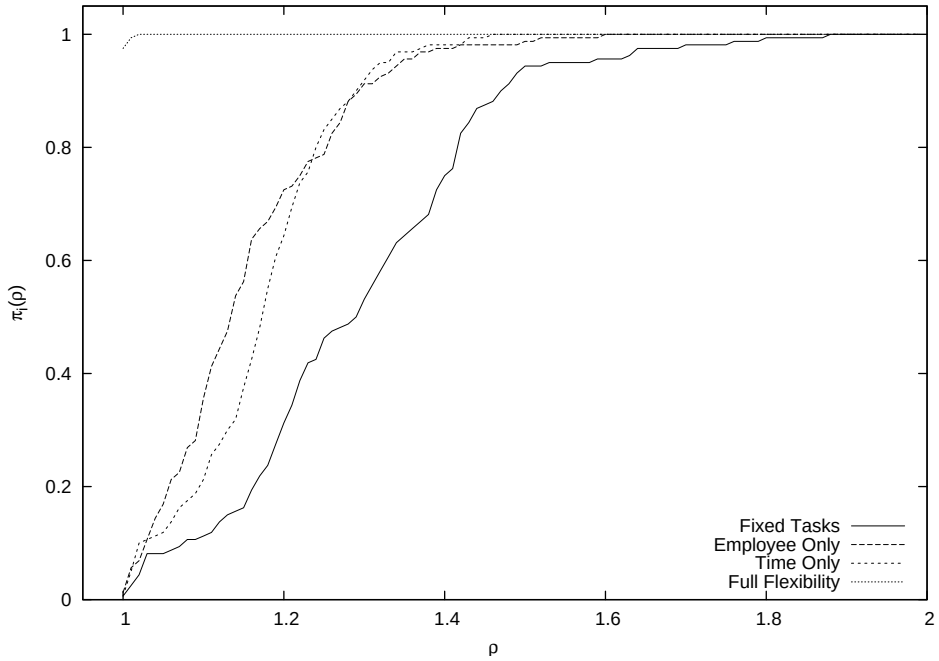

FIGURE 7.5 – Profile curves comparing the solution values obtained for the various strategies

generated instances in order to validate our heuristic method and to compare the various strategies. The strategy allowing the most flexibility proved universally best when compared to the other three policies at the expense of being slightly less efficient. It obtained near optimal solutions as seen from comparing its results to a model's lower bound or optimal values. The heuristic was able to obtain high quality solutions using small computational times for problems with 50 employees, 5 activities and up to 200 tasks over a one-week horizon.

While our results were obtained using randomly generated data, we believe they are general in nature. We conjecture that our conclusions will remain valid for any real-world application even though the scenario's complexity will somewhat influence the heuristic effectiveness and efficiency.

Acknowledgments : This research was supported by a collaborative R&D grant offered by Kronos and the Natural Sciences and Engineering Research Council of Canada. The authors wish to thank the personnel of Kronos for their collaboration.

ARTICLE 3 : ASSIGNING TEAM TASKS AND MULTIPLE ACTIVITIES TO FIXED WORK SHIFTS

Article soumis à *Journal of Heuristics* (2010) et écrit par :

QUENTIN LEQUY

École Polytechnique de Montréal and GERAD

GUY DESAULNIERS

École Polytechnique de Montréal and GERAD

MARIUS M. SOLOMON

Northeastern University and GERAD

Abstract

The multi-activity and task assignment problem consists of assigning interruptible activities and uninterruptible tasks to given work shifts so as to match as close as possible a demand functions of time. In this paper, we consider an extension of this problem, called the multi-activity and team task assignment problem, which takes into account uninterruptible pieces of work done by several employees simultaneously whereas tasks were previously done only by single employees. These tasks of a new kind are called team tasks and the relation between the number of employees contributing to them and the time they spend are called team patterns. We propose then a enumerative mixed integer programming formulation and a two-module approach to find solutions to this problem. The first module is an approximation model for assigning tasks without activities, which is solved thanks a Variable Neighbourhood Descent. The second module consists of assigning activities by allowing tasks reassignments during the process. Computational experiments have been performed to validate our approach.

Keywords : Activity assignment, work shifts, team tasks, precedence relationships, column generation.

1 Introduction

In this paper, we present an extension to the multi-activity and task assignment problem (MATAP) with fixed work shifts described in Lequy *et al.* (2010). This kind of problem comes from planning scheduling problem that arises in companies such as supermarkets or leisure resorts. In this problem, the work of employees is viewed either as activities or as tasks. An activity can be interrupted at any time by quickly replacing an employee by another one whereas a task must be done straight forward. Tasks must fulfill several constraints like due-date completion or precedence relationships and represent only a small part of the whole workload. However, not covering them yields a high penality cost. Activities represent daily work and especially the service offered by the companies to their customers. Each of them has to meet as close as possible a varying demand functions of time, otherwise a lack of employees will diminish substantially quality of the service.

The MATAP can deal only with tasks performed by only one employee. We introduce in this problem tasks which can be performed by several employees simultaneously. They come either from single tasks which can be divided to speed up their completion, or from some types of work which requires more than one employee to be performed, such as carrying some heavy load. In both cases, the number of employees working on them at the same time is related to the time they have to spend to perform the tasks. Those potential relationships compose their team patterns. This kind of tasks are called team tasks and adding team tasks in the MATAP forms the multi-activity and team task assignment problem (MATTAP). We consider that team tasks which have to be perform by more than one employee do not compose the majority of the tasks.

To the best of our knowledge, this problem has never been addressed in the literature. However solving scheduling problem represents a wide part of operations research and a lot of works has been published concerning this kind of problems. The surveys of Ernst *et al.* (2004a,b) give many references and new papers have been published since. Nevertheless, few of them concern activity and task assignment matching with our own definition.

The multi-activity assignment problem, which is a part of MATAP and MATTAP, has been addressed in Lequy *et al.* (2009). In this article, two exact modelss and several solution methods are proposed, including a rolling horizon heuristic based on a column generation model. Lequy *et al.* (2010) introduce tasks for forming the MATAP, describe an exact model and a two-stage heuristic composed by an approximation model and an activity assignment based on the rolling horizon heuristic of Lequy *et al.* (2009), and compare four different strategies for reassigning tasks during activity assignment. Prior to those articles, three master's thesis have been written concerning multi-activity assignment problem in the specific context of air traffic controllers with complex additional rules such as synchronization between trainees and instructors, first addressed by Omari (2002). Vatri (2001) integrates shift scheduling, shift assignment and activity assignment problems at the same time, but in a simplified context. No breaks are however scheduled in the shifts to reduce complexity of the problem. Therefore Bouchard (2004) proposes methods to take care of breaks thanks to accelerating heuristics.

Recently, Côté *et al.* (2009) solves multi-activity shift scheduling problem for unidentified employees by using mixed integer programs reduced thanks to constraint programming techniques. Thoses techniques yield filtering rules about breaks, rest periods and work streches and narrowing the set of feasible schedules. Bard et Wan (2006) address a scheduling problem similar to the multi-activity assignment problem where activities are executed in different workstation groups. Activities definition, even if they are called task, is coming from the case of a U.S. Postal Service mail processing and distribution center, and differs from ours due to activity duration constraints and satisfying demand. A combinaison between a integer programming model based on multi-commodity network and a tabu search provides results for the targeted data test. A case where movements between workstations are restricted is considered in Bard et Wan (2008). They compare a heuristic based on an aggregation of workstations versus a heuristic based on introduction of activity under-covering.

In this article, we first introduce the concept of team task which is quite general and fill some lacks in MATAP. Second we propose an exact model for the problem, which can be solved to optimality for small instances. Third we propose a two-module approach to solve real-life instances. Finally, computational results are reported in order to show efficiency of our approach.

This paper is organized as follows. First we defined the whole problem in Section 2 and give a global formulation in Section 3. We also explain the main lines of our resolution method in this section. Then we present in Section 4 the two modules on which our approach is based. The first one is an approximative model and the second one is the heuristic for assigning activities with task reassigning. Next we described our approach in Section 5 by introducing first the heuristic and next how to improve an initial task assignment thanks to both modules. Finally, computational experiments show robustness of our methods on randomly generated instances close to real-life in Section 6.

2 MATTAP definition

The MATTAP is a planning problem defined over a one week horizon. Activity forecasts lose accuracy for horizon greater than one week. In order to define the MATTAP, we have first to state some terminology and some concepts.

We discretize the horizon into disjoint periods of equal duration, typically, 15 or 30 minutes. In the following, we assume that all times correspond to the start or the end of a time period, and all durations are integer multiple of this time period length.

An employee works several shifts during a week. Each shift spans over a contiguous subset of periods during a day, characterized by a starting period and an ending period. A shift is itself composed by segments separated by breaks. A segment must be entirely filled by activity or task and no idle time is allowed : activity over-covering is prefered to increase customer service quality.

We categorize the work of an employee in three types : activities, single tasks, and team tasks. To perform one of them, an employee must possess certain qualifications (skills) specific to the activity or the task. Even being skilled, an employee is not allowed to perform an activity outside a minimal and a maximal duration given by the work rules. Moreover, a nonnegative number of employees is required for each activity for each time period of the horizon and define the demand curve for each activity.

A single task has a fixed length, must be completed in a given time window and have to respect some precedence relationships with other tasks. There are two types of precedence relationships. Two tasks linked by a strong relationship are either both assigned or both not assigned whereas the first task of a weak relationship can be assigned even if the second task is not assigned.

A team task has not a fixed length but can be assigned following what we call a team pattern. A team pattern is a set of couples composed by a number of employees (that can be one) and a time length. For instance, a team task can be performed by two employees during three hours, or by three employees during two hours. Each employee assigned to a team tasks starts and ends his work at the same time than

all other employees. Team tasks have also to be completed in a time window and to respect some precedence relationships. A pure team task cannot be performed by only one employee.

For convenience, single tasks are often considered as team tasks with a unique team pattern composed by one employee required and the time length of the task, even if they cannot be divided. However, considering team tasks as single task means that we keep only the team pattern with only one employee if it exists.

A transition occurs when an employee changes of work, that means he(she) switches from an activity to another one, from a task to a other task, or from type of work to another one. The beginning or the end of a segment does not count as a transition.

A feasible solution for the MATTAP satisfies therefore those constraints :
- exactly one activity or one task must be assigned in each time period of each segment ;
- the assignment of each activity must respect the activity duration bounds ;
- an employee cannot be assigned to an activity or a task without being qualified for it ;
- a task cannot be completed outside its time window ;
- all precedence relationships between the tasks must be respected ;
- the tasks involved in a strong precedence relationship must both be assigned or none of them.
- a team task must be assigned following a valid team pattern.

For activities, an under-covering occurs for each employee less than the demand for an activity in one time period. A under-covering incurs a cost that represent the loss in productivity for the corresponding activity and one time period. For tasks, an under-covering occurs when a task is not assigned and incurs also a cost.

The global cost of a feasible solution is a sum of the activity under-covering costs, the task under-covering costs and the transitions costs. We search for a feasible solution which minimizes the global cost.

3 Global Model

We first need to introduce some notation to design all the elements of the model. The main concept under this model is the concept of task patterns, explained further.

Let T be the set of periods got by discretizing horizon, $\mathcal{E}$ the set of employees and P the set of segments. Each employee $e \in \mathcal{E}$ works several fixed shifts and then works several segments denoted by P_e. A segment $p \in P_e$ spans on several periods denoted by $T_p \subset T$. Given a segment contains no breaks, T_p is a contigous subset of periods for each segment $p \in P$

Let A be the set of activities. Due to the discretization of the horizon, we consider the demand for each activity has the same value during each small period $t \in T$, denoted $d_{a,t}$. If the number of employees assigned to activity a during t is less than $d_{a,t}$, the difference between those two values is the number $u_{a,t}$ of under-coverings and the current solution contains $u_{a,t}$ basic periods of under-covering. The cost associated with a basic periods of under-covering is denoted by c_a and the total cost of the under-coverings equals to the sum over all.

Let B be the set of activity blocks. An activity block is potential assignment for an activity and for a qualified segment starting at given period with a length fulfilling the duration constraints over the activity. The activity blocks defined for activity a and segment p belongs to the subset $B_{p,a} \subset B$, which can be empty. The concept of blocks can be extented to the tasks and a task block is then a potential assignment for a task to a segment.

Let J be the set of tasks and D_j the set of team patterns associated with task j. Each team pattern $d \in D_j$ is related to a number of employees n_j^d and to a specific time length ℓ_j^d. Given this time length may be different among patterns D_j, let $W_j^d \subset T$ the subset of completion-valid starting time periods for task j if team pattern d is selected. That means if an assignment for task j with pattern d is assigned with a starting period in W_j^d, then this assignment ends in the time window associated with the task.

A task pattern is hence characterized by :

- a task $j \in J$
- a team pattern $d \in D_j$, being unique for single task.
- a starting period $t \in W_j^d$
- a subset of employee $E \subset \mathcal{E}$ such as $|E| = n_j^d$ and $\forall e \in E, \exists p \in P_e, [t, t+\ell_j^d] \subset T_p$.

The set of task patterns for a task $j \in J$ and $d \in D_j$ is denoted by $\mathcal{H}_j^d$. Then we defined $\mathcal{H}_j = \bigcup_{d \in D_j} \mathcal{H}_j^d$ and $\mathcal{H} = \bigcup_{j \in J} \mathcal{H}_j$. We also denote by $\mathcal{H}_p$ the subset of task patterns which can be assigned to segment p.

For a task pattern $h \in \mathcal{H}$, j_h, t_h and ℓ_h denote respectively the task, the starting period and the length associated with the pattern. Let P_h denote the subset of segments associated with pattern h.

Let δ_p^{min} be the minimal duration which may be taken by any assignment to segment $p \in P$ whatever activity is assigned. Let also be P^* the set of segments for which $\delta_p^{min} \geq 2$. For each $p \in P^*$, no activity or task blocks can have a starting period belonging to $\{2, \ldots, \delta_p^{min} - 1\}$ without yielding infeasibility in activity assignment. The same goes for ending periods at the end of the segments. All task patterns that do not respect these conditions are removed from the set $\mathcal{H}$.

Let Γ be a subset of $J \times J$ representing precedence relationships and $\bar{\Gamma} \subset \Gamma$ be the subset of strong precedence relationships. If (j, j') belongs to Γ, then task j must be completed before task j' starts.

For a period t and a segment p, let $\kappa_{p,t}^b$, $b \in B$ be the binary coefficient which indicates whether activity block $b \in B$ is defined for p and overlaps period t or not. Let $\kappa_{p,t}^h$, $h \in \mathcal{H}$ be the binary coefficient equals 1 if p belongs to P_h and task pattern h overlaps period t, 0 otherwise. Let M be a constant with a value big enough to nullify all the constraints where it appears.

We associate one binary variable $\mathbf{z}_j$ with each task $j \in J$ and one binary variable $\mathbf{y}_h$ with each task pattern $h \in \mathcal{H}$. $\mathbf{z}_j$ equals 1 if task j is not assigned, 0 otherwise. c_j is the cost for an uncovered task j. Concerning activities, $\mathbf{u}_{a,t}$ is a nonnegative variable indicating the number of under-coverings of activity a in period t and $\mathbf{x}_b$ is binary variable equal to 1 if activity block b is selected, and 0 otherwise. The cost of

variables $\mathbf{u}_{a,t}$ is $\underline{c}_a$ and c_τ is a unit cost for a transition.

For period t, a binary coefficient $s_{b,t}$, $b \in B$ equals to 1 if and only if block b starts at the beginning of period t, and 0 otherwise and a binary coefficient $e_{b,t}$ equals to 1 if and only if block b ends at the end of period t, and 0 otherwise. Similarly a binary coefficient $s_{h,t}$, $h \in \mathcal{H}$ indicates whether task pattern h starts at the beginning of period t or not.

Using this notation, the MATTAP can be formulated as the following mixed-integer program (MIP) :

$$\underset{\mathbf{u},\mathbf{x},\mathbf{z},\mathbf{y}}{\text{Minimize}} \quad \sum_{a\in A}\sum_{t\in T} c_a \mathbf{u}_{a,t} + \sum_{j\in J} c_j \mathbf{z}_j + c_\tau \Big(\sum_{b\in B} \mathbf{x}_b + \sum_{h\in\mathcal{H}} |P_h|\, \mathbf{y}_h - |P| \Big) \tag{8.1}$$

subject to :

$$\sum_{p\in P}\sum_{b\in B_{p,a}} \kappa_{b,t}\mathbf{x}_b + \mathbf{u}_{a,t} \geq d_{a,t}, \qquad \forall a \in A,\, \forall t \in T \tag{8.2}$$

$$\sum_{a\in A}\sum_{b\in B_{p,a}} \kappa^b_{p,t}\mathbf{x}_b + \sum_{j\in J_p}\Big(\sum_{h\in\mathcal{H}_j} \kappa^h_{p,t}\mathbf{y}_h\Big) = 1, \qquad \forall p \in P,\, \forall t \in T_p \tag{8.3}$$

$$\sum_{b\in B_{p,a}} e_{b,t-1}\mathbf{x}_b - \sum_{\substack{a'\in A\\ a\neq a'}}\sum_{b\in B_{p,a'}} s_{b,t}\mathbf{x}_b - \sum_{h\in\mathcal{H}_p} s_{h,t}\mathbf{y}_h \leq 0, \qquad \forall p \in P,\, t \in T_p,\, a \in A \tag{8.4}$$

$$\sum_{h\in\mathcal{H}_j} \mathbf{y}_h + \mathbf{z}_j = 1, \qquad \forall j \in J \tag{8.5}$$

$$M\mathbf{z}_{j'} + \sum_{h'\in\mathcal{H}_{j'}} t_{h'}\mathbf{y}_{h'} - M\mathbf{z}_j - \sum_{h\in\mathcal{H}_j} (t_h + \ell_h)\,\mathbf{y}_h \geq 0, \qquad \forall (j,j') \in \Gamma \tag{8.6}$$

$$\mathbf{z}_j - \mathbf{z}_{j'} \geq 0, \qquad \forall (j,j') \in \bar{\Gamma} \tag{8.7}$$

$$\mathbf{u}_{a,t} \geq 0, \qquad \forall a \in A,\, \forall t \in T \tag{8.8}$$

$$\mathbf{x}_b \in \{0,1\}, \qquad \forall b \in B \tag{8.9}$$

$$\mathbf{y}_h \in \{0,1\}, \qquad \forall h \in \mathcal{H} \tag{8.10}$$

$$\mathbf{z}_j \in \{0,1\}, \qquad \forall j \in J \tag{8.11}$$

Objective function (8.1) is composed by three terms : the cost of under-coverings for activities, the cost for uncovered tasks and the cost for the transitions. The first two costs are easy to understand, but the last is not obvious. Given that each segment is entirely assigned, the number of transitions for a segment p equals to the number of different assigned activity blocks and task patterns in this segment, minus one :

$$\sum_{a \in A} \sum_{b \in B_{p,a}} \mathbf{x}_b + \sum_{h \in \mathcal{H}_p} \mathbf{y}_h - 1.$$

Summing over all segments and simplifying the triple sum notation yields the formula of the objective function.

Inequalities (8.2) are the soft activity demand constraints and allow one to compute the numbers of under-coverings for each activity and each period. Equalities (8.3) guarantee no idleness. Constraints (8.4) force two activity blocks following each other to correspond to two different activities. Equalities (8.5) are task covering constraints and compute task under-coverings. Inequalities (8.6) guarantee to fulfill weak precedence relationships, that means time relationship and assignment relationship. Constraints (8.7) enforce strong precedence relationships. Domains of variables are defined by constraints (8.8) – (8.11).

Except for small instances, this model is too hard to solve because of the huge size of $\mathcal{H}$ combined to precedence relationships. For very few tasks spanned over a horizon of one day, it is still possible to get an integer solution but solving this model with an exact method for large instances consume too mcuh time.

The problem without any team tasks is already hard to solve as described in Lequy *et al.* (2010), where an heuristic based on mathematical programming already is applied to find a good feasible task and activity assignment. In their paper, the resolution is decomposed in two phases. In a first phase, an approximation model find a feasible solution concerning task by relaxing some constraints on activities such as minimal duration and by modifying objective function. In a second step, a rolling horizon heuristic assigns activities with flexibility on task already assigned. This method allows to decrease the impact of bad decisions taken during first phase.

Our approach is based on the same principle and is also composed by two modules. The two modules belongs however to a more general framework of optimization, which uses this two modules to explore some neighbourhoods. This heursitic is described in Section 5.1. In its local search, the first module tries to improve an task assignment thanks to the approximation model described in Section 4.1 and the second module is a rolling horizon heuristic with task reassigning, presented in Section 5.2. We detail first mathematical models used in our approach in Section 4 and the approach itself in Section 5

4 Modules for the heuristic

4.1 Approximation model for tasks assignment

In this section, we want to find a feasible task assignment which fulfill precedence relationships but without considering all the constraints on activities. We can obtain such a task assignment through an approximation model, similar to the one in Lequy *et al.* (2010). We describe first all additional notation we need to use to formulate the approximation model and then our formulation.

To count the number of employees associated with activity a during period t, let $\mathbf{x}_{p,a,t}$, $p \in P$, be a continuous variable equals to the fraction of employee owning segment p assigned to activity a during period $t \in T_p$.

Let $\kappa_{p,t}^h \in \{0,1\}$ be a boolean coefficient which indicates whether or not task pattern $h \in \mathcal{H}$ is covered in segment $p \in P$ during period $t \in T_p$. The approximation model is the following one :

$$\underset{\mathbf{u},\mathbf{x},\mathbf{z},\mathbf{y}}{\text{Minimize}} \sum_{a \in A} \sum_{t \in T} c_a \mathbf{u}_{a,t} + \sum_{j \in J} c_j \mathbf{z}_j \tag{8.12}$$

subject to :

$$\sum_{p \in P} \mathbf{x}_{p,a,t} + \mathbf{u}_{a,t} \geq d_{a,t}, \qquad \forall a \in A, \forall t \in T \tag{8.13}$$

$$\sum_{a \in A} \mathbf{x}_{p,a,t} + \sum_{h \in \mathcal{H}_p} \kappa^h_{p,t} \mathbf{y}_h = 1, \qquad \forall p \in P, \forall t \in T_p \tag{8.14}$$

$$\sum_{h \in \mathcal{H}_j} \mathbf{y}_h + \mathbf{z}_j = 1, \qquad \forall j \in J \tag{8.15}$$

$$M_{j'} \mathbf{z}_{j'} + \sum_{h' \in \mathcal{H}_{j'}} t_{h'} \mathbf{y}_{h'} - M_j \mathbf{z}_j - \sum_{h \in \mathcal{H}_j} (t_h + \ell_h) \mathbf{y}_h \geq 0, \qquad \forall (j, j') \in \Gamma \tag{8.16}$$

$$\mathbf{z}_j - \mathbf{z}_{j'} \geq 0, \qquad \forall (j, j') \in \bar{\Gamma} \tag{8.17}$$

$$\sum_{\substack{h \in \mathcal{H}_p \\ t_h + \ell_j = t}} \mathbf{y}_h + \sum_{\substack{h' \in \mathcal{H}_p \\ t_{h'} \in [t+1, t+\delta^{min}_p - 1]}} \mathbf{y}_{h'} \leq 1, \qquad \forall p \in P^*, \forall t \in T_p \tag{8.18}$$

$$0 \leq \mathbf{x}_{p,a,t} \leq 1, \qquad \forall b \in B \tag{8.19}$$

$$\mathbf{u}_{a,t} \geq 0, \qquad \forall a \in A, \forall t \in T \tag{8.20}$$

$$\mathbf{y}_d \in \{0, 1\}, \qquad \forall d \in D \tag{8.21}$$

$$\mathbf{z}_j \in \{0, 1\}, \qquad \forall j \in J \tag{8.22}$$

Objective function (8.12) aims at minimizing only task and activity under-coverings. Inequalities (8.13) are simplified version of covering constraints (8.2). Equalities (8.14) enforce filling entirely the segments. Constraint sets (8.15)–(8.17) play the same role that constraint set (8.5)–(8.7). Inequalities (8.18) ensure that assigned task patterns in a segment are not too close from each other such as allowing an activity block to be assigned in between. Finally constraints (8.19)–(8.22) define the domain of the decision variables.

The model will be solved by a commercial solver. It becomes however more and more difficult to get even an integer solution in a reasonnable amount of time when the number of tasks increases. That's why in our heuristic, some variables will forced to take specific value depending on the needs of the heuristic.

4.2 Module for activity assignment

For activity assignment, we use a rolling horizon heuristic with task reassigning similar to the one described in Lequy *et al.* (2010), but with a specificity for team task : synchronicity between task blocks of the same team task must be ensured when the tasks is up to be moved. Given that the module is used in an higher level framework, we have to make the hypothesis that a team pattern has already be choosen for each task j and is denoted d_j.

We described in this section the assignment model for the whole horizon, but only a time restriction is used in the rolling horizon heursitics. For this formulation, we call sequence a subset of activity and task blocks which may be well assigned to a segment. For each segment $p \in P$ Ω_p denote the set of feasible sequences and binary variables $\theta_{p,s}, s \in \Omega_p$ equal to 1 if the sequence s is selected for segment p denote sequence variables. A variable $\theta_{p,s}$ has a cost $c_{p,s}$ equal to the sum of transition costs. To garantee that each task block of a task pattern starts at the same period for each team tasks, we need to introduce synchronicity variables. Let $R_{j,h}$ be a binary variable indicating whether or not task $j \in J$ is assign following the task pattern $h \in \mathcal{H}_j^{d_j}$. Let $\kappa_j^{p,s}$ be a binary parameters indicating whether sequence $s \in \Omega_p$ for segment $p \in P$ contains a task block for task $j \in J$ following pattern d_j or not. Let $t_j^{p,s}$ be the starting period of block for task $j \in J$ in sequence $s \in \Omega_p$ for segment $p \in P$ if it exists, 0 otherwise. The assignment model is then :

$$\underset{\mathbf{u},\theta}{\text{Minimize}} \quad \sum_{a \in A} \sum_{t \in T} c_a U_{a,t} + \sum_{p \in P} \sum_{s \in \Omega_p} c_{p,s} \theta_{p,s} \tag{8.23}$$

subject to

$$\sum_{s\in\Omega_p} \theta_{p,s} = 1, \qquad \forall p \in P \tag{8.24}$$

$$\sum_{p\in P}\sum_{s\in\Omega_p} \kappa_{a,t}^{p,s}\theta_{p,s} + U_{a,t} \geq d_{a,t}, \qquad \forall a \in A, \forall t \in T \tag{8.25}$$

$$\sum_{h\in\mathcal{H}_j^{d_j}} R_{j,h} = 1, \qquad \forall j \in J \tag{8.26}$$

$$\sum_{p\in P}\sum_{s\in\Omega_p} \kappa_{j}^{p,s}\theta_{p,s} = n_j^{d_j}, \qquad \forall j \in J \tag{8.27}$$

$$\sum_{s\in\Omega_p} t_j^{p,s}\theta_{p,s} - \sum_{h\in\mathcal{H}_j^{d_j}} t_h R_{j,h} = 0, \qquad \forall j \in J, \forall p \in P \tag{8.28}$$

$$\theta_{p,s} \in \{0,1\}, \qquad \forall p \in P, \forall s \in \Omega_p \tag{8.29}$$

$$U_{a,t} \geq 0, \qquad \forall a \in A, \forall t \in T, \tag{8.30}$$

$$R_{j,h} \in \{0,1\}, \qquad \forall j \in J, \forall h \in \mathcal{H}_j \tag{8.31}$$

where binary parameters $\kappa_{a,t}^{p,s}$ equal to 1 if sequence s contains an assignment to activity $a \in A$ at period $t \in T_p$, and 0 otherwise.

Objective function (8.23) aims at minimizing the sum of under-coverings and the sum of transitions cost produced by selected sequences. Equalities (8.24) garantee that a sequence per segment is selected. Inequalities (8.25) are activity demand constraints and constraints (8.27) stipulates that each task has only one assigned block in the final solution. Equalities (8.26) ensure that only one task pattern is selected. Equalities (8.27) garantee that, for each task, the number of task blocks in final selected sequences matches with the number of task blocks in selected team task pattern. Equalities (8.28) force team blocks in sequences for segment p to start at the selected period. Constraints (8.29)–(8.31) define the domain of the decision variables. This model is refered further as the activity assignment model for team task (AAMFTT).

AAMFTT has to be solved with the help the column generation process. Column generation is an iterative process to get optimal solution of a specific kind of linear program (Dantzig et Wolfe, 1960; Desaulniers *et al.*, 2005; Lübbecke et Desrosiers, 2005). The column generation process is divided in two main stages. The first one consists of solving a restriction of the linear program by taking in consideration only

a subset of the huge column set. This restriction is called the restricted master problem and its resolution provides dual variables for the second stage. The second stage consists then of finding columns violating optimality conditions for restricted master problem by solving subproblems with the reduced cost formula as an objective function. If such columns are found, corresponding columns are added to restricted subset of columns and stage one is applied again. Otherwise, process is stopped and the obtained solution is also optimal for the initial linear problem.

A subproblem is here equivalent of solving a shortest path problem in the network associated with a segment. A network is defined for each segment p and is composed by arcs representing either a task block, an activity block or a transition between them.

During the activity assignment, task reassignments are allowed given some flexibility windows which garantee precedence relationships are still respected and ensure that subproblems keep being basic shortest path problems in their associated temporal networks.

Even for instances coming from slice of the rolling horizon, we suppose that we work only with the team pattern d_j coming from the the task assignment and task j is associated a subset F_j of contigous periods called flexibility window, such as assignment for task j must start at a period in F_j. Let f_j^s and f_j^e two periods such as

$$f_j^s = \max\left\{\min_{t\in W_j} t; \max_{j_p\in J|(j_p,j)\in\Gamma}\left\lfloor\frac{t_{j_p}+\ell_{j_p}^{d_{j_p}}+t_j}{2}\right\rfloor; t_j-\left\lfloor\frac{\ell_j}{2}\right\rfloor\right\}$$

and

$$f_j^e = \min\left\{\max_{t\in W_j} t; \min_{j_s\in J|(j,j_s)\in\Gamma}\left\lfloor\frac{t_j+\ell_j^{d_j}+t_{j_s}}{2}\right\rfloor-\ell_j; t_j+\left\lfloor\frac{\ell_j}{2}\right\rfloor-1\right\}.$$

Then $F_j=\{f_j^s,\ldots,f_j^e\}$ and we denote ℓ_j the number of periods $\ell_j^{d_j}$.

By taking the flexibility windows in account for task starting periods, network for segment $p\in P$ contains following nodes :

α_p **:** the source node ;

β_p **:** the sink node ;

$K_{a,t}$: the beginning of an activity block for activity a starting at period t ;

$L_{j,t}$: the beginning of a task block for task j starting at period t ;

$G_{a,t}$: the end of an activity block for activity a ending at period t ;

$H_{j,t}$: the end of a task block for task j ending at period t.

Nodes are linked by arcs given the following rules. α_p is linked to all nodes K_{a,s_p} and all nodes H_{j,s_p} at the starting period s_p of the segment p. β_p is linked to all nodes E_{a,e_p} and nodes Q_{j,e_p} at the ending period e_p of the segment p. A transition arc links a node $E_{a,t}$ or a node $Q_{j,t}$ to a node $B_{a,t+1}$ or a node $H_{j,t+1}$. An activity arc links a node $B_{a,t}$ to a node $B_{a,t+d-1}$ where d is a valid duration for an activity block. A task arc links a node $H_{j,t}$ to a node $Q_{j,t+\ell_j-1}$ where t belongs to F_j. The network may be cleaned by keeping only arcs belonging to a path between α_p and β_p. This operation may be performed by a simple forward-backward labelling algorithm. Such a network is illustrated in figure 8.1 where empty round nodes represent either a node $K_{a,t}$ or $L_{j,t}$ and square nodes represent either a node $G_{a,t}$ or $H_{j,t}$.

Let σ_p, $p \in P$, $\pi_{a,t}$, $a \in A$, $t \in T$, ρ_j, $j \in J$ and $\mu_{j,p}$, $j \in J$, $p \in P$ be the dual variables associated with constraints set (8.24), (8.25), (8.27), and (8.28), respectively. Each transition arc has a cost c_τ. A task block for task j and segment p starting at period t has a cost of $-\rho_j - t\mu_{j,p}$. An activity arc between B_{a,t_i} and E_{a,t_j} has a cost $-\sum_{t_i \le t \le t_j} \pi_{a,t}$. All the arcs outgoing from α_p has a cost of $-\sigma_p$. The shortest path in this network has the same value than the minimum reduced cost of variables $\theta_{p,s}, s \in \Omega_p$ in linear relaxation of (8.23)–(8.31) given a basis.

In the example of figure 8.1, the segment spans from period 1 to period 8. Employee is qualified for two activities, a^1 and a^2, and two tasks, j^1 and j^2. Durations for activity a^1 must lay in $[4, 8]$ and in $[2, 3]$ for activity a^2. For task j^1, flexibility window F_{j^1} equals to $[1, 3]$ and the task length is 4. For task j^2, flexibility window F_{j^2} equals to $[4, 5]$ and the task leng is 2.

Two heuristics are applied to the assignment model for obtaining an integer solution. The first heursitic looks for an integer solution directly with the last restricted master problem solved to get the linear solution of the assignment model but with integrity constraints. A MIP solver is used with a maximum time limit. If the heuristics is not deemed successful, that means if no integer solution has been found or

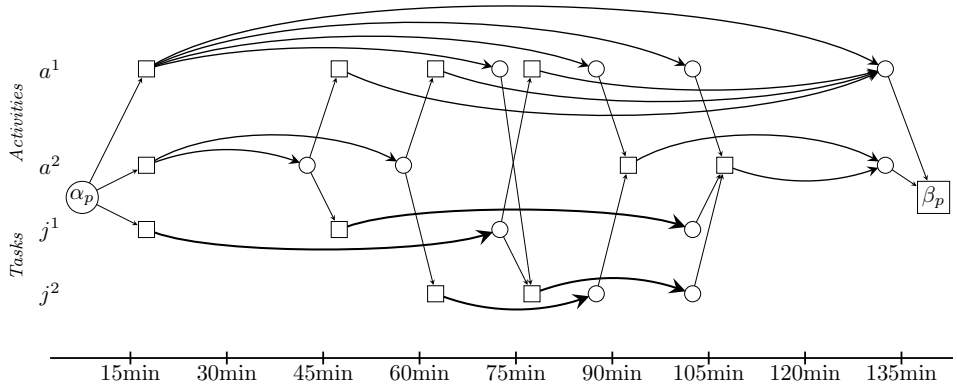

FIGURE 8.1 – Example of a cleaned network

the objective function value of the integer solution is too far of the objective function value of the relaxation, an iterative rounding procedure is applied.

This second heuristic consists of fixing sequentially a choosen binary variable in a permanent way until the column generation method provides an integer linear relaxation solution. Fixing $\theta_{p,s}$ without fixing first variables $R_{j,h}$ may lead to infeasibility. For avoiding this kind of situation, binary variables $R_{j,h}$ should have to be fixed to integer values prior fixing any of $\theta_{p,s}$.

The following iterative rounding procedure is then applied on variables $R_{j,h}$. During an iteration of this procedure, the highest fractional value variable among $R_{j,h}$ is selected. If this variable has never been selected, it incurs a bonus N and its cost becomes $-N$, in order to promote this variable to take the value 1. Its fractional value is else compared to the previous value that the variable took when it was selected. If the previous value is lower than the current value, an other bonus is given to variable and thus $-N$ is substract from its current cost. Otherwise, the variable is fixed to 0. The first case means that the previous bonus may be too low to allow the current variable to take the value 1 and this variable receives a bigger bonus to check if it is the case. Given the ideal value of the bonus is not known, a higher bonus than the previous one is given while the fractional value increases. If the fractional value stops increasing, that means fixing this variable to 1 must lead to a empty solution space. Hence the variable is fixed to 0. This procedure ends when all the $R_{j,h}$ are integer in the linear relaxation solution. When such a solution is obtained, all the arcs in net-

works representing a task j which do not match with the selected team task pattern $R_{j,h}$ are removed from the networks as well as the generated variables in the master problem. Then the iterative procedure to find the final integer solution starts. At each iteration, variable $\theta_{w,s}$ with the highest fractional value in the linear relaxation solution is selected to be fixed to 1 and column generation method is started over again to provide the linear relaxation solution ensuing from this decision for the next iteration.

Some accelerating strategies can also be applied to AAMFTT and its subproblems : density matrix reduction for master problem and arc density reduction for networks. Both methods are explained in (Lequy *et al.*, 2009) and can be applied without modifications.

5 Heuristic for MATTAP

Getting solution for MATTAP directly by solving model (8.1)–(8.11) take too much computational time for large size instance. That's why we developped a heursitic based on the two modules described in Section 4.

5.1 Variable Neighbourhood Descent

To get good feasible solutions in a short amount of time, we use the variable neighbourhood descent, denoted VND, which is a part of the well-known variable neighbourhood search. This metaheuristic uses a system of imbricated neighbourhoods (Mladenović et Hansen (1997), Hansen et Mladenović (2001)) in order to improve sequentially an initial solution. Typically applied on combinatorial problem such as traveling salesman problem or clustering problem as p-median problem, it has been also successfully applied to provide integer solution for large MIP instances (see Lazić *et al.* (2009)).

The heuristic is described in Algorithm 3. As arguments, function `VND` takes then a initial solution S of the problem, a minimal size k_{min} of the neighbourhood system,

Algorithme 3: `VND` pseudo-code

Entrées : S, k_{min}, k_{max}, t_{vnd}, t_{mip}
Sorties : S

```
1  t_start ⟵ cpuTime()
2  t ⟵ 0
3  v_best ⟵ ∞
4  tant que t < t_vnd and v(S) < v_best faire
5  |  k ⟵ k_min
6  |  v_best ⟵ v(S)
7  |  tant que t < t_vnd and k ≤ k_max faire
8  |  |  TimeLimit ⟵ min(t_mip, t_vnd − t)
9  |  |  S' ⟵ LocalSearch( S, k, TimeLimit )
10 |  |  si v(S') < v(S) alors
11 |  |  |  S ⟵ S'
12 |  |  |  k ⟵ k_min
13 |  |  sinon
14 |  |  |  k ⟵ k + 1
15 |  |  fin
16 |  |  t ⟵ cpuTime() − t_start
17 |  fin
18 fin
```

a maximal size k_{max}, a time limit for solving integer program t_{mip} and a time limit t_{vnd} for the whole algorithm. The cost of a solution S is denoted by $v(S)$ and function `VND` return the best solution occurring during the search.

The function `LocalSearch`(S, k, $TimeLimit$) is described in Section 5.2 and consists of finding in the neighbourhood of size k around S a solution which may improve the current solution. The modules described in Section 4 are involved in this function.

The function `VND` requires an initial solution to start with. We could use a solution where no tasks are assigned, but preliminary tests show that a solution where all the tasks are assigned provide better results. Depending of the nature of the team tasks, we have two cases. If all the tasks possess a team pattern containing only one employee, we simply consider that all the tasks are single tasks and use the heuristic for MATAP to find the initial solution. Otherwise, we use the heuristic for MATAP on the instance without considering the problematic team tasks, and the VND should

assign those tasks during the search due to their under-covering costs.

5.2 Function LocalSearch

The function `LocalSearch` take three arguments : the current solution S, the size of the neighbourhood k and a time limit. The local search is divided in two parts. First an approximative model is used as a surrogate to find the best task assignment in the neighbourhood. Then the task assignment is evaluated thanks to a activity assignment heuristic.

First, approximative model (8.12)–(8.22) provide a new task assignment as follows. Given integer solution S, each task j incurs a weight w_j and tasks are then sorted by their increasing weights. Variables $\mathbf{z}_j$ and $\mathbf{y}_d, d \in D_j$ are fixed to their value in S if task j is in the first $|J|-k$ ordered tasks. Other decisions variables can take whatever value in their domains. Then a MIP solver tries during $TimeLimit$ seconds to find the best solution it can for the model. Task assignment S is given to the solver as a reference for the branch-and-bound.

To define the weights, we first need to introduce $\mathcal{P}$ a partition of J such as every task j in $I \in \mathcal{P}$, j has only precedence relationships with other task in I. That means, $\forall I' \in \mathcal{P}, \forall j' \in I', (j,j') \notin \Gamma$ and $(j',j) \notin \Gamma$. If $\mathcal{P}_j$ denote the subset $I \in \mathcal{P}$ such as $j \in I$ and $\bar{T}$ denote $1 + \max_{t \in T} t$, the weight for a task j are equal to

$$w_j = \mathbf{z}_j \min_{j' \in \mathcal{P}_j} \left(\min \left(W_{j'} \right) - \phi_{j'} \bar{T} \right) - \sum_{h \in \mathcal{H}_j} (t_h - \phi_j \bar{T}) \mathbf{y}_h$$

where ϕ_j is the number of times a task has be selected to be free in `LocalSearch`. We reset this number to zero at line 17 of Algorithm 3. With this kind of weigths, we first favor groups of precedence relationships which are not fulfilled to be treated in the same time and we favor then tasks in the same time area to be treated with each other. When k_{max} is reached, the loop is done again with the new task assignment and therefore new weights, in comparison to the ones of the previous loop.

The activity assignment heuristic is a rolling horizon, based on a mathematical model described in section 4.2, which works as follows. The whole time horizon is divi-

ded in overlapping time slices. For each slice in chronological order, the mathematical model is applied while keeping most information of the previous slice (what is in the overlapping section is reconsidered in the current section model). For each slice, the model AAMFTT is then solved by using the heuristics described in the same section and the results is translated into assignments for the global solution.

5.3 FastVND

We can accelerate the function `VND` by changing the second module and adjusting parameters. For evaluating S, it is faster to assign activities without reassigning task. This case is equivalent to force variable R_{j_h} to take the value 1 if the corresponding task pattern appears in the task assignment. The master problem can be simplified (all the constraints concerning task can be removed) as well as subproblems (only the task arcs matching with the tasks assignment are kept and activity arcs overlapping them can be removed). In order to avoid satisfying test at the line 9 of Algorithm 3 when the solution is improved only by few transitions, we consider a different cost v_2 instead of v for evaluating a solution, with

$$v_2(S) = \left\lfloor \frac{v(S)}{\min\limits_{a \in A} c_a} \right\rfloor .$$

To get the final solution of the VND, the second module with task reassigning should be performed at the end of the function. This version of the VND is called FastVND.

6 Computational experiments

We conducted a series of computational experiments on randomly generated instances. In this section, we report the results of these experimentations after describing the test instances and our experimental plan.

The instances are randomly generated in order to look like real-life instances, which may be encountered in large retail stores. In those instances, employees work 8-hours

shifts with a half-an-hour break in the middle but their workload is limited to 35 hours per week. Almost all the shifts are centered around noon, but 20% of them take place during night. Employees are qualified for about half of the five activities, which have a minimal duration about one hour and a maximal duration about three hours. The one-week horizon is divided in 15-minutes periods. An activity under-covering costs 1500 and a task under-covering costs 15000 multiple by the the task duration. This last cost is very expensive because we favor activity under-coverings rather than task under-coverings to match with our context. In a instance, we deal with 40 tasks, a mean length of 180 minutes and approximatively 25 precedence relationships. We have two types of team tasks in thoses instances. For the first test bench, all the tasks can be split in two or not, and we consider only a subset of them as team tasks depending of the situation. In the second test bench, pure team tasks (tasks requiring at least two employees) are generated by splitting single tasks and removing the possiblity to assign them to only one employee. All precedence relationships are strong ones.

Parameters of our commercial solver are set to default value for the concern algorithm, except for branch-and-bound absolute tolerance which is set to 1500. This cost is the one of an activity under-covering because we consider that a solution with an optimal number of under-covering is good enough to be consider as an optimal solution, even if it is not the case. All tests were performed on an *Intel®Core™2 CPU 6700* processor clocked at 2.66GHz with 4GB RAM. We used Xpress solver of *Fair Isaac Corporation* for solving linear and integer programs.

6.1 Experimentations concerning task split and FastVND

First, we want to compare how the VND perform when task can be split. To do so, we created a bench of 10 instances where all the tasks can be performed either by only one employee or by two employees. In this last case, when the length of the tasks is not a even number of periods, we consider that the synergy between employees reduces the total number of periods for the tasks. To be fair, we have balanced the number of team patterns with and without synergy. That means half of the tasks is considered as synergetic whereas the other half is considered as regular. With this

test bench, we applied our heuristic on the instances where :

- we consider all the tasks as single tasks (*Control*) ;
- we consider only regular tasks as team tasks, the others as single tasks (*VND without synergy*) ;
- we consider only synergetic tasks as team tasks, the others as single tasks (*VND with synergy*).

Our parameters for VND are $t_{vnd} = 36000$ and $t_{mip} = 600$. Parameters k_{min} and k_{max} depends on the solver computation power and on the number of tasks. For our experimentations, we take the value $k_{min} = 0.5\sqrt{2\,|J|} = 4$ and $k_{max} = \sqrt{2\,|J|} = 9$ in order to search on $|J|$ tasks if the time limit allows it.

Table 8.2 reports as follows the results of the runs we performed. The first column reports the identifier of the instance. Then we report the number of under-covering for the initial solution obtained by the heuristic for MATAP, and its computational time in seconds. Next we report the number of under-covering in the final solution, the total computational time and the number of call of `LocalSearch` for each of the three cases. For the VND without synergy and the VND with synergy, we reports also the number of task which appear split in the final solution. All methods return solutions where all the tasks are assigned. The best under-covering score is highlight with boldface for each instance.

Id	**Initial**		**Control**			**Without synergy**				**With synergy**			
	U.-c.	Time	U.-c.	Time	iter.	U.-c.	Time	# Split	iter.	U.-c.	Time	# Split	iter.
1	38	128	33	2416	30	33	4136	0	28	**29**	9045	5	41
2	29	73	27	395	7	27	2285	0	22	**24**	2833	5	33
3	58	50	56	958	30	56	863	0	15	**55**	2238	1	31
4	50	53	45	1834	42	**40**	2219	4	41	46	1205	1	23
5	35	44	**31**	1078	26	32	2137	2	49	32	1667	1	33
6	67	95	53	2134	40	53	12242	2	85	**48**	24469	10	104
7	21	54	20	1909	44	16	3845	3	51	**15**	7050	2	51
8	22	59	16	437	7	**14**	3749	4	39	17	2793	2	34
9	25	72	24	391	7	**22**	2222	1	45	23	2882	3	38
10	23	72	21	893	19	**19**	3011	3	57	20	3055	3	56

TABLEAU 8.1 – Results for regular VND

We observe that the initial solution is improved all the time. We can also remark a big improvment for instances 1,2, and 6 in the cases of VND with synergy, what match with several split of tasks. Except for instance 4 and 8, the VND with synergy is not outclass by other cases. Control got only one time the best solution but the other cases are close. Given our method is a heuristic, some "bad" results can be observed as for instance 4 and 8 in the case of VND with synergy. VND without synergy is more even, and got always good results. Results are interesting for three reasons. First, it shows that the heuristic for MATAP performed well in relation to its computation time. Second, the VND itself is able to improve the initial solutions, even without team patterns. Third, allowing tasks to be split leads to solutions of a better quality, especially with synergy. As might expected, computation times increase when the VND is used with team tasks.

Now we present the same results by applying FastVND instead of VND on the same test bench. Our new parameters are $t_{vnd} = 1800$, $t_{mip} = 120$. The results are reported in table 8.1 exactly as in table 8.2.

Id	**Initial**		**Control**			**Without synergy**				**With synergy**			
	U.-c.	Time	U.-c.	Time	iter.	U.-c.	Time	# Split	iter.	U.-c.	Time	# Split	iter.
1	38	128	33	1398	19	**32**	2049	2	22	**32**	2113	1	16
2	29	73	29	403	7	29	525	0	7	**24**	1462	4	21
3	58	50	57	516	15	**56**	427	0	7	**56**	946	0	15
4	50	53	49	720	15	**39**	1658	2	32	45	1043	2	21
5	35	44	32	1087	27	31	1268	0	31	**30**	1542	2	31
6	67	95	53	1141	21	55	1450	0	21	**52**	2023	5	24
7	21	54	21	985	22	18	1404	1	22	**17**	1641	0	19
8	22	59	15	1547	31	15	1680	1	29	**14**	2015	5	31
9	25	72	24	1219	27	23	1142	1	21	**19**	1543	3	29
10	23	72	22	858	18	**19**	1257	3	23	20	1206	3	19

TABLEAU 8.2 – Results for FastVND

The FastVND provide results almost as good as the results of VND, and the trends of table 8.1 are globally preserved, even if cases with split tasks are closer in terms of best results. The good performance of the the FastVND can be explained by the following reasons. First, both methods are descent heuristics with no diversification

process, then they are sensible to the initial direction they take, which can lead to different local optimals. Second, objective function (8.12) acts as a surrogate of function $v(x)$ and has thus to match quite well with it. In FastVND, the evaluation function is better represented by the surrogate than in VND because transitions have less influence.

6.2 Experimentations for assigning pure team tasks

We want to show here than our heuristic is able to assign team tasks even if they do not appear in the initial solution. We performed thus tests on instances containing pure team tasks, which do not appear in the initial solution. The test bench is composed by ten instances which contain from 1 to 10 pure team tasks over 40 tasks. The results are presented in table 8.3. In this table, each line reports the means for ten instances with the same number of pure team tasks. The first column reports the number of pure team tasks for all the ten instances. The second column reports the number of tasks assigned in the initial solution, the computational time appears in the third column. The fourth and fifth columns report the same data for the VND. We use the FastVND with $t_{vnd} = 1800$, $t_{mip} = 120$.

Number of	**Initial Solution**		**VND Improvment**	
team tasks	Tasks	Time	Tasks	Time
1	37.5	37.2	40	592.6
2	33.8	29.4	38.8	603.7
3	31.3	30.2	40	677.6
4	29.2	35.6	40	735.8
5	27.2	26.0	40	802.8
6	24.8	25.8	40	748.2
7	23.5	25.2	40	827.6
8	22.1	26.5	40	810.4
9	21	25.1	39.3	800.0
10	19.7	24.7	40	852.4

TABLEAU 8.3 – Results for assigning pure team tasks with FastVND

Except for two instances, i.e. in 98 cases over 100, our heuristic is able to assign all

the team tasks in less than 15 minutes. As might be expected, the more tasks there are, the bigger are the computational time. For the two failures, our algorithm is able to assign more tasks than the initial solution, but did not success in assigning all the tasks. The reason may be that some assigned tasks should mutually keep the local search from assigning some other tasks (surely linked by precedence relationships). We are not even sure that a full tasks assignment for those instances exists.

7 Conclusion

This paper has introduced the Multi-Activity and Team Task Assigment Problem, with due date time window, skills and precedence relationships. We proposed one mixed integer programming formulation and a two-module heuristic for tackling large scale instances, that means instances over one week and fourty tasks. We perfomed experimental tests on randomly generated instances in order to validate our heuristic methods. If splitting tasks is allowed, our heuristic is able to improve quality solution in a reasonable amount of computation time in comparison to a solution with single tasks only. Moreover our heuristic can assign pure team tasks which do not appear in the initial solution.