Université Fédérale

Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace

Présentée et soutenue par :
Adrinana PACHECO

le jeudi 17 décembre 2020

Titre :

Techniques de décisions hiérarchiques pour l'allocation et l'ordonnancement de tâches
Hierarchical decision techniques for task allocation and scheduling

---

École doctorale et discipline ou spécialité :

EDSYS : Informatique et Robotique

Unité de recherche :

Équipe d'accueil ISAE-ONERA CSDV

Directeur(s) de Thèse :

M. Cédric PRALET (directeur de thèse)
Mme Stéphanie ROUSSEL (co-directrice de thèse)

Jury :

M. Alain HAÏT Professeur ISAE-SUPAERO - Président
M. Hadrien CAMBAZARD Maître de conférences INP Grenoble - Examinateur
Mme Sophie DEMASSEY Maître assistante Mines ParisTech - Rapporteure
Mme Marie-Jo HUGUET Professeur INSA Toulouse - Examinatrice
M. Cédric PRALET Ingénieur de recherche ONERA - Directeur de thèse
M. Alain QUILLIOT Professeur des Universités Clermont Auvergne - Rapporteur
Mme Stéphanie ROUSSEL Ingénieure de recherche ONERA - Co-directrice de thèse
Mme Christine SOLNON Professeure INSA Lyon - Examinatrice

Agradecimientos

En primer lugar quiero agradecer a mis tutores Cédric y Stéphanie por su tiempo y dedicación y por su excelente labor como mentores durante los tres años de mi doctorado, impulsándome hacia adelante y motivándome cada día a ser más independiente en mi trabajo. De ellos me llevo muchas enseñanzas tanto a nivel técnico como a nivel personal.

Quiero agradecer a mi familia. A mis padres por ser un apoyo emocional muy grande todos los días a pesar de la distancia, por apoyarnos a mi hermano y a mi para cumplir todos nuestros sueños, y por inculcarme el amor por el aprendizaje (y por los enigmas y acertijos matemáticos) desde que tengo memoria. A mi hermano por estar siempre ahí para mi y por, sin saberlo, darme la motivación que me faltaba este año. Sin ellos no hubiera logrado este sueño!

A mis profes, Hugo León Benjumea y Fabio León Restrepo, quienes fueron los primeros en mostrarme el maravilloso universo de la física, de la geometría, del álgebra..., y quienes supieron transmitirme su pasión por la ingeniería cómo nadie más lo ha hecho. A mi profe Alejandro Montoya quien me permitió reorientar mi carrera profesional al enseñarme todo un nuevo universo, el de *l'ordonnancement*, y quien influyó mucho en la elección de este doctorado.

A Thomas por compartir conmigo tanto tiempo de calidad y por enseñarme tantas cosas importantes, y a los Piekolek por recibirme en su familia y por tomarse el tiempo de revisar mi manuscrito. A Migue por ser para mi un buen amigo y un ejemplo a seguir, y a Daniel, Cindy, Alejo, Andrés y Liz por las risas y todos los buenos momentos en los almuerzos juntos.

Quiero agradecer también a la Escuela Doctoral EDSYS por financiar mi doctorado, y finalmente quiero agradecer a quienes hicieron de estos tres años una aventura increíble: a mis amigos en Toulouse y en Medellín, a los que alegraron mis días, a los que se preocuparon por mi y a los que me recordaron de dónde vengo y quién quiero ser.

Remerciements

Je voudrais tout d'abord remercier mes encadrants de thèse, Cédric et Stéphanie, pour leur temps et leur dévouement ainsi que pour leur excellent travail de mentorat pendant les trois années de mon doctorat, et pour leur motivation chaque jour à être plus indépendant dans mon travail. J'ai beaucoup appris de vous, tant sur le plan technique que personnel. Merci !

Je tiens à remercier ma famille. À mes parents pour avoir été un grand soutien émotionnel chaque jour malgré la distance, pour nous avoir aidé à mon frère et à moi à réaliser tous nos rêves, et pour m'avoir transmis le goût d'apprendre (et de résoudre des énigmes et des puzzles mathématiques) depuis toujours. À mon frère pour avoir toujours été là pour moi et pour m'avoir donné sans le savoir la motivation qui me manquait cette année. Sans eux, je n'aurais pas pu réaliser ce rêve !

À mes professeurs, Hugo León Benjumea et Fabio León Restrepo, qui ont été les premiers à me montrer le merveilleux monde de la physique, de la géométrie, des mathématiques..., et qui ont su me transmettre leur passion pour l'ingénierie comme personne d'autre ne l'a fait. À mon professeur Alejandro Montoya qui m'a permis de réorienter ma carrière professionnelle en me faisant découvrir un tout nouveau univers, celui de l'ordonnancement, et qui a eu une grande influence sur le choix de ce doctorat.

À Thomas pour avoir partagé avec moi des moments précieux et tant d'expériences, et à la famille Piekolek pour m'avoir accueilli comme l'une d'entre vous et pour avoir pris le temps de réviser mon manuscrit. À Migue pour être un si bon ami et un modèle pour moi, et à Daniel, Cindy, Alejo, Andres et Liz pour tous les bons moments passés ensemble au déjeuner.

Je tiens également à remercier l'école doctorale EDSYS pour le financement de ma thèse et enfin, je tiens à remercier ceux qui ont fait de ces trois années une incroyable aventure : mes amis à Toulouse et à Medellin, ceux qui ont égayé mes journées et ceux qui m'ont rappelé d'où je viens et qui je veux être.

Acknowledgements

First of all I would like to thank my supervisors Cédric and Stéphanie for their time and dedication and for their excellent work as mentors during the three years of my PhD, pushing me forward and motivating me every day to be more independent in my work. I have learned a lot from them, both on a technical and personal level.

I would like to thank my family: my parents for being a great emotional support every day no matter the distance, for supporting my brother and me in fulfilling all our dreams and for instilling in me a love for learning (and for mathematical enigmas and brainteasers) since I can remember; and my brother for always being there for me and for unknowingly giving me the motivation I lacked this year. Without them I would not have achieved this dream!

To my professors, Hugo León Benjumea and Fabio León Restrepo, who were the first to show me the wonderful universe of physics, geometry, algebra..., and who knew how to transmit their passion for engineering to me in a way that no one else has ever done. To my professor Alejandro Montoya who allowed me to reorient my professional career by showing me the whole new world of scheduling problems, and who had a great influence on the choice of this thesis.

To Thomas for sharing with me valuable time and for teaching me so many important things, and to the Piekolek family for welcoming me into their family and for taking the time to review my manuscript. To Migue for being such a good friend and a role model to me, and to Daniel, Cindy, Alejo, Andres and Liz for the laughs and all the good times over lunch together.

I would also like to thank the EDSYS Doctoral School for funding my PhD, and finally, I want to thank those who made these three years an incredible adventure: my friends in Toulouse and in Medellín, those who brightened my days, those who cared for me and those who reminded me where I come from and who I want to be.

Contents

List of Figures

List of Tables

Table of acronyms

Français

DMR	Déploiement Multi-Robots
POH	Problème d'Ordonnancement Hiérarchique
PPC	Programmation Par Contraintes
RO	Recherche Opérationnelle
ROADEF	Société Française de Recherche Opérationnelle et d'Aide à la Décision

English

ABSTRIPS	Abstraction-Based Stanford Research Institute Problem Solver
ACO	Ant Colony Optimization
AML	ASPEN Modeling Language
ANML	Action Notation Modelling Language
ASPEN	Automated Scheduling and Planning Environment
CBS	Constraint-Based Scheduling
CCSP	Conditional Constraint Satisfaction Problem
CHIMP	Conflict-driven Hierarchical Meta-CSP Planner
CLP	Constraint Logic Programming
COP	Constraint Optimization Problem
CP	Constraint Programming
CSP	Constraint Satisfaction Problem
DAG	Directed Acyclic Graph
DCSP	Dynamic Constraint Satisfaction Problem
EDF	Earliest Deadline First
EUROPA	Extensible Universal Remote Operations Planning Architecture

FAPE	Flexible Acting and Planning Environment
GRASP	Greedy Randomized Adaptive Search Procedure
GSCCB	Guide Search, Check Consistency and Backtrack
GTN	Goal-Task Network
GTOHP	Grounded Totally Ordered Hierarchical Planner
HATP	Hierarchical Agent-based Task Planner
HDDL	Hierarchical Domain Definition Language
HGN	Hierarchical Goal Network
HiPOP	Hierarchical Partial-Order Planning
HS	Hierarchical Scheduling
HSP	Hierarchical Scheduling Problem
HTN	Hierarchical Task Network
IJCAI	International Joint Conferences on Artificial Intelligence
ILP	Integer Linear Programming
ILS	Iterated Local Search
INLP	Integer Non Linear Programming
IP	Integer Programming
IPC	International Planning Competition
IPPS	Integrated Process Planning and Scheduling
JSSP	Job Shop Scheduling Problem
LBBD	Logic-Based Benders Decomposition
LP	Linear Programming
LPT	Largest Processing Time
MDP	Markov Decision Process
MILP	Mixed Integer Linear Programming
MIP	Mixed Integer Programming
MMASP	Multi-Machine Assignment Scheduling Problem
MP	Mathematical Programming

MRD	Multi-Robot Deployment
OR	Operations Research
OSSP	Open Shop Scheduling Problem
OvS	Optimization via Simulation
PANDA	Planning and Acting in Network Decomposition Architecture
PDDL	Planning Domain Definition Language
PLATINUm	PLanning and Acting with TImeliNes under Uncertainty
PERT	Program Evaluation and Review Technique
POCL	Partial-Order Causal-Link
POMDP	Partially Observable Markov Decision Process
PSO	Particle Swarm Optimization
PTSP	Production and Transportation Scheduling Problem
RBF	Radial Basis Function
RCPSP	Resource-Constrained Project Scheduling Problem
RCPSP/max	Generalized Resource-Constrained Project Scheduling Problem
RCPSPR	Resource-Constrained Project Scheduling Problem with Routing
RCPSPTT	Resource-Constrained Project Scheduling Problem with Transfer Times
RCPSPTW	Resource-Constrained Project Scheduling Problem with Time Windows
RSM	Response Surface Methodology
SAT	Boolean (or Propositional) Satisfiability Problem
SBO	Surrogate-Based Optimization
SGS	Schedule Generation Schemes
SHOP	Simple Hierarchical Ordered Planner
SM-DE	Surrogate Model based Differential Evolution
SO	Simulation Optimization
SPT	Shortest Processing Time
STRIPS	Stanford Research Institute Problem Solver
TIHTN	Task Insertion Hierarchical Task Network

TN	Task Network
TSP	Travelling Salesman Problem
VNS	Variable Neighborhood Search
WBS	Work Breakdown Structures

Chapter R

Résumé en français

Sommaire

Introduction

Contexte

Dans de nombreux problèmes de décision rencontrés en pratique, il est très utile et parfois nécessaire de raisonner à différents niveaux d'abstraction. Dans ce contexte, les différentes caractéristiques d'un problème peuvent être données sous la forme de structures hiérarchiques traduisant la décomposition du problème en un ensemble de sous-problèmes. Cette notion ne doit pas être confondue avec la notion de hiérarchie que l'on retrouve dans le cadre de la planification organisationnelle qui est divisée en trois étapes : planification stratégique, planification tactique et planification opérationnelle. Cette dernière forme de hiérarchie concerne davantage l'horizon temporel (court, moyen ou long terme) que la décomposition d'un problème global.

Un premier exemple de problème d'ordonnancement hiérarchique est le problème de Déploiement Multi-Robots (DMR) [115, 143]. Dans un problème de DMR, nous considérons une mission d'exploration de robots impliquant des décisions sur la manière dont les tâches d'exploration (ou d'acquisition) sont partagées entre les robots, sur comment ces tâches sont successivement effectuées par chaque robot, et sur comment les mouvements des robots sont coordonnés de manière à éviter les collisions ou à maintenir les liens de communication. Par exemple, dans ce type de problème, nous nous intéressons à haut niveau **(a)** à la manière dont les différentes tâches d'acquisition à effectuer sont réparties entre les robots disponibles, à un niveau plus bas **(b)** aux déplacements successifs de chaque robot dans un graphe de

points de passage connectés afin d'effectuer les tâches qui lui sont attribuées, et toujours à un niveau plus bas **(c)** à l'évolution fine de la quantité d'énergie disponible pour chaque robot au fur et à mesure du déroulement de la mission. Des Problèmes d'Ordonnancement Hiérarchique (POH) associés aux différents cas d'étude de DMR seront abordés tout au long de ce manuscrit.

D'autres exemples d'applications concernent la gestion d'une constellation de satellites d'observation de la Terre [92, 116], l'implémentation de fonctions sur une architecture embarquée [55, 111], ou encore l'équilibrage de lignes d'assemblage dans l'industrie aéronautique [97, 114]. Un POH basé sur ce dernier cas d'étude sera abordé dans le dernier chapitre de ce manuscrit.

Ces problèmes d'ordonnancement nécessitent de pouvoir gérer différents niveaux de décision concernant l'allocation des tâches aux ressources disponibles et l'ordonnancement des tâches sur ces ressources. Par ailleurs, dans ces problèmes de décision, chaque tâche peut être décomposée en sous-tâches et les contraintes peuvent être modélisées en utilisant différents niveaux d'abstraction. Différentes approches sont souvent considérées pour résoudre ce type de problèmes d'ordonnancement.

- Une première approche commune consiste à décomposer explicitement le problème à résoudre en plusieurs sous-problèmes et à définir pour chacun de ces sous-problèmes une technique de résolution dédiée.
- Une deuxième approche consiste à utiliser des solutions plus génériques, comme c'est le cas pour la planification dans le cadre des réseaux de tâches hiérarchiques (*Hierarchical Task Networks* ou HTN en anglais), où le problème générique considéré consiste à décomposer les tâches de haut niveau en tâches dites atomiques en utilisant un catalogue de méthodes de décomposition des tâches fourni en entrée.

Plan et contributions

L'objectif de cette thèse est de définir des cadres de modélisation et des algorithmes pour traiter des problèmes d'ordonnancement hiérarchique comme ceux mentionnés ci-dessus.

En relation avec les cadres de planification existants, cette thèse cherche à manipuler des représentations hiérarchiques qui sont adaptées aux problèmes d'ordonnancement sous contraintes (*Constraint-Based Scheduling* ou CBS en anglais). Ces représentations sont basées sur des modèles d'ordonnancement "tâches-ressources" définis par un ensemble de tâches candidates et par un ensemble de ressources disponibles pour effectuer ces tâches, et non sur des modèles de planification "états-actions" définis par les paramètres décrivant l'état du système et par les actions disponibles pour changer cet état.

L'approche mentionnée ci-dessus comporte plusieurs avantages. **(1)** Tout d'abord, de nombreux cas réels comme les applications décrites précédemment sont formalisées naturellement comme des problèmes d'ordonnancement "tâches-ressources". **(2)** De plus, le fait de

pouvoir bénéficier de la notion centrale de ressource permet d'envisager des raisonnements transverses plus directs entre les différents niveaux de la hiérarchie décisionnelle. Dans le cadre des HTN, ces interactions sont moins évidentes car elles sont potentiellement cachées derrière les paramètres plus fondamentaux décrivant l'état du système. **(3)** Un autre avantage lors du traitement des tâches hiérarchiques concerne la modularité dans la description des problèmes à résoudre (réutilisation de modèles de tâches existantes), l'expressivité dans les décompositions permises pour les tâches, et enfin la lisibilité des solutions qui peuvent être obtenues.

Dans cette thèse, les techniques de résolution développées s'appuient fortement sur les outils de programmation par contraintes existants et nous essayons de les exploiter au mieux dans nos cadres hiérarchiques. En conséquence, sur les problèmes considérés, nous n'avons pas comparé nos approches à toutes les techniques existantes en Recherche Operationelle (RO).

Le présent manuscrit est organisé comme suit.

Tout d'abord, un état de l'art est présenté au chapitre 1. Cet état de l'art passe en revue les généralités des problèmes d'ordonnancement, notamment plusieurs approches pour les modéliser et les résoudre. Ensuite, il aborde la notion de hiérarchie dans les domaines de la planification et de l'ordonnancement. Les techniques de décomposition des problèmes sont ensuite décrites, pour finalement conclure ce chapitre avec le concept de Problème d'Ordonnancement Hiérarchique (POH).

Un premier cas d'étude relatif à une application hiérarchique de DMR est présenté au chapitre 2. Un aperçu de cette application ainsi que le modèle basé sur contraintes associé sont présentés. De manière à éviter le traitement des situations de collision en ligne, différentes alternatives de modélisation de ce problème de DMR sont étudiées dans ce chapitre. Toutes ces alternatives de modélisation serviront à présenter les méthodes de décision présentées dans les chapitres 3 à 5.

Une première méthode de décision pour traiter les POH est présentée au chapitre 3, ainsi qu'un premier cadre de modélisation pour le traitement des POH. La méthode de décision présentée dans ce chapitre est basée sur l'abstraction de tâches et sur la décomposition itérative de tâches abstraites.

Afin de fournir des cadres plus génériques pour modéliser les POHs, le chapitre 4 introduit un cadre basé sur la décomposition qui permet de considérer une classe plus large de ressources, ainsi que des alternatives d'exécution (méthodes de décomposition) pour certaines tâches comme dans le cadre des HTN.

De nouvelles stratégies de décision itérative sont ensuite présentées dans le chapitre 5. Au lieu de considérer la décomposition itérative de tâches abstraites comme dans le chapitre 3, ces nouvelles stratégies utilisent une approche itérative basée sur la substitution et une approche itérative de génération de coupes. Elles sont toutes deux basées sur le cadre de modélisation introduit précédemment, ce qui permet à nouveau de considerer plus de types de ressources et des alternatives d'exécution pour certaines tâches.

Le chapitre 6 constitue une dernière partie exploratoire de cette thèse et cherche à étudier d'autres applications qui peuvent être traitées comme des POH. Plus précisément, ce chapitre présente un autre cas d'étude concernant l'optimisation de l'équilibrage de lignes d'assemblage en aéronautique, ainsi qu'une première stratégie de décomposition pour traiter ce nouveau cas d'étude.

Enfin, les conclusions des travaux présentés dans cette thèse sont fournies ainsi que les directions de travaux à venir. Les annexes présentent quelques résultats expérimentaux complémentaires sur la comparaison entre les approches présentées.

R.1 Chapitre 1 : État de l'art

Dans ce chapitre, un aperçu très général des problèmes d'ordonnancement est présenté dans la section R.1.1 (section 1.1 de la partie correspondante en anglais). Les sections R.1.2 et R.1.3 (sections 1.2 et 1.3) abordent le point central de ce chapitre, à savoir le concept de hiérarchie dans les problèmes de planification et d'ordonnancement et les techniques de modélisation et de résolution couramment utilisées pour traiter ce type de problèmes. Enfin, la section R.1.4 (section 1.4) présente brièvement le concept de Problèmes d'Ordonnancement Hiérarchique (POH) qui sera abordé dans les autres chapitres avec plus de détails.

R.1.1 Généralités sur les problèmes d'ordonnancement

L'importance de l'ordonnancement a été révélée dans de nombreux domaines et environnements décisionnels, notamment dans différents types d'industries de fabrication et de services. Un problème d'ordonnancement consiste en l'affectation d'un ensemble de ressources à un ensemble de tâches qui doivent être réalisées sur des périodes données et agencées sur les ressources, afin d'optimiser un ou plusieurs objectifs tout en satisfaisant des contraintes spécifiques. Les ressources disponibles pour exécuter les tâches peuvent être associées à des moyens techniques ou humains, tels que des machines dans un atelier ou des opérateurs sur une chaîne de production. Les tâches à exécuter peuvent également prendre différentes formes, selon le type d'application ou d'organisation. Elles sont généralement appelées opérations dans un environnement de production.

Éléments d'un problème d'ordonnancement. De nombreux travaux de recherche sur l'ordonnancement ont été consacrés aux problèmes d'ordonnancement déterministes. Diverses notations ont été introduites au cours des quatre dernières décennies pour couvrir différents types de problèmes [113]. Un cadre de notation capturant la structure d'une grande variété de problèmes d'ordonnancement a été introduit par Graham *et al.* [58]. Plus de précisions sur les éléments d'un Problème d'Ordonnancement sont fournies dans la section 1.1.1.

Quelques exemples de problèmes d'ordonnancement fréquemment abordés (ordonnancement d'atelier, ordonnancement avec des durées de *setup* dépendant de la séquence, problèmes

de gestion de projet avec contraintes de ressources ou RCPSP[1]) sont donnés dans la section 1.1.2.

Complexité. La complexité de la résolution d'un problème d'ordonnancement est liée à sa nature *combinatoire*. La théorie de la complexité, développée pour étudier la complexité des problèmes de calcul, cherche à mesurer cette difficulté intrinsèque en fournissant une preuve mathématique formelle. Un grand nombre de problèmes d'ordonnancement se sont révélés *NP-difficiles* [88, 89], ce qui signifie qu'ils ne peuvent pas être résolus avec un algorithme en temps polynomial dans le pire cas, sauf si $P = NP$. Plus de précisions sur la complexité des problèmes d'ordonnancement sont fournies dans la section 1.1.3.

Modélisation et résolution d'un problème d'ordonnancement. Dans la section 1.1.5 plusieurs approches pour modéliser et résoudre les problèmes d'ordonnancement sont présentées. On peut distinguer deux grandes catégories : les **méthodes exactes** qui ont la capacité théorique de résoudre les problèmes d'ordonnancement de manière optimale, et les **méthodes incomplètes** dont l'objectif est de trouver des solutions de bonne qualité mais sans garantie d'optimalité. Comme indiqué ci-dessus, un grand nombre de problèmes d'ordonnancement se sont révélés *NP-difficiles*. Une partie de la littérature récente est consacrée aux algorithmes incomplets, qui semblent être des méthodes assez appropriées lorsqu'il s'agit de traiter des instances de grande taille avec des temps limités pour générer des solutions de bonne qualité. Dans la section 1.1.5 nous détaillons des stratégies de modélisation et de résolution, notamment la programmation mathématique, la Programmation Par Contraintes (PPC) et les méthodes incomplètes.

R.1.2 Cadres de modélisation pour la planification et l'ordonnancement hiérarchique

Les techniques de planification hiérarchique et d'ordonnancement ont été largement utilisées pour répondre à des applications pratiques de grande taille. Cette section introduit d'abord la notion de hiérarchie en ordonnancement, à travers le concept de *Work Breakdown Structure* (WBS). Cette section décrit ensuite les modèles hiérarchiques utilisés dans le domaine de la planification, et en particulier son cadre le plus représentatif, le cadre des HTN. Enfin, les variantes et les extensions de la planification hiérarchique tirant parti ou impliquant des notions d'ordonnancement sont examinées.

Ordonnancement Hiérarchique. Le concept de *Work Breakdown Structure* (WBS) [61] a été développé comme un cadre commun pour la gestion de projet. On entend par WBS une décomposition hiérarchique d'un *projet*, qui peut correspondre à plusieurs phases ou plusieurs livrables. Un WBS est généralement représenté verticalement, sous la forme d'une structure arborescente dans laquelle les relations parents-enfants sont établies entre les éléments du

[1]Resource-Constrained Project Scheduling Problem

projet, ce qui permet de mieux comprendre et contrôler l'exécution de ce dernier. Un WBS de base ne permet dans un projet qu'une seule alternative (ou méthode de décomposition) pour l'exécution de chacune des tâches du projet. Dans ce cas, chaque tâche peut être directement décomposée avec l'ensemble des sous-tâches qui la composent. Les feuilles de ces structures correspondent à des tâches atomiques qui ne peuvent pas être décomposées. Les nœuds internes qui ont des enfants sont les tâches composites pouvant être décomposées. Les arrangements hiérarchiques et les WBSs peuvent être utilisés pour modéliser la décomposition des tâches entre les différents niveaux d'une hiérarchie [83, 84] comme le montre l'exemple de la figure 1. Cet exemple considère deux tâches obligatoires de niveau supérieur (TLT_1 et TLT_2) décomposées en deux et trois sous-tâches respectivement, et plusieurs contraintes de précédence (flèches) parmi les tâches. Les alternatives d'exécution ou méthodes de décomposition (lignes pointillées) pour certaines des sous-tâches ($ST_{1,1}$ et $ST_{2,2}$) sont intégrées au WBS de cet exemple. La traduction d'un WBS, comme celui de la figure 1 en un modèle de PPC est donnée dans la section 1.2.1.

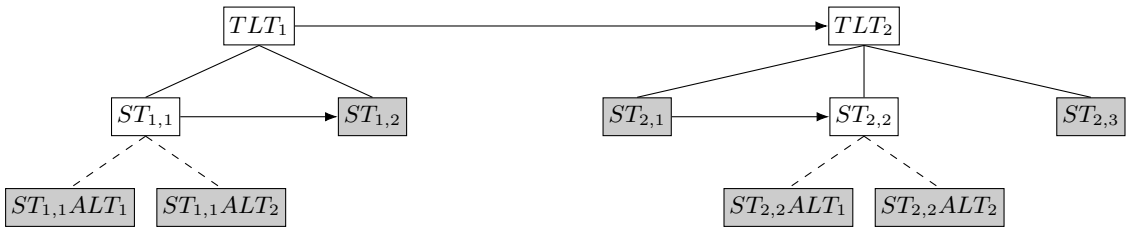

Figure 1: Exemple d'un WBS avec des alternatives pour certaines tâches.

Planification Hiérarchique. La planification automatisée (*Automated Planning* ou AI Planning en anglais) est un domaine de l'intelligence artificielle qui étudie les processus de prise de décision computationnelle effectués par des entités intelligentes (robots, humains, programmes informatiques) pour atteindre certains objectifs le plus efficacement possible [53, 54]. La planification comme processus de prise de décision considère un modèle conceptuel général d'un système dynamique, appelé système *état-transition* ou système à *événements discrets*. Contrairement à un problème d'**ordonnancement**, où l'ensemble des tâches est initialement connu, un problème de **planification** considère plutôt un ensemble d'*actions* ayant un impact sur l'évolution des états d'un système.

Pour passer d'un état initial à un état final, on dispose de *fonctions de transition d'état* qui consistent en un ensemble d'*opérateurs* de planification qui sont instanciés en *actions*. La planification HTN est une représentation alternative des problèmes de planification, dans laquelle l'objectif n'est pas d'atteindre un état objectif, mais de réaliser un ensemble de tâches. Chaque tâche de l'ensemble donné est soit une tâche primitive, exécutée par une action primitive, soit une tâche non primitive, décomposée en un ensemble d'autres tâches. Plus de détails sur la formalisation d'un problème de planification HTN sont donnés dans la section 1.2.2.

Extensions du cadre standard de la planification HTN pour prendre en compte des caractéristiques d'ordonnancement. L'un des *grands inconvénients* du cadre standard de la planification classique et HTN en pratique est lié aux concepts implicites essentiels de *temps et ressources* [131]. La représentation des applications du monde réel peut s'avérer assez complexe sans ces notions centrales, car les ressources disponibles ne peuvent pas être directement associées à l'exécution des tâches et certaines interactions temporelles peuvent être difficiles à exprimer. Ces notions centrales sont explicites dans l'ordonnancement, et permettent de raisonner directement sur les contraintes temporelles et les consommations de ressources par les différentes tâches. Plus de détails sur les extensions générales et sur le temps et les ressources dans la planification HTN sont donnés dans la section 1.2.3.

R.1.3 Ordonnancement multi-étages par décomposition des problèmes

Revenant sur le domaine des problèmes d'ordonnancement, cette section aborde les techniques de décomposition des problèmes rencontrées dans les travaux récents qui offrent l'avantage de ne pas représenter un problème d'ordonnancement dans sa totalité.

Les techniques de décomposition des problèmes sont apparues comme des approches utiles lorsqu'il s'agit de résoudre des problèmes d'ordonnancement coûteux en termes de calcul. Au lieu de modéliser et de résoudre un problème d'ordonnancement dans son intégralité, une approche de décomposition le divise en plusieurs *étapes d'optimisation* cherchant à diminuer la complexité combinatoire. Dans un tel schéma décomposé, les étapes d'optimisation peuvent représenter différents niveaux de décision hiérarchique, par exemple deux niveaux traitant respectivement de la répartition des tâches et de l'ordonnancement sur les ressources, ou simplement un ensemble de sous-problèmes *plus faciles*.

Décomposition de problèmes et programmation mathématique. La décomposition de Benders [16, 51] est une technique qui reformule un problème d'optimisation en deux étapes. La première étape considère un premier sous-ensemble de variables pour résoudre un *problème maître* et la seconde étape considère un *sous-problème* (ou problème auxiliaire) qui implique un deuxième sous-ensemble de variables et qui est résolu pour chacune des solutions de la première étape. Ce processus cherche à générer de façon itérative de nouvelles contraintes appelées coupes de Benders (*Benders cuts* en anglais) pour le problème principal. Ces coupes élaguent les décisions de la première étape qui sont irréalisables (coupes de faisabilité) ou sous-optimales (coupes d'optimalité). Dans la section 1.3.1, nous détaillons les algorithmes de décomposition de Benders classiques, ainsi que les algorithmes de génération de colonnes qui sont également basés sur une décomposition du problème en un problème maître et un sous-problème et qui permet d'ajouter des variables au problème maître.

Décomposition en problèmes de satisfaction des contraintes (*Constraint Satisfaction Problems* ou CSPs) dynamiques. Un CSP dynamique [37] est une séquence de CSP (section 1.1.5), chacun d'eux étant une transformation du problème précédent, liée

à des changements de contraintes ou de domaines, ou à des ajouts ou suppressions de variables. Tous ces changements peuvent être exprimés en termes d'ajouts ou de suppressions de contraintes (assouplissements) [148, 149] sur les différents problèmes. Les CSPs dynamiques peuvent être considérés comme une sorte de décomposition de problème impliquant plusieurs sous-problèmes à résoudre en séquence. Plus de détails sont donnés dans la section 1.3.2.

Décomposition des problèmes et optimisation hybride. Des approches à deux niveaux pour résoudre les problèmes de planification et d'ordonnancement intégrés (*Integrated Process Planning and Scheduling* ou IPPS) ont été largement utilisées dans le domaine des systèmes de fabrication et de la planification de la production. La décomposition de Benders basée sur la logique (*Logic-Based Benders Decomposition* ou LBBD) est une généralisation de l'approche de la décomposition de Benders classique qui permet d'appliquer l'approche classique à une classe plus large de problèmes. Elle permet par exemple d'hybrider programmation mathématique et programmation par contraintes. Plus de détails sont présentés dans la section 1.3.3.

Optimisation basée sur la simulation (*Simulation-Based Optimization* or SBO). L'optimisation par simulation (*Simulation Optimization* ou SO) ou l'optimisation par substitution (*Surrogate-Based Optimization* ou SBO) est une technique utilisée pour aborder des problèmes de grande taille coûteux en termes de calcul et provenant de divers domaines [6]. Elle est également connue sous le nom d'optimisation basée sur les métamodèles, d'optimisation par boîte noire, d'optimisation paramétrique ou d'optimisation par simulation. Contrairement à la programmation mathématique, ces techniques ne reposent pas sur un *modèle mathématique*, mais considèrent une sorte de *modèle de simulation* disponible comme une boîte noire [74, 76]. L'optimisation basée sur la simulation peut être considérée comme une stratégie de décomposition des problèmes dans laquelle une couche supérieure (un problème maître) peut incorporer un modèle de substitution d'une ou plusieurs couches en aval (ou sous-problèmes). Les sous-problèmes plus fins peuvent être utilisés pour évaluer des fonctions objectifs basées sur un modèle à fin grain, et ils peuvent fournir au problème de substitution principal des informations pertinentes pour ajuster la simulation après chaque évaluation de la fonction objectif pour une entrée particulière. Plus de détails sont donnés dans la section 1.3.4.

R.1.4 Problèmes d'ordonnancement hiérarchiques (POH)

Un POH peut être défini en réutilisant les concepts précédemment introduits de planification HTN et WBS. À chaque niveau de la structure hiérarchique décrivant un POH, un groupe de *sous-tâches* à effectuer peut être *abstrait* en fonction de la granularité du système (détail des discriminations sur le système) [125, 155]. Par exemple, un problème de Job Shop classique peut être considéré comme un simple problème d'ordonnancement hiérarchique à deux niveaux, dans lequel les tâches à exécuter (tâches de niveau supérieur) sont décomposées en ensembles d'opérations (sous-tâches) à effectuer sur les machines disponibles. Plus de détails

sur ce point sont donnés dans le chapitre 3 dans lequel un premier cadre de modélisation d'un POH est introduit (section 3.2.1).

Dans sa structure hiérarchique, un POH peut également comporter plusieurs alternatives d'exécution, ou méthodes de décomposition, pour l'exécution des tâches. Ainsi, en tant que structure de planification HTN, un POH peut être considéré comme une structure *plus flexible* qui, contrairement à un WBS, n'est pas entièrement dépliée. On trouve plus de détails sur ce point dans les chapitres 4 et 5.

R.2 Chapitre 2 : Un cas d'étude hiérarchique de déploiement multi-robots

R.2.1 Introduction

Dans ce chapitre, un cas d'étude concernant une application de déploiement multi-robots (DMR) est présenté. Un Problème d'Ordonnancement Hiérarchique (POH) basé sur différents formats de cette application de DMR a été étudié au cours de cette thèse. Ce problème fait intervenir *un ensemble de robots* qui doivent *naviguer à travers les zones partagées* d'un terrain et effectuer *des tâches d'exploration* (ou *observations*) à différents endroits.

L'application de DMR a été largement utilisée pour résoudre des problèmes comme la détection coopérative à l'aide d'équipes air-sol [30], la réponse aux catastrophes [122], et les systèmes d'exploration/sauvetage dans des environnements hostiles [139]. Dans le domaine de la RO, des applications similaires ont été abordées. C'est le cas des problèmes d'ordonnancement et de routage intégrés [151], nécessitant la coordination des activités de production et des activités de transport (le *Production and Transportation Scheduling Problem* ou PTSP). Parmi les problèmes similaires figurent les extensions du RCPSP classique qui peuvent impliquer des temps de transfert (RCPSPTT) [119] ou du routage (RCPSPR) [151]. D'autres applications provenant de différents domaines pourraient également être adaptées et traitées comme des sortes de problèmes de DMR.

Ce chapitre est organisé comme suit. La section R.2.2 (section 2.2 de la partie correspondante en anglais) présente un aperçu de l'application de DMR, notamment une description détaillée des cas d'études auquels nous nous intéressons. La section R.2.3 (section 2.3) présente le Problème d'Ordonnancement Hiérarchique associé ainsi qu'un encodage en PPC. La section R.2.4 (section 2.4) détaille plusieurs mécanismes anti-collision pour éviter les conflits entre robots lors de l'exécution de la mission de DMR. Enfin, la section R.2.5 (section 2.5) conclut ce chapitre.

R.2.2 Aperçu de l'application de déploiement multi-robots

Dans un problème de DMR, un **ensemble de robots** doit effectuer un ensemble de **demandes d'observation** sur des zones spécifiques d'un terrain. Les robots ne peuvent pas effectuer plus d'une observation à la fois (ressources disjonctives). Ils doivent également **transférer les données d'observation en temps réel** à un centre de mission, et à cette fin chaque robot utilise une **fréquence d'émission** spécifique liée à la nature de l'observation effectuée. Pour éviter les interférences, deux robots qui utilisent la même fréquence ne peuvent pas transférer des données d'observation en parallèle (ressource disjonctive). La **redondance** est également utile dans ce type d'application, c'est pourquoi certaines cibles (ou demandes) d'observation doivent être observées par plusieurs robots distincts et doivent être espacées d'un certain temps (**délai de revisite**). De plus, certaines **contraintes de précédence** peuvent être imposées sur les observations à effectuer.

Pour représenter la structure du terrain qui est **partagé entre tous les robots**, un graphe de points de passage connectés comme celui de la figure 2 est utilisé. Dans cette figure, deux **observations atomiques** (O_1, O_2) doivent être effectuées sur la **cible d'observation (zone ou demande)** RQ_1, deux **observations atomiques** (O_4, O_5) sur la cible d'observation RQ_3, et une seule observation (O_3) sur la cible RQ_2.

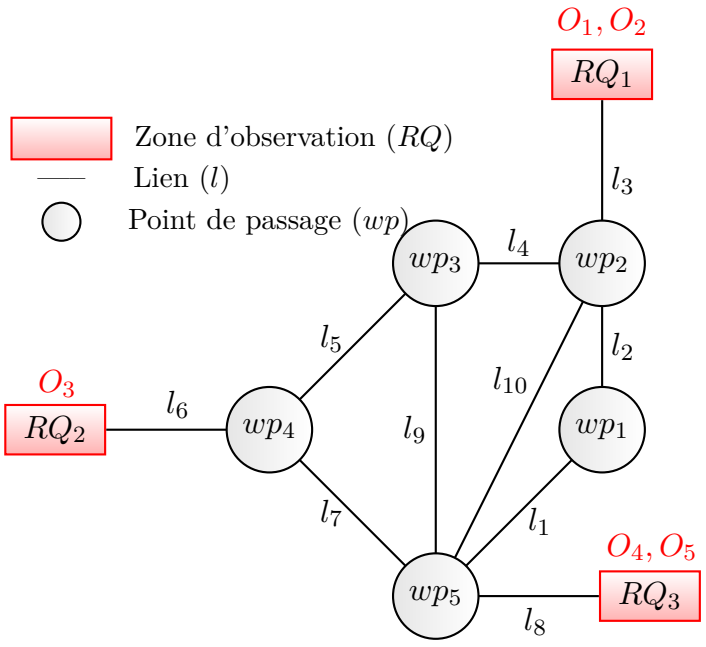

Figure 2: Schéma des points de passage connectés pour représenter la structure du terrain.

Dans le problème de DMR, **plusieurs chemins candidats** sont considérés pour naviguer entre des paires de zones d'observation. Pour chaque trajectoire alternative, les mouvements des robots entre les lieux d'observation peuvent être décomposés en mouvements successifs sur des **liens** entre des paires de **points de passage** adjacents. Dans un premier mécanisme anti-collision *simple*, pour éviter les conflits entre les robots dans le réseau partagé, chaque lien

et chaque point de passage ne peuvent être occupés par plus d'un robot à la fois (ressources disjonctives). Enfin, l'objectif de la mission est d'effectuer **toutes les observations le plus rapidement possible**.

Afin de tester les différentes stratégies qui ont été étudiées tout au long de cette thèse, des *benchmarks* autour de l'application de DMR sont développés dans les chapitres 3 et 4. Un premier générateur d'instances du problème a été défini pour ce type de benchmark (voir la section 2.2.2 pour plus de détails).

R.2.3 Ordonnancement basé contraintes pour l'application de DMR

Le but du problème de DMR est d'**allouer chaque observation candidate** à un robot, de **fixer la séquence d'observations** réalisée par chaque robot, et de **définir les tâches de navigation** entre les lieux d'observation. Une première approche possible pour résoudre ce problème consiste à développer un modèle de programmation par contraintes couvrant toutes les spécifications de la mission. Les caractéristiques de ce modèle, comprenant les données d'entrée, les variables de décision, les contraintes considérées et la fonction objectif sont décrites dans la section 2.3.

R.2.4 Mécanismes anti-collision

Différentes alternatives pour la modélisation du problème de DMR sont envisagées, en fonction des ressources du réseau qui peuvent être partagées et en fonction de la durée pendant laquelle chaque mouvement de robot monopolise ces ressources partagées. Les approches diffèrent en termes de robustesse et de synchronisation requise entre les robots au moment de l'exécution, moment où la durée des mouvements du robot peut être plus courte ou plus longue que prévu.

1. Le cas le plus simple : liens non partageables. Dans cette première approche anti-collision, les emplacements associés aux zones d'observation et les points de passage sont considérés comme partageables. La raison de cette hypothèse est que les robots ont une vitesse plus faible (voire nulle) lorsqu'ils effectuent des observations ou tournent (respectivement) à ces endroits, de sorte que la gestion en ligne des collisions est plus facile et moins dangereuse dans ce cas que lorsque le robot se déplace sur des liens, qui ne sont pas partageables.

2. Liens et points de passage non partageables et transfert (*handover*) minimum. Dans cette deuxième approche, les seuls endroits considérés comme partageables sont ceux associés aux zones d'observation, compte tenu, comme dans l'approche précédente, de la facilité de gestion des collisions en ligne. Cette deuxième approche est illustrée dans la figure 3 ou une durée de *handover* ou de transfert entre les ressources est considerée. Plus de détails sur cette méthode anticollision sont disponibles dans la section 2.4.2.

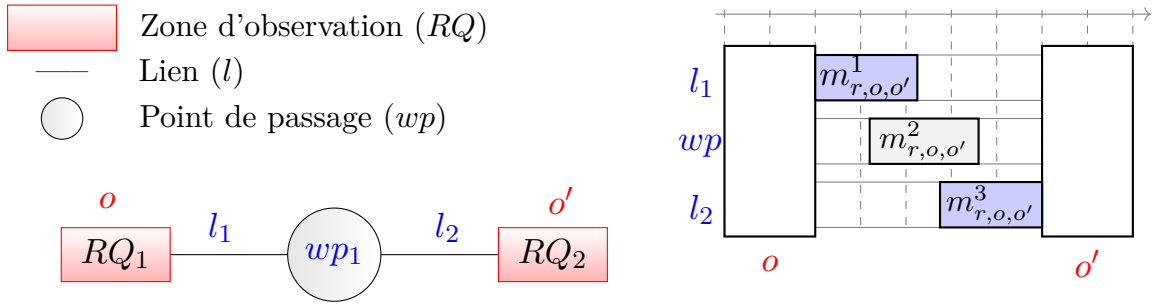

Figure 3: Consommation des ressources du réseau pour le cas "transfert minimum".

3. Chemins non partageables. Dans cette troisième approche, pour une transition d'un robot r entre deux observations o et o', toutes les ressources de chemin utilisées pendant la transition sont réservées pour toute la durée du déplacement, comme l'illustre la figure 4. Plus de détails sur cette méthode anti-collision sont disponibles dans la section 2.4.3.

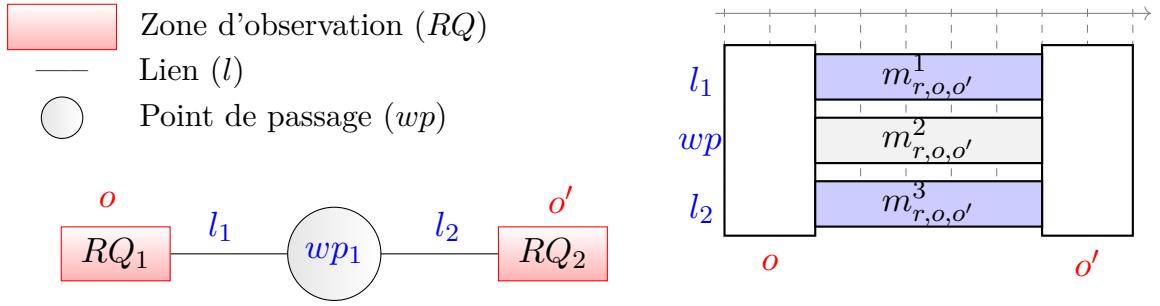

Figure 4: Consommation des ressources du réseau pour le cas "chemins non partageables".

R.2.5 Conclusions

Ce chapitre a présenté un cas d'étude de DMR dans lequel un ensemble de robots doit naviguer dans des zones communes d'un terrain et effectuer des tâches d'exploration (ou des observations) à différents endroits le plus rapidement possible. Le cas d'étude général a été présenté, ainsi que le problème d'ordonnancement hiérarchique qui lui est associé et un encodage en programmation par contraintes. Enfin, plusieurs représentations alternatives pour le problème de DMR comprenant des mécanismes anti-collision ont été introduites. Ces différentes approches sont examinées plus en détail dans les chapitres suivants.

R.3 Chapitre 3 : Une première méthode de décision basée sur l'abstraction des tâches et la décomposition itérative des tâches

R.3.1 Introduction

Ce chapitre vise à fournir un premier cadre pour modéliser certaines classes de problèmes d'ordonnancement hiérarchique, ainsi que des stratégies préliminaires pour résoudre ces problèmes. Ce nouveau cadre réutilise certaines notions des concepts de planification WBS et HTN décrits précédemment, notamment les tâches composites qui couvrent plusieurs sous-tâches. Il est présenté dans la section R.3.2 (section 3.2 de la partie correspondante en anglais) avec la définition formelle du POH considéré dans ce chapitre, et une première traduction de ce cadre en programmation par contraintes.

Pour traiter les POHs qui peuvent être modélisés dans ce cadre, une première *méthode de décision* basée sur l'abstraction et la décomposition des tâches du problème est introduite dans les sections R.3.3 et R.3.4 (sections 3.3 et 3.4), ainsi que plusieurs règles heuristiques et paramètres de configuration associés. Les résultats expérimentaux obtenus grâce à la mise en œuvre de la stratégie de raffinement proposée pour différents cas représentatifs des intances du problème, notamment des références bien connues et des cas d'application de DMR, sont examinés dans la section R.3.5 (section 3.5). Enfin, la section R.3.6 (section 3.6) présente les conclusions des travaux présentés dans ce chapitre et donne quelques pistes de recherches connexes ultérieures.

Les contributions introduites dans ce chapitre ont été présentées au début de la thèse au "Journées Francophones de Programmation par Contraintes (JFPC)" [106].

R.3.2 Problèmes d'ordonnancement hiérarchique

Dans cette section, nous présentons un premier cadre de modélisation pour traiter les problèmes d'ordonnancement hiérarchique définis par **(1)** un ensemble de tâches qui peuvent être atomiques (*i.e.* directement réalisables) ou composites (*i.e.* décomposées en un sous-ensemble de tâches), **(2)** un ensemble de ressources disjonctives disponibles pour les exécuter, et **(3)** la minimisation du makespan tout en tenant compte des contraintes temporelles et des contraintes de disponibilité des ressources requises.

Un problème d'ordonnancement hiérarchique P est défini comme un tuple $(\mathcal{R}, U)$, où $\mathcal{R}$ est un ensemble de ressources disjonctives qui ne peuvent effectuer qu'une seule activité à la fois, et U est un ensemble de tâches atomiques et composites.

Plus de détails sur ce premier cadre de modélisation ainsi qu'une illustration sur un exemple de DMR sont donnés dans la section 3.2.

L'encodage en programmation par contraintes du POH défini est relativement simple. Il

est possible de réutiliser les primitives d'ordonnancement disponibles dans l'outil CP Optimizer d'IBM ILOG, dont certaines peuvent être utilisées pour définir des hiérarchies de tâches [83, 84]. Le modèle ainsi obtenu est présenté dans la section 3.2.2.

R.3.3 Abstraction d'une tâche composite

La méthode PPC présentée dans la section précédente est basée sur une **version complètement dépliée** du problème d'ordonnancement hiérarchique. Un tel dépliage complet peut être assez **coûteux en termes de temps de calcul** au fur et à mesure que le nombre de tâches atomiques et composites augmente. Pour faciliter le passage à l'échelle et guider la résolution à plus haut niveau, une méthode permettant d'éviter le dépliage systématique de toutes les tâches au début de la recherche est proposée. Cette méthode raisonne sur des *abstractions de tâches composites* dans un premier temps, puis *raffine* ces abstractions étape par étape lorsque cela est nécessaire, en revenant progressivement à une modélisation non abstraite pour les tâches composites.

Dans ce contexte, *abstraire* une tâche composite signifie représenter, de manière non détaillée, toutes ses sous-tâches, et raisonner en **estimant l'impact global de ces sous-tâches** sur le problème d'ordonnancement à résoudre. Plus précisément, chaque tâche composite c est abstraite par une tâche atomique notée par $Abs(c)$. Cette dernière est définie par une durée $du_{Abs(c)}$ et par un ensemble de ressources $\mathcal{R}_{Abs(c)}$ consommées pendant toute la durée de $Abs(c)$. Les techniques utilisées pour définir les particularités d'abstraction $du_{Abs(c)}$ et $\mathcal{R}_{Abs(c)}$ sont définies en détail dans la section 3.3, qui introduit d'un côté des abstractions dites optimistes et de l'autre des abstractions dites pessimistes.

R.3.4 Stratégie de décomposition itérative (raffinement)

Une stratégie globale de décomposition ou de raffinement est utilisée pour passer progressivement d'un plan construit pour un problème P_0 qui ne contient que des abstractions de tâches de haut niveau, à un plan construit pour un problème P_n qui correspond au problème $(\mathcal{R}, U)$ original. Pour passer d'un problème P_i à un problème raffiné P_{i+1}, le principe de base est de sélectionner à chaque étape de l'algorithme un ensemble non vide de tâches composites qui sont présentes dans P_i sous leur forme abstraite, et de raffiner ces tâches de sorte que le nouveau problème P_{i+1} obtenu contienne au minimum une tâche abstraite de moins.

Différents paramètres doivent être établis pour définir exactement comment passer du problème P_i au problème P_{i+1} :

- **Nombre de tâches à raffiner** : ce premier réglage à effectuer concerne le nombre de tâches abstraites qui sont raffinées à chaque étape.
- **Heuristique pour la sélection des tâches à raffiner** : concernant le choix d'une règle heuristique qui favorise le raffinement des tâches abstraites les plus intéressantes,

trois heuristiques de sélection ont été étudiées.

- **Informations transmises entre les itérations** : il est utile de définir des méthodes pour exploiter les premières résolutions afin de guider les résolutions ultérieures (contraintes de type *start-to-start*, par exemple), car ces dernières peuvent être plus facilement coincées dans certaines zones d'un espace de recherche potentiellement beaucoup plus vaste au fur et à mesure que le processus de raffinement est exécuté.

Les paramètres permettant de raffiner un problème d'ordonnancement hiérarchique et l'algorithme de décomposition itérative sont présentés dans la section 3.4.

R.3.5 Résultats expérimentaux

Benchmarks. Afin de tester les différentes stratégies, nous avons généré des benchmarks autour de l'application de déploiement multi-robots décrite précédemment. Nous avons également testé les méthodes sur des références bien connues de la littérature, comme les problèmes d'Open Shop et de Job Shop [141]. Les générateurs d'instances de problèmes développés pour ce type de benchmarks sont décrits dans la section 3.5.

Résultats expérimentaux. Les résultats obtenus révèlent que l'outil d'optimisation CP Optimizer travaillant directement sur un seul problème global parvient à trouver de bonnes solutions assez rapidement pour toutes les instances du problème et qu'il est plus performant que notre algorithme de décomposition itérative. Pour les petits problèmes, notre méthode est capable de trouver le *makespan* optimal trouvé par CP Optimizer. En revanche, sur des problèmes plus larges, en ajoutant des contraintes de précédence *start-to-start* (voir la section 3.4) entre des résolutions successives, l'espace de recherche est restreint et il existe un risque de retirer potentiellement la solution optimale.

R.3.6 Conclusions

Dans ce chapitre, une première méthode de décision pour traiter les POH a été présentée, ainsi qu'un nouveau cadre pour modéliser certaines classes de ces problèmes. Pour réaliser cette première étude, le présent chapitre ne traite que de classes relativement simples de POH. Une interprétation simple de ce cadre en programmation par contraintes a également été présentée. L'approche globale tente d'utiliser la hiérarchie de décomposition des tâches pour guider la recherche. La hiérarchie de décomposition est donc exploitée au niveau *algorithmique*, et *pas seulement à des fins de modélisation.*

Les nouvelles formes de la méthode introduite pourraient être considérablement améliorées dans le cadre de recherches ultérieures en considérant des stratégies d'abstraction plus fines, par exemple, des abstractions utilisant des ressources cumulatives. Les informations trans-

férées entre les itérations pourraient également être réévaluées afin que l'espace de recherche ne soit pas réduit de manière significative.

Dans le chapitre 5, plus de stratégies de décision itératives sont introduites. Ces nouvelles stratégies utilisent une *stratégie itérative de substitution* et une *approche itérative de génération de coupes* permettant de considérer une classe plus large de ressources et des alternatives d'exécution (méthodes de décomposition) pour certaines tâches comme dans le cadre des HTN.

R.4 Chapitre 4 : Modèles génériques pour les POHs

R.4.1 Introduction

Ce chapitre vise à fournir un cadre générique basé sur la décomposition pour modéliser les problèmes d'ordonnancement hiérarchique. Dans le chapitre précédent, nous avons étudié une approche basée sur la décomposition partielle des abstractions de tâches. Dans ce chapitre, le cadre proposé est toujours basé sur la décomposition mais permet d'envisager une classe plus large de ressources, ainsi que des alternatives d'exécution (méthodes de décomposition) pour certaines tâches, comme dans le cadre des HTN. Les POH considérés impliquent des ressources qui doivent effectuer des opérations de préparation (ou *setup*) complexes entre les tâches principales qu'elles réalisent. Pour ces problèmes, notre objectif est de calculer les dates d'exécution optimales des tâches, en tenant compte des contraintes temporelles et des contraintes de disponibilité des ressources.

Certaines contributions de ce chapitre ont été présentées en 2019 lors de la conférence internationale sur l'intelligence artificielle, IJCAI[2] [107] et à la conférence internationale sur les principes et la pratique de la programmation par contraintes, CP[3] [108]. De courts résumés des principales contributions ont également été présentés en 2019 et 2020 lors de la conférence annuelle ROADEF[4] [109, 110].

Ce chapitre est organisé comme suit. La section R.4.2 (section 4.2 dans la partie correspondante en anglais) décrit les problèmes d'ordonnancement hiérarchique prenant en compte des durées de *setup* et la section R.4.3 (section 4.3) introduit une première approche spécifique de la décomposition des problèmes d'ordonnancement illustrée sur l'application de DMR. Les sections R.4.4 et R.4.5 (sections 4.4 et 4.5) présentent des modèles génériques pour les POHs, notamment un modèle global (un cadre à une seule couche) et un modèle basé sur la décomposition (un cadre à plusieurs couches). La section R.4.6 (section 4.6) illustre le cadre de décomposition générique pour le cas d'application du DMR. Ces cadres génériques peuvent être étendus à plusieurs problèmes d'ordonnancement impliquant des opérations de *setup*, et couvrent toutes les approches anti-collision présentées dans le chapitre 2. Enfin, la section R.4.7 (section 4.7) conclut sur les principales contributions présentées dans ce chapitre.

[2]International Joint Conferences on Artificial Intelligence
[3]International Conference on Principles and Practice of Constraint Programming
[4]Société Française de Recherche Opérationnelle et d'Aide à la Décision

R.4.2 Problèmes d'ordonnancement hiérarchique impliquant des opérations de setup

Les problèmes d'ordonnancement hiérarchique étudiés dans ce chapitre impliquent des ressources avec durées de *setup* (préparation) entre les tâches qu'elles doivent réaliser. Comme mentionné au chapitre 1, une durée de *setup* entre deux tâches j et k représente le temps qui doit s'écouler entre la fin de la tâche j et le début de la tâche k si k suit immédiatement j sur une ressource donnée [124].

Les durées de *setup* peuvent également être utilisées pour représenter des abstractions d'opérations de *setup* potentiellement complexes. Par exemple, dans l'application de DMR (voir chapitre 2), une opération de *setup* qui nécessite qu'un robot se rende d'un point de passage A à un point de passage B peut être approchée par une durée de *setup* constante obtenue par un simple calcul du chemin le plus court. Dans la pratique cependant, ces opérations de *setup* correspondent à des mouvements réels des robots, et le fait que le réseau de liens entre les points de passage soit **une ressource partagée entre les robots** doit être pris en compte pour évaluer l'efficacité réelle d'un plan (voir figure 5). Dans ce cas, à un niveau détaillé, il peut y avoir plusieurs chemins de navigation candidats pour se déplacer entre A et B, et chaque alternative de chemin correspond à un ensemble de déplacements sur les liens du graphe de points de passage.

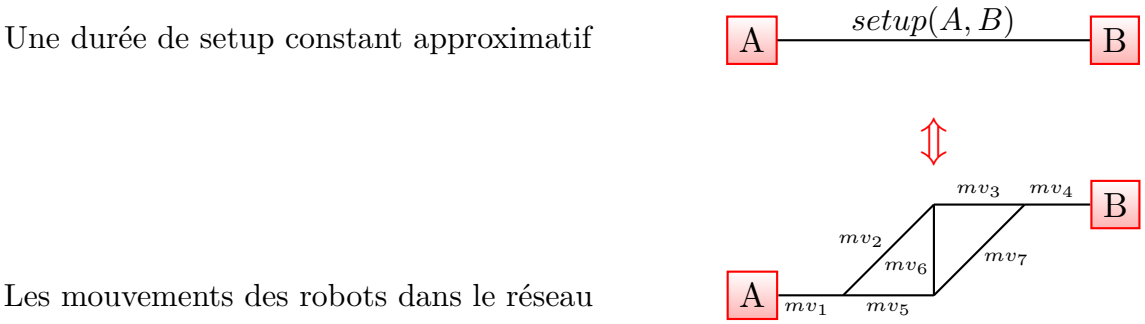

Figure 5: Exemple d'approximation des durées de *setup* dans une application de DMR.

D'autres exemples d'applications réelles impliquant des ressources avec des durées de *setup* sont décrits dans la section 4.2. L'objectif poursuivi dans ce chapitre est de proposer un cadre générique pour traiter de tels problèmes d'ordonnancement hiérarchique impliquant des opérations de *setup* complexes.

R.4.3 Décomposition en un problème d'ordonnancement à deux niveaux : un exemple

Pour illustrer l'approche basée sur la décomposition proposée dans ce chapitre, nous repartons de l'application de DMR. Pour traiter cette application, une approche globale telle que celle présentée dans le chapitre 2 doit traiter un grand (parfois énorme) nombre de tâches.

Pour réduire la complexité du calcul, une approche classique consiste à décomposer explicitement le problème en plusieurs sous-problèmes. Dans notre cas, le problème de DMR peut être divisé en deux parties : **(1)** une partie qui décide des observations successives réalisées par chaque robot sur la base d'un modèle à gros grain des opérations de navigation (appelée **couche L1**), et **(2)** une partie chargée de détailler les navigations des robots au sein du réseau de points de passage et de liens partagés (appelée **couche L2**). Ces sous-problèmes peuvent être intégrés par exemple dans une **stratégie de décision à deux niveaux** qui synthétise d'abord un ordonnancement de haut niveau basé sur un **modèle à gros grain** des opérations de *setup* (couche de décision L1), puis détaille cet ordonnancement basé sur un **modèle à grain fin** (couche de décision L2).

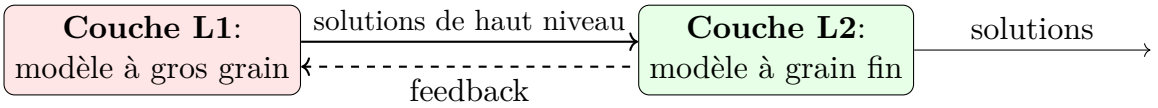

Figure 6: Décomposition du modèle en deux couches de décision.

L'avantage de cette approche est que la couche de décision **L2 ne doit prendre en compte que les opérations de *setup* qui sont effectivement utilisées dans la solution à gros grain** produite par la couche de décision L1.

Des approches *top-down* sont couramment utilisées en pratique pour la prise de décision hiérarchique, mais comme les décisions de haut niveau sont calculées à partir d'un modèle à gros grain, elles peuvent ne pas aboutir à des solutions de la plus haute qualité. C'est pourquoi on peut utiliser des stratégies de décision hiérarchique qui utilisent itérativement les deux couches d'ordonnancement (avec un retour d'information ou *feedback* de la couche de décision L2 comme illustré dans la figure 6) pour traiter des opérations de *setup* complexes. C'est le cas des approches présentées dans les deux chapitres suivants.

Modèle d'ordonnancement à gros grain : Couche L1. Dans le modèle d'ordonnancement de haut niveau de la couche L1, la navigation entre deux observations données o et o' est abstraite de manière très grossière comme une simple durée de *setup* entière requise entre la fin de la réalisation de o et le début de la réalisation de o'. Le modèle en PPC pour la couche L1 se trouve dans la section 4.3.2.

Modèle d'ordonnancement à grain fin : Couche L2. Le modèle d'ordonnancement de bas niveau de la couche L2 prend en considération tous les chemins de navigation disponibles dans le graphe de points de passage pour détailler le routage des robots dans le réseau partagé et gérer les conflits de navigation. Cette couche de décision ne prend en compte que les trajets de navigation entre les tâches d'observation présentes dans la solution à gros grain σ^1 produite par L1 (beaucoup moins d'options de navigation par rapport au modèle PPC global). Le modèle PPC pour la couche L2 se trouve dans la section 4.3.3.

Vers une approche générique. Les modèles précédents (donnés dans les sections 4.3.2 et 4.3.3) correspondent à une décomposition "manuelle" du problème d'ordonnancement associé à l'application de DMR. Dans les deux sections suivantes, des mécanismes plus génériques sont introduits. Deux points spécifiques sont abordés, à savoir **(1)** comment définir un cadre générique pour avoir une modélisation compacte pour chaque couche, et **(2)** comment obtenir un cadre générique pour gérer l'interaction entre les couches.

R.4.4 Un cadre générique à une seule couche pour modéliser les problèmes d'ordonnancement hiérarchique

Dans cette section, un cadre de modélisation *générique* pour traiter les POH avec des durées de *setup* est présenté. Ce cadre générique concerne le modèle hiérarchique d'un problème d'ordonnancement au sein d'une seule couche. Ce modèle généralise le cadre présenté précédemment dans le chapitre 3, d'abord en considérant les ressources disjonctives avec des durées de *setup*, et ensuite en considérant plusieurs décompositions possibles pour les tâches composites. Ce cadre générique à une seule couche, ainsi que l'encodage correspondante en programmation par contraintes se trouvent dans la section 4.4.

Le cadre générique est basé sur la notion de réseau de tâches de la forme (U, C) dans lequel U est un ensemble de tâches et C un ensemble de contraintes temporelles. Parmi les tâches de U, on trouve des tâches atomiques et des tâches composites. A chaque tâche composite est associé un ensemble de méthodes de décomposition (ou alternatives) qui sont elles-mêmes représentées sous la forme d'un réseau de tâches. Les tâches atomiques ne peuvent pas être décomposées. Elles ont une durée et peuvent consommer des ressources. Les ressources sont elles-mêmes divisées en deux catégories : des ressources disjonctives classiques et des ressources disjonctives avec setup qui seront raffinées au niveau hiérarchique suivant. Une traduction de ce cadre générique en PPC est également donnée.

R.4.5 Un cadre générique multi-couches pour modéliser les POHs

Notre objectif est de considérer deux couches d'ordonnancement L1 et L2 qui représentent des opérations de *setup* à des niveaux d'abstraction différents. Pour modéliser chacune des couches de décision individuellement, nous considérons deux modèles de POH comme décrit précédemment. Pour faire interagir ces deux couches, il faut indiquer la relation formelle entre les problèmes d'ordonnancement auxquels elles s'attaquent. Pour cela, un *système générique* est défini pour générer automatiquement un problème P_2 pour L2 à partir d'une solution σ_1 trouvée pour un problème P_1 pour L1. Plus de détails sur ce schéma générique multi-couche sont donnés dans la section 4.5.

Plus précisément, les ressources avec durée de setup de la couche L1 sont remplacées dans la couche L2 par un ensemble de ressources qui les raffinent. Les tâches atomiques de la couche L1 qui sont présentes dans la solution σ_1 et qui consomment ces ressources avec durée de setup sont transformées dans la couche L2 en tâches composites. Les méthodes de décomposition

associées à ces tâches composites conduissent à des sous-tâches qui consomment les ressources raffinées. Dans le modèle de la couche L2, sont également ajoutées des contraintes de présence de tâches et des contraintes de précédence de manière à garantir la cohérence entre la solution σ_1 envoyée par la couche L1 et la solution générée par la couche L2. Seules les séquences de tâches sont transmises de L1 à L2 car afin de garantir une certaine flexibilité dans L2 les dates d'exécution des tâches ne sont pas transmises.

R.4.6 Décomposition dans un cas d'étude de déploiement multi-robots

L'approche générique proposée est illustrée en considérant un exemple de l'application de DMR comme celui décrit au chapitre 2. Les ressources avec durée de setup de la couche L1 sont les robots. Dans la couche L2, elles sont raffinées en prenant en compte les points de passage et les liens. Les méthodes de décomposition introduites correspondent à plusieurs chemins candidats pour se déplacer entre deux positions données. Notons que les différents mécanismes anti-collision peuvent être modélisés au sein de ce cadre d'interaction générique. Les modèles à gros grain et à grain fin sont présentés en détail dans la section 4.6.

R.4.7 Conclusions

Dans ce chapitre, nous avons introduit des **mécanismes génériques basés sur la décomposition** pour modéliser les problèmes d'ordonnancement impliquant des ressources pour lesquelles il existe des opérations de *setup* complexes entre les principales tâches qu'elles doivent effectuer. Ce cadre générique, au lieu de considérer la décomposition partielle itérative des abstractions de tâches comme dans le chapitre précédent, introduit une décomposition du problème permettant de considérer une classe plus large de ressources avec des durées de *setup* ainsi que des alternatives d'exécution (méthodes de décomposition) pour certaines tâches, comme dans la planification HTN. Ce cadre générique peut être étendu à plusieurs problèmes d'ordonnancement impliquant des opérations de *setup* complexes.

R.5 Chapitre 5 : Stratégies de décision à deux niveaux pour l'ordonnancement hiérarchique itératif

R.5.1 Introduction

Ce chapitre présente plusieurs stratégies itératives multi-couches pour traiter les problèmes génériques d'ordonnancement hiérarchique définis au chapitre précédent, en particulier ceux qui peuvent être modélisés comme des problèmes impliquant des ressources qui doivent effectuer des opérations de *setup* complexes entre les tâches principales qu'elles réalisent.

L'idée principale du processus itératif entre les couches interactives est brièvement décrite à la figure 7. Deux types d'interactions entre les deux couches sont proposés par la suite.

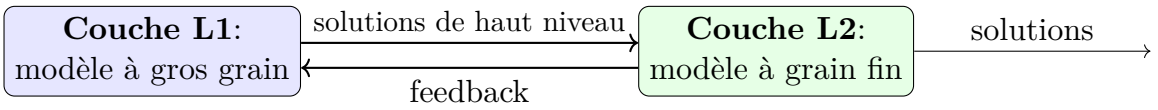

Figure 7: Interaction entre deux couches de décision.

La première approche utilise une stratégie **itérative flexible** dans laquelle la couche décisionnelle de haut niveau encapsule un modèle de substitution d'une ou plusieurs couches en aval. Au cours des itérations, ces couches plus fines peuvent fournir au modèle de substitution de nouvelles informations pertinentes permettant d'améliorer sa qualité. Cette première contribution a été présentée en 2019 lors de la conférence IJCAI [107] et un court résumé a été présenté lors de la conférence ROADEF [109].

La deuxième approche multi-couches utilise **une stratégie de génération de coupes** dans laquelle une couche en aval contient un module d'explication capable de générer des contraintes portant sur des variables de décision de haut niveau. Pour le problème de DMR, ces contraintes (ou coupes) tiennent compte des interférences trouvées dans les solutions de bas niveau et que l'ordonnanceur de haut niveau doit prendre en considération pour minimiser le makespan. Quatre variantes de la stratégie de génération de coupes ont été étudiées. Cette deuxième contribution a été présentée en 2019 lors de la conférence CP [108], et un court résumé a été présenté en 2020 lors de la conférence ROADEF [110].

Ce chapitre est organisé comme suit. La section R.5.2 (section 5.2 dans la partie correspondante en anglais) présente l'approche itérative basée sur des modèles de substitution pour faire interagir les couches de planification et examine la relation de cette approche avec la méthode *Surrogate-Based Optimization* (SBO). La section R.5.3 (section 5.3) présente les différentes variantes de la stratégie de génération de coupes. La section R.5.4 (section 5.4) présente une comparaison entre la stratégie basée sur les modèles de substitution et les différentes variantes de l'approche de génération de coupes proposée. Enfin, la section R.5.5 (section 5.5) fournit des conclusions sur les contributions et donne quelques pistes pour des recherches futures.

R.5.2 Une stratégie décisionnelle par substitution pour l'ordonnancement hiérarchique itératif

Approche de résolution itérative. Dans cette première approche multi-couches, un niveau supérieur encapsule un modèle de substitution d'un ou plusieurs niveaux en aval. Ces étapes plus fines (niveaux hiérarchiques) peuvent fournir à l'étape supérieure des informations pertinentes qui permettent d'adapter certains paramètres d'entrée du modèle de substitution.

Dans le cadre que nous proposons, le modèle de la couche L1 est un modèle de substitution au travers des ressources avec durée de setup qui représente des opérations plus complexes à bas niveau. Au départ, la couche L1 dispose d'une approximation de ces durées de setup. Puis, à chaque fois qu'un nouvel ordonnancement détaillé est produit par la couche L2, les

durées de setup du modèle gros grain de la couche L1 sont mises à jour et une nouvelle solution de haut niveau est recherchée. Ce faisant, la couche L1 apprend itérativement une meilleure approximation du contenu de la couche L2, le but étant de converger très rapidement vers de meilleures solutions complètes. Plus de détails sur ce processus itératif sont donnés dans la section 5.2.

Travaux associés. L'approche d'ordonnancement à deux niveaux proposée précédemment peut d'abord être reliée aux coupes de Benders. Dans notre cas, la couche L2 ne calcule pas de coupes mais renvoie simplement un plan détaillé à partir duquel les paramètres du modèle gros grain de la couche L1 sont mis à jour. L'approche que nous proposons est plus flexible puisqu'à une étape donnée, chaque durée de *setup* considéré par L1 peut être inférieure ou supérieure aux durées de *setup* réels. L'interaction entre la couche L1 et la couche L2 est en fait plus proche des travaux sur les modèles d'approximation (ou de substitution) pour traiter les problèmes à grande échelle coûteux en termes de calcul [74, 76]. Plus de détails sur ce point sont donnés dans la section 5.2.2.

Résultats expérimentaux. Des expérimentations ont été réalisées sur plusieurs instances de problèmes multi-robots générés de manière aléatoire. Ces instances sont décrites en détail dans la section 5.2.3.1. Pour illustrer les interactions entre les deux couches, les valeurs de makespan obtenues lors des itérations de l'approche proposée sont présentées à la figure 8 pour une instance de problème de DMR. Les meilleurs makespans successivement obtenus par L2 au cours des itérations sont marqués d'un point rempli. Plus de détails sur les résultats expérimentaux sont donnés dans la section 5.2.3. Pour comparer le processus de décision à deux niveaux avec une stratégie de résolution en une seule fois, nous avons mis en œuvre le modèle PPC global décrit au chapitre 2. Des résultats représentatifs sont donnés dans les figures 5.4 et 5.5. Ces résultats démontrent que l'approche itérative proposée est à la fois simple et beaucoup plus efficace que le modèle complet pour résoudre des problèmes de grande taille. Elle permet d'obtenir rapidement des solutions de bonne qualité quelle que soit la taille du problème, et intuitivement le processus itératif utilisé permet à la couche L1 de proposer très rapidement différentes solutions prometteuses à la couche L2, sur la base d'approximations qui peuvent librement manipuler des minorants et des majorants des durées de *setup*, contrairement aux approches qui ne manipuleraient que des coupes valides.

R.5.3 Stratégies de génération de coupes pour l'ordonnancement hiérarchique itératif

Pour la deuxième stratégie introduite, l'application de DMR sert à nouveau de base pour définir clairement l'approche.

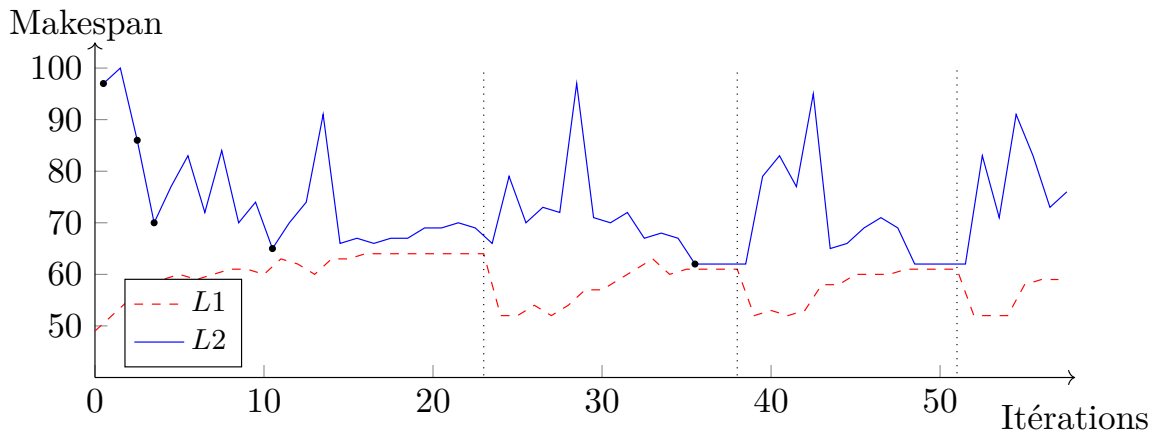

Figure 8: Évolution des valeurs du makespan pendant l'interaction entre les deux couches.

Approche de résolution itérative. Nous introduisons ici une autre stratégie de résolution itérative liée aux LBBD[5], où un solveur maître propose itérativement des solutions à un solveur esclave qui génère de nouvelles contraintes appelées *coupes*. La couche L2 contient un module d'explication qui détecte les interférences entre les tâches consommant les ressources réseau partagées, et synthétise des coupes qui sont envoyées en retour à la couche L1. Dans notre cas, les coupes générées ne sont pas nécessairement valides dans le sens où elles pourraient élaguer les solutions optimales.

Génération de coupes dans le module d'explication de la couche L2. Le module d'explication de L2 renvoie des informations sur les interférences trouvées dans la solution de bas niveau et qui ont un impact négatif sur la minimisation du makespan. Ces interférences sont détectées en examinant les conflits liés à l'utilisation des ressources du réseau lors des traversées de chemins. La figure 9 illustre un scénario dans lequel deux robots sont en conflit pour l'utilisation des points de passage wp_1 et wp_2, et relient l_2 pour parcourir les chemins nécessaires à l'exécution des séquences d'observations. Un exemple d'interférence est représenté en rouge dans les figures 10 et 11, où la durée de la transition est plus longue pour le robot r_1 puisqu'il doit attendre que certaines ressources du réseau soient libérées par r_2. Plus de détails sur le module d'explication sont donnés dans la section 5.3.2.

Les variantes proposées de la stratégie de génération de coupes diffèrent dans la manière dont elles permettent de diversifier la recherche. Quatre catégories de coupes qui peuvent être générées par le module d'explication sont introduites, par ordre croissant de raffinement.

1. Coupes larges : On impose que les durées de setup considérées dans la couche L1 soient supérieures ou égales aux durées obtenus avec interférences identifiées dans L2.

2. Coupes modérées : On impose que les durées de setup considérées dans la couche L1 soient supérieures ou égales aux durées identifiées dans L2 seulement si elles concernent

[5]Logic-Based Benders Decomposition

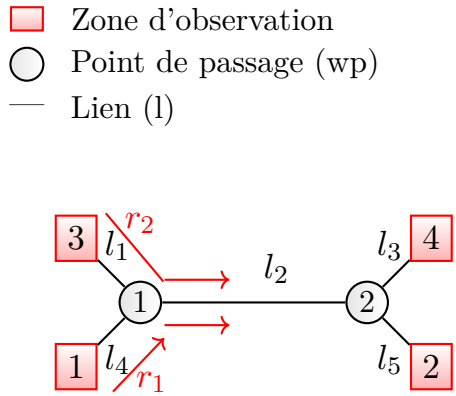

Figure 9: Interférence entre deux robots.

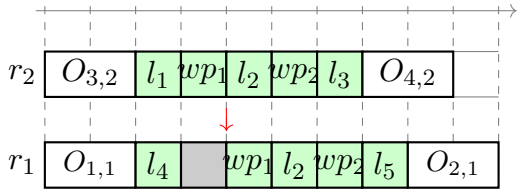

Figure 10: Interférence dans le cas "transfert minimum".

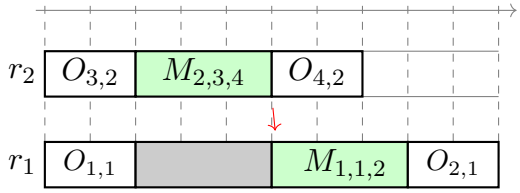

Figure 11: Interférence dans le cas "chemins non partageables".

des séquences d'observation identiques à celles générant des interférences.

3. Coupes raffinés : On impose que les durées de setup considérées dans la couche L1 soient supérieures ou égales aux durées identifiées dans L2 seulement si elles concernent des séquences d'observation identiques à celles générant des interférences et s'il y a à nouveau un chevauchement temporel entre les transitions impliquées dans les interférences.

4. Coupe globale valide : Pour chaque robot, on interdit la séquence de toutes les observations proposées par la couche L1.

Résultats expérimentaux. L'approche à deux niveaux et les quatre stratégies de génération de coupes proposées ont été évaluées sur plusieurs cas d'étude de DMR. Pour la quasi-totalité des instances, les quatre stratégies proposées pour l'approche à deux couches donnent de meilleurs résultats que l'approche globale en PPC dans un temps de calcul nettement plus court. Comme le montre le tableau 5.2 disponible dans la version complète, les résultats montrent également que sur l'ensemble des benchmarks testés, il n'y a pas un seul gagnant

parmi les quatre stratégies de génération de coupes proposées et qu'il existe une sorte de compromis exploration/exploitation selon les cas, avec également une disparité dans le nombre de coupes ajoutées et dans le temps écoulé jusqu'à ce que la meilleure solution soit trouvée. Pour tirer parti de toutes les coupes, l'étape suivante consisterait à définir un *portfolio solver* exploitant les différents types de coupes, l'objectif étant de battre chaque stratégie individuelle de génération de coupes. Les approches de type *portfolio* combinent différents solveurs pour en obtenir un globalement meilleur, et leur efficacité a déjà été démontrée dans le domaine de la PPC [4, 5, 105].

R.5.4 Comparaison : stratégie basée sur les modèles de substitution vs. approche de génération de coupes

Les expérimentations présentées dans cette section confirment également la conclusion des résultats expérimentaux de la section 5.3 sur l'intérêt d'un solveur de type portfolio exploitant les différents types de coupes, l'objectif étant de surpasser chacune des cinq stratégies itératives individuelles (y compris la stratégie basée sur les modèles de substitution). Dans certains cas, il peut être plus avantageux de diversifier l'exploration de l'espace de recherche en générant des coupes modérées, des coupes larges ou en actualisant l'ensemble des durées de *setup*, tandis que dans d'autres cas, il peut être plus pratique d'explorer un espace de recherche qui n'est pas si éloigné du problème actuel en générant des coupes plus fines. Plus de détails sur cette comparaison peuvent être trouvés dans la section 5.4.

R.5.5 Conclusions

Dans ce chapitre, deux nouvelles approches flexibles pour traiter les problèmes d'ordonnancement impliquant des opérations de *setup* complexes ont été présentées. Ces processus à deux couches ne sont pas utilisés pour obtenir une solution optimale mais pour obtenir des solutions de bonne qualité dans un temps de calcul court. Les approches exploitent les points forts des solveurs de PPC existants et donnent des temps de calcul acceptables, même sur des problèmes pour lesquels l'ensemble des décompositions possibles des opérations de *setup* est considérable.

Les résultats obtenus démontrent l'efficacité et la complémentarité de ces coupes. La complémentarité des coupes avec la stratégie de substitution conduisent à l'idée de les fusionner dans un solveur de type *portfolio*. Enfin, les approches proposées peuvent être étendues à d'autres problèmes d'ordonnancement impliquant des opérations de *setup* complexes entre les tâches principales.

R.6 Chapitre 6 : Cas d'étude sur l'optimisation de l'équilibrage des lignes d'assemblage

R.6.1 Introduction

Ce dernier chapitre consiste en une dernière partie exploratoire de cette thèse et cherche à étudier d'autres applications pratiques qui peuvent être traitées comme des POHs. Les expérimentations menées dans ce chapitre serviront de résultats préliminaires pour de futures recherches. Plus précisément, nous considérons dans ce chapitre un autre cas d'étude concernant une application liée aux *chaînes d'assemblage en aéronautique*.

Ce chapitre est organisé comme suit. La section R.6.2 (section 6.2 dans la partie correspondante en anglais) présente un aperçu de l'application d'équilibrage des lignes d'assemblage comprenant une description formelle des données d'entrée d'un cas d'étude général. La section R.6.3 (section 6.3) présente un premier cadre basé sur la décomposition pour traiter ce nouveau cas d'étude. Finalement, la section R.6.4 (section 6.4) traite des expérimentations préliminaires et fournit des conclusions sur les apports de ce dernier chapitre. Il donne également quelques indications pour la suite des recherches.

R.6.2 L'application d'équilibrage des lignes

Comme indiqué précédemment, l'application qui nous intéresse dans ce chapitre est une application de chaîne d'assemblage d'avions dans laquelle la chaîne de production est organisée comme une *pulse line* contenant plusieurs postes de travail pour traiter une section d'avion. La proximité de ce type d'applications avec le RCPSP impliquant des fenêtres temporelles est analysée plus loin dans ce chapitre. D'autres cas d'étude impliquant des fenêtres temporelles et provenant de différents domaines peuvent être adaptées à l'approche présentée ci-dessous.

D'un point de vue général, la conception des lignes d'assemblage d'avions est pertinente à la fois pour les besoins industriels et à des fins académiques [97]. Dans ce contexte, une activité clé est l'*équilibrage de ligne*, qui dans l'aéronautique est utilisé pour désigner un ordonnancement des tâches d'assemblage et n'est pas lié à l'équilibrage réel de cet ordonnancement [114].

On considère ici que les activités sont réalisées sur une ligne pulsée, composée de plusieurs stations de travail. L'avion arrive à la première station et après une certaine durée, appelée le takt, il passe sur la station suivante. Les opérations de montage sont terminées quand l'avion sort de la dernière station de travail. L'objectif considéré est de déterminer le nombre minimum d'opérateurs requis dans la ligne pulsée, mais on peut aussi chercher, par exemple, à diminuer le *takt time* (c'est-à-dire le temps passé sur chaque station de travail de la ligne pulsée), ou à équilibrer effectivement l'attribution des tâches aux ressources disponibles. Les contraintes prises en compte dans ce problème sont les suivantes :

- les zones dans lesquelles les activités d'assemblage sont réalisées sont limitées en termes de nombre d'opérateurs pouvant travailler simultanément (occupation de zone) ;
- certaines tâches peuvent neutraliser des zones dans le sens où pendant que la tâche est réalisée, la zone neutralisée ne peut être occupée par aucune autre tâche ;
- il existe des contraintes de précédence entre les tâches ;
- et les tâches ne peuvent pas être à cheval sur deux stations consécutives.

Plus de détails sur ce problème sont donnés dans la section 6.2.

R.6.3 Modèle basé sur la décomposition

Dans cette section, un premier cadre basé sur la décomposition est proposé pour l'application considérée. Comme dans le chapitre 5, une approche à deux couches est introduite, mais ici la couche de haut niveau L1 est basée sur une formulation MILP (*Mixed Integer Linear Programming*), et la couche L2 peut soit être basée sur un modèle *Constraint-Based Scheduling* détaillé, soit utiliser certaines entrées de la couche L1 pour générer rapidement des plans à partir d'un simple ordonnanceur heuristique. Plus de détails sur ce point se trouvent dans la section 6.3.

L'approche envisagée pour la couche L1, qui repose sur des intervalles temporels partiellement occupés, est décrite plus formellement dans la section 6.3.1. Dans la couche L1, l'idée est d'approximer l'occupation et la neutralisation des zones par les tâches. Pour cela, on découpe les stations de travail en intervalles temporels et considérons les contraintes au global sur ces intervalles. Dans la couche L2, nous récupérons dans la solution transmise par la couche L1 le nombre d'opérateurs pour chaque station de travail ainsi que pour chaque tâche la station de travail dans laquelle elle est réalisée, et nous considérons les contraintes fines d'occupation et de neutralisation. Les différentes alternatives pour la couche L2 sont examinées dans la section 6.3.2.

R.6.4 Conclusion

Dans ce chapitre, un autre cas d'étude concernant une application réelle d'*équilibrage de lignes* a été introduite. Cette application peut être reliée au RCPSP impliquant des fenêtres temporelles (RCPSPTW), et des cas d'étude similaires impliquant des fenêtres temporelles peuvent être adaptés à la représentation introduite. L'interaction entre les deux couches et les informations transférées dans un éventuel processus itératif devraient être étudiées de manière plus approfondie.

Les POH introduits dans les chapitres précédents de cette thèse, ne permettent pas de saisir certaines caractéristiques plus complexes du type de problème décrit dans ce chapitre, par exemple, les **ressources cumulatives** et les **contraintes de neutralisation** considérées

pour les zones d'avion. Ainsi, une formalisation plus générique des POH serait nécessaire pour traiter efficacement des problèmes tels que celui examiné dans ce chapitre et couvrir un plus grand nombre de types de ressources et de contraintes.

Les résultats obtenus sont des **résultats préliminaires** dans le cadre de l'exploration de nouvelles stratégies de décomposition pour différents types d'applications, et serviront de base à de futures recherches.

Conclusions

Contributions

Dans cette thèse, nous introduisons plusieurs cadres et approches de modélisation pour aborder les problèmes de prise de décision *hiérarchique*, formalisés ici sous le nom de problèmes d'ordonnancement hiérarchique. Nous manipulons des représentations hiérarchiques adaptées aux problèmes CBS basées sur des modèles "tâches-ressources" et non sur des modèles "états-actions" comme dans la planification HTN.

Un POH basé sur plusieurs représentations alternatives d'un cas d'étude de déploiement multi-robots est formalisé, ainsi qu'un premier encodage en programmation par contraintes. Ces représentations alternatives du problème impliquent trois mécanismes anti-collision différents afin d'éviter les collisions lors des traversées d'un réseau partagé, ou au moins de réduire la nécessité de traiter les situations de collision en ligne.

Une première méthode de décision pour traiter les POH reposant sur les abstractions de tâches et la décomposition itérative des tâches est introduite. Cette approche tente d'utiliser la décomposition des tâches pour guider la recherche, en l'exploitant au niveau algorithmique et pas seulement à des fins de modélisation.

Des cadres plus génériques pour modéliser les POH permettant de prendre en compte des ressources disjonctives avec des durées de *setup* et des alternatives d'exécution sont également présentés. En particulier, un cadre générique à une seule couche et un cadre générique à plusieurs couches sont introduits pour modéliser les POH. Ces cadres sont illustrés dans un cas d'étude de DMR.

Plusieurs stratégies multi-couches efficaces pour faire face aux POH impliquant des opérations de *setup* complexes sont proposées. La première utilise une approche itérative flexible basée sur des approximations, dans laquelle une couche décisionnelle de haut niveau encapsule un modèle de substitution d'une ou plusieurs couches en aval. La seconde utilise une stratégie de génération de coupes dans laquelle une couche en aval contient un module d'explication capable de générer des contraintes portant sur des variables de décision de haut niveau. Les résultats obtenus démontrent l'efficacité et la complémentarité de l'approche itérative reposant sur les modèles de substitution et des quatre stratégies de génération de coupes proposées. Ces approches exploitent les points forts des solveurs PPC existants et donnent des temps de calcul acceptables, même sur des problèmes pour lesquels l'ensemble des décompositions possibles des opérations de *setup* est important. Enfin, elles peuvent être étendues à d'autres problèmes d'ordonnancement impliquant des opérations de *setup* complexes entre les tâches principales.

Finalement, un autre cas d'étude concernant une application en production liée à l'optimisation de l'équilibrage des lignes est introduit, ainsi qu'un premier cadre (exploratoire) de décomposition pour traiter ce nouveaux cas.

Perspectives de recherche

Le mécanisme d'abstraction des tâches et de décomposition itérative des tâches présenté au chapitre 2 pourrait être considérablement amélioré par l'utilisation de stratégies d'abstraction plus avancées ou plus fines, par exemple en envisageant des abstractions utilisant des ressources cumulatives qui pourraient représenter un pourcentage de la consommation d'une ressource par une sous-tâche. Les **informations transférées entre les itérations** pourraient également être revues, de sorte que l'espace de recherche ne se retrouve pas significativement réduit.

La complémentarité des coupes et de la stratégie de substitution conduit à l'idée de les fusionner dans un *portfolio* de coupes. Cette idée pourrait être affinée dans le cadre de recherches à venir, pour lesquelles on pourrait envisager des stratégies de redémarrage lorsque les solutions trouvées par les deux couches ne peuvent pas être améliorées. Par ailleurs, les approches proposées peuvent être étendues à d'autres problèmes d'ordonnancement impliquant des opérations de *setup* complexes entre les tâches principales. Pour tous les mécanismes de décomposition introduits, les réglages concernant le temps autorisé par itération et les paramètres d'apprentissage devraient également être révisés.

L'idée d'un apprentissage incomplet pour la stratégie de substitution et l'approche par génération de coupes pourrait également être explorée. L'objectif pourrait être d'adopter des mécanismes déjà existants dans le domaine du *Machine Learning*, par exemple, en ajoutant un réseau de neurones pour représenter dans la couche L1 le contenu de la couche L2. Cette idée est déjà utilisée par certains travaux connexes dans la littérature.

En ce qui concerne la modélisation, des cadres plus génériques capables de traiter les POH avec des classes de ressources plus générales pourraient être envisagés. Par exemple, il pourrait être utile de traiter les ressources cumulatives comme mentionné dans le dernier cas étudié. Un défi dans ce dernier cas est de pouvoir gérer un compromis entre la généricité (comme dans la planification HTN) et l'efficacité dans le traitement d'un problème de RO donné. La dernière partie exploratoire de cette thèse pourrait conduire à aller plus loin pour aborder d'autres applications qui peuvent être traitées comme des POH. Nous pouvons également chercher à explorer davantage des alternatives de décomposition qui impliquent des paradigmes autres que la PPC.

Introduction

Context

In many **real-world decision-making problems**, it is very useful and sometimes a requirement to **reason at different hierarchical levels of abstraction**. In this context, the notion of **hierarchy** is associated with the relations between the different features of a problem, which may be given in the form of hierarchical structures. This concept *must not be mistaken for* as the type of hierarchy in which an organizational planning is split in three main steps: strategic, tactical and operational planning. The latter form of hierarchy is more concerned with the time horizon (short, medium or long term) of the defined goals of an organization planning.

A first example of a practical application is **the Multi-Robot Deployment** (MRD) problem [115, 143]. An MRD problem considers a robot exploration mission involving decisions on how exploration (or acquisition) tasks are shared between robots, how these tasks are successively performed by each robot, and how robots moves are coordinated so as to avoid collisions or to maintain communication links. For instance, in this kind of problem we are interested at a high level **(a)** in the way in which the different acquisition tasks that must be carried out are distributed among the available robots; at a lower level **(b)** in the successive moves of each robot within a graph of connected waypoints in order to carry out its allocated tasks; and still at a lower level **(c)** in the fine evolution of the quantity of available energy of each robot as the mission develops. Scheduling problems associated with different MRD case studies will be discussed throughout this manuscript.

Another example of such hierarchical decision making problems is **the management of a constellation of Earth observation satellites** [92, 116], for which decisions must be made concerning how candidate observation tasks are shared between satellites, how each satellite realizes the set of observation tasks allocated to it, and the basic commands that must be sent to instruments for achieving the acquisition plan. For instance, in this kind of problem, we consider at a high level **(a)** the allocation of acquisition tasks to the available satellites; also at a high level **(b)** the realization of the complex observation tasks such as covering large geographical areas; at a lower level **(c)** the realization of the required elementary observation tasks to perform complex observation tasks; and at a lower level **(d)** the fine kinematics of the satellites to determine the sequence of acquisitions to be realized by each satellite.

Another example is the case study related to **the mapping of functions onto an embedded architecture** [55, 111], for which decisions must be made concerning the allocation of functions onto real-time computation units, then decisions concerning the

scheduling of functions on each unit, and last decisions on the routing of data exchanged between functions inside an available inter-units network.

A last example concerns **line balancing applications** [97, 114] in the aeronautics industry. In such applications, the production line is organized as a pulse line containing several work stations to process an aircraft section. The decisions to be made concern the allocation of tasks to such work stations, the scheduling of tasks on each station, and the allocation of the workforce over the line. A scheduling problem based on this case study will be discussed in the last chapter of this manuscript.

Such scheduling problems require being able to **deal with different decision levels** covering *task allocation* and *task scheduling* over resources. Additionally, **each task might be decomposed into subtasks**, as in the case of a function computation task which is decomposed into one subtask for reading the set of data used by the function, one subtask for actually performing the computation, and one subtask for writing the result of the computation in a dedicated buffer.

Also, in such decision making problems, **the constraints can be modelled using different abstraction levels**. For instance, in an MRD application, there are constraints on the available energy for each robot all along the mission. At the time of task sharing between robots, it is not possible for *combinatorial* reasons to consider complex dynamic models which link the available energy to the instantaneous power consumption. Instead, a *simpler model* might be used, in which a fixed amount of energy is consumed by each task. The complex energy model can be considered again once tasks have been allocated to robots and once robot moves have been synthesized. Similarly, for Earth observing satellites it is possible to consider a fixed duration between the realization of two successive acquisition activities before considering a finer model taking into account the kinematics of satellites and the coordinates of images to be realized [93].

Different approaches are often considered to deal with scheduling problems as the ones described above.

- A first common approach is to **explicitly break down the problem to be solved into several sub-problems** and to **define for each of these sub-problems a dedicated resolution technique**.

- A second approach is to use more generic solutions as is done in *planning* with the Hierarchical Task Networks (HTN) framework, where the generic problem considered is to **decompose high-level tasks into so-called atomic tasks using a catalogue of task decomposition methods** provided as an input.

Outline and contributions

The objective in this thesis is to **define modelling frameworks** and **algorithms** for dealing with hierarchical decision making problems as the ones described above.

In relation to existing planning frameworks, this thesis seeks to **manipulate hierarchical representations which are adapted to Constraint-Based Scheduling (CBS) problems**. These representations **are based on "tasks-resources" scheduling models**, defined by a set of *candidate tasks* and by a set of *available resources* to perform these tasks, **and not on "states-actions" planning models**, defined by the parameters describing the state of the system and by the actions available to change this state.

The aforementioned approach has several major advantages. **(1)** First of all, many real-world case studies as the applications described above are quite **naturally formalised as "tasks-resources"** scheduling problems. **(2)** Also, being able to benefit from the **central notion of resource** allows for more direct cross-cutting reasoning at the different levels of the decision-making hierarchy. In the HTN framework, these interactions are less obvious as they are potentially hidden behind more basic parameters describing the current states of the system. **(3)** Another advantage when handling hierarchical tasks concerns the **modularity** in the description of the problems to be solved (reuse of models of existing tasks), the **expressiveness** in the decompositions allowed for the tasks, and finally the **legibility** of the solutions that can be obtained.

In this thesis, the resolution techniques developed strongly rely on existing CBS solvers and we attempt to exploit them to their best in our hierarchical frameworks. Accordingly, on the considered problems, we did not compare our approaches to all of the existing OR techniques.

The present manuscript is organized as follows.

First of all, a **state of the art** is presented in Chapter 1. This state of the art reviews the generalities of scheduling problems, including several approaches to model and solve them. Then, it discusses the notion of *hierarchy* in the planning and in the scheduling domains. Problem decomposition techniques are then described, to finally close this chapter with the concept of Hierarchical Scheduling Problem (HSP).

A **first case study related to a hierarchical MRD application** is introduced in Chapter 2. An overview of this application is presented along with its corresponding CBS model. To reduce the need to deal with collision situations online, different alternatives for modelling this MRD problem have been considered in this chapter. All of these modelling alternatives will serve to present the decision methods presented in Chapter 3 to 5.

A first decision method to deal with HSPs is presented in Chapter 3, along with a

first modelling framework for handling HSPs. The decision method presented in this chapter is based on **task abstractions and iterative task decomposition**.

In order to provide more generic frameworks to model HSPs, Chapter 4 introduces **a decomposition-based framework** which allows to consider a wider class of resources, as well as execution alternatives (decomposition methods) for some tasks as in the HTN framework.

New iterative decision strategies are then introduced in Chapter 5. These novel strategies, instead of considering the iterative partial decomposition of task abstractions as in Chapter 3, uses an *iterative surrogate based strategy* and *an iterative cut generation approach*, based on the previously introduced modelling framework, which again allows to consider more types of resources and execution alternatives for some tasks.

Chapter 6 consists of a final exploratory part of this thesis and seeks to investigate other applications which can be treated as HSPs. More precisely, this chapter presents **another case study regarding a real-world manufacturing application related to line balancing optimization**, along with a first decomposition strategy to deal with this new case study.

The conclusions of the work presented in this thesis are provided together with future research directions. Last, the Annexes present some complementary results on the comparison between the presented approaches.

Chapter 1

State of the art

Contents

In this chapter, a very general outline of scheduling problems is presented in Section 1.1. Sections 1.2 and 1.3 address the spotlight of this chapter, concerning the concept of hierarchy in planning and scheduling problems and a review of modelling and solving techniques commonly used to address these kinds of problems. Finally, Section 1.4 briefly introduces the concept of Hierarchical Scheduling Problems (HSPs) which will be discussed in the remaining chapters with more details.

1.1 Generalities about scheduling problems

The importance of scheduling has been revealed in numerous fields and decision-making environments, including different types of manufacturing and service industries. A scheduling problem deals with the allocation of a set of resources to a set of tasks which must be scheduled over given time periods, to optimize one or more objectives while satisfying specific constraints. The available resources to execute the tasks can be associated with technical or human facilities, such as machines in a workshop or operators on a production line. The tasks to be performed can also take different forms, according to the type of application or organization. They are usually called operations in a production environment.

1.1.1 Components of a scheduling problem

Many research works on scheduling have been focused on deterministic scheduling problems. In such problems, all the parameters associated with tasks and resources are known in advance, and there is no randomness involved. Many different forms of notations to address such problems [113] have been introduced in the last four decades. A notation framework capturing the structure of a huge variety of scheduling problems was introduced by Graham *et al.* [58]. This framework, initially adapted to workshop scheduling problems, has been constantly adapted and extended, allowing the classification of a wide variety of scheduling problems. It consists in a triplet $\alpha \mid \beta \mid \gamma$, describing a scheduling problem. The α field is related to the problem resources, which in production environments are often referred to as *machines*. The β field describes the task characteristics and constraints, and the term *job* in this field is often introduced to represent a set of operations to be performed. Finally, the γ field corresponds to the objective function. Below, a non-exhaustive review of the main elements that may be involved in these fields is presented.

1.1.1.1 Input data

The number of available resources or machines is usually denoted by m, and the number of operations to perform, or jobs, is denoted by n. The subscripts i and j respectively refer to a machine and a job in the following inputs of a scheduling problem. These inputs are:

- a **processing time** ($p_{i,j}$), which corresponds to the processing time of job j on machine i. It can be machine independent and in this case, denoted p_j;
- a **release date** (r_j), which refers to the earliest date at which any machine can start to process job j;
- a **due date** (d_j), which refers to the date before which a job must be completed. The due date is often called a **deadline**;
- a **weight** (w_j), which denotes the priority or importance of a job j, in relation to the other jobs that must be executed.

1.1.1.2 The α field: the machine environment

The α field contains just one entry, specifying one of the following machine configurations:

- **identical machines in parallel** (Pm)**:** a job j requires a single operation on any of the m identical machines which are set in parallel, or on any given subset of machines Mj which must be specified in the β field;
- **flow shop** (Fm)**:** each job j must be executed on each of the m machines in series, following the same route as the rest of the jobs. A queue of waiting jobs is considered for each machine, usually following the *First In First Out* (FIFO) modality;
- **job shop** (Jm)**:** each job j must be executed following a pre-defined route on the m machines. Each single job can be executed once, or several times on any machine;
- **flexible job shop** (FJc)**:** a generalization of the previous case, involving identical parallel machines placed in different groups or machine stations, called work centers.

More machine environments include: single machine (1), machines in parallel with different speeds (Qm), unrelated (non-identical) machines in parallel (Rm), flexible flow shop (FFc), open shop (Om), among others [113].

In general, the resources considered for a scheduling problem are available in limited quantity, defined by a capacity (cap_i). According to their availability over time, two types can be distinguished. The **renewable resources** are available again in the same amount after each resource utilization (machines, human resources, and space). On the contrary, the global consumption of the **non-renewable or consumable resources** is limited over time (budget and raw materials). A particular type of resource which may have infinite capacity, is the **state resource** [124], characterised by an evolving *state* over time. A task may require a resource to be in a given state to be executed. The resources can also be shareable or non-shareable. The **disjunctive or non-shared resources** can only execute one task at a time, while the **cumulative resources** can execute several tasks simultaneously. In general, the resources are *consumed* by the tasks but they can also be *produced*. The multi-capacity resources that can be consumed and/or produced by tasks are called **reservoir resources**, *e.g.* a fuel tank [124]. A cumulative resource is a special case of a reservoir resource, which is consumed at the beginning of the execution of a task, and produced in the same amount once released.

1.1.1.3 The β field: job characteristics and constraints

The β field may contain no entry at all, a single entry, or multiple entries related to the constraints. These constraints can be related to the conditions of the task execution, or to the features of the available resources. Some examples of the possible entries are:

- **preemptions** ($prmp$), if the execution of a job can be temporarily interrupted (preempted) on any machine, and continued in the same or in another machine;
- **precedence constraints** ($prec$), needing a task execution to be completed before starting to execute another one. Three different elementary layouts of precedence constraints can be distinguished. These layouts are generally combined in a scheduling problem:
  - **chains**, when each job has at most one predecessor and at most one successor;
  - **intrees**, when each job has at most one successor;
  - **outtrees**, when each job has at most one predecessor;
- **sequence dependent setup times** ($s_{j,k}$), considered in many practical applications, when executing a job k immediately after executing a job j. The setup times may also depend on each machine i ($s_{i,j,k}$). They can also be associated with groups of jobs (or job families), and considered zero when processing successively jobs in a same family;
- **breakdowns** ($brkdwn$), defining the periods over which a machine is not available (maintenance, shifts, or technical specifications);

- **blocking** (*block*) between two successive machines with a limited buffer, or without any buffer. A job must remain in the upstream machine (blocked), if the downstream machine or its buffer are full;
- **no-wait** (*nwt*), if the jobs cannot wait in between of any two successive machines, meaning that the starting time of the jobs in a first machine must ensure job execution without waiting at any transition.

This field can also consider release dates (r_j), batch processing ($batch(b)$), machine eligibility restrictions (M_j), permutation ($prmu$), job families ($fmls$), recirculation ($rcrc$), among others [113].

Many research works on theoretical scheduling are focused on problems which only involve *non-preemptive* tasks. A more specific variant of scheduling problems may involve preemptive or non-preemptive *elastic tasks*, for which the resource consumption during their execution can take different values. The sum of the resource consumption of a task over time corresponds to its **energy** value [124].

1.1.1.4 The γ field: the objective function

The γ field usually contains a single entry, related to the objective to be optimized. It can also be empty, when the goal is only to obtain a feasible schedule *i.e.* to satisfy the constraints which are considered. There is also a growing interest in multi-objective optimization, in which different criteria can be combined in a same goal. This goal can involve one or several of the following aspects: the time (total execution time or delays), the resources (maximal or weighted quantities, workload), the costs (production, logistics, return on investment), or the required energy (or debit). Time related features are usually involved in γ, for instance, the **completion time** (C_j), the **flowtime** ($F_j = C_j - r_j$), the **lateness** ($L_j = C_j - d_j$), or the **tardiness** ($T_j = max(L_j, 0)$) of a job j.

The γ field may include, for instance, the *minimization* of the **maximum lateness** (L_{max}), the minimization of the **total weighted completion time** ($\sum w_j C_j$), or the minimization of the **maximum completion time**, also called the **makespan** (C_{max}).

1.1.2 Examples of frequently addressed problems

1.1.2.1 The classical and the flexible job shop scheduling problem (JSSP)

$Jm \mid\mid C_{max}$, refers to the classical widely addressed job shop scheduling problem, with m machines and without recirculation (a job may not visit a machine more than once).

$FJc \mid r_j, s_{i,j,k} \mid C_{max}$, refers to the flexible job shop scheduling problem with c work centers. The jobs have release dates, and machine and sequence dependent setup times are considered.

The objective for both problems is the makespan minimization. Commonly, a job shop scheduling problem with n jobs and m machines is denoted as a $n \times m$ JSSP.

1.1.2.2 The scheduling problem with sequence dependent setup times

$1 \mid s_{j,k} \mid C_{max}$ refers to a single machine scheduling problem involving sequence dependent setup times. This definition corresponds to the well-known *Travelling Salesman Problem* (TSP), where the objective again is the makespan minimization. It can be easily adapted to capture features of many other real world scheduling problems, especially those related to industrial environments.

1.1.2.3 Resource-Constrained Project Scheduling Problem (RCPSP)

$PS \mid prec \mid C_{max}$, refers to the general RCPSP [26], where PS concerns the resource-constrained project scheduling, containing a set of acyclic precedence constraints usually given as a directed acyclic graph. Each activity has a specific consumption on each resource, which can be a renewable or a consumable resource.

$PS \mid temp \mid C_{max}$, refers to the generalized RCPSP or RCPSP/max, which can involve temporal constraints (calendar constraints for example), arbitrary precedences, minimal and maximal time lags, and time windows (RCPSPTW).

1.1.3 Complexity of scheduling problems

The complexity related to the resolution of a scheduling problem is linked to its **combinatorial** nature. The complexity theory, developed to study the complexity of computational problems, seeks to measure this intrinsic difficulty providing a formal mathematical proof. The complexity of a problem corresponds to the infimum of the complexities of the algorithms that may solve that problem (worst-case complexity). The decision problems are those that can be addressed as a yes-no question on all the input values [25]. A wide range of scheduling problems have proven to be **NP-hard** [88, 89], meaning that they cannot be solved with a polynomial time algorithm, unless $P = NP$.

Some examples of frequently addressed NP-hard scheduling problems include among others single machine ($1 \mid\mid \sum T_j$ and $1 \mid s_{j,k} \mid C_{max}$), parallel machine ($P_2 \mid\mid C_{max}$), and job shop scheduling problems ($J_2 \mid rcrc \mid C_{max}$). Other well known NP-hard problems are, for example, bin packing, knapsack, graph coloring, or TSP problems.

Some NP-hard problems can be tackled more efficiently than others. In practice, incomplete methods and approximation algorithms are widely used to successfully deal with combinatorial optimization problems. The **incomplete algorithms** allow feasible solutions to be obtained within reasonable processing times. The solutions provided by these methods may have no *formal* guarantee of performance. Some incomplete methods will be addressed in Section 1.1.5.3. The **approximation methods** may rely on a simplified model of a combinatorial optimization problem to provide a solution, and may involve complete or incomplete algorithms to do so. The obtained solution values are guaranteed to be within a factor of ε from the value of an optimal solution [156]. The value ε, which corresponds to the *performance guarantee*, is usually called the *approximation ratio*.

1.1.4 Specificities of a solution

The term **schedule** often refers to the solution of a scheduling problem, which defines the precise execution dates for each task or job in the system. The term **sequencing** is used when the jobs are only relatively placed to each other without considering time, or any precise dates (a permutation of the n jobs). A feasible non-preemptive schedule is an **active schedule** if no operation can be executed earlier without delaying one or more tasks. A feasible non-preemptive schedule is a **semi-active schedule** if none of the operations can be executed earlier without modifying the task sequencing on any machine.

A set of solution schedules is a **dominant** set *for a given objective* if it contains at least one optimal solution for this objective. Similarly, the set is considered to be a dominant set *for a set of constraints* if it contains at least one feasible solution, satisfying these constraints.

The **critical path** in a solution corresponds to the longest stretch of depending tasks, from the start time to the end time of the schedule. The most common dependency relationship between tasks is a *finish-to-start* or an *end-to-start* relationship (precedences). Each task on the longest path belongs to the **set of critical tasks**, which are related by precedence constraints or execution conditions on the available resources.

1.1.5 Modelling and solving a scheduling problem

In this section, several approaches to model and solve scheduling problems are presented. Two main categories can be distinguished, namely: **exact methods** which have the theoretical capacity to solve scheduling problems to optimality, and **incomplete methods** which efficiently find approximate solutions to these problems. As stated above (Section 1.1.3), a wide range of scheduling problems have proven to be

NP-hard, thus an exact algorithm cannot run in polynomial time to solve all the instances of this kind of problems, unless $P = NP$.

Part of the recent research is focused on incomplete algorithms, which appear to be quite convenient methods when addressing large instances of this kind of problems with the need to obtain feasible solutions within reasonable processing times.

In the following, we give a bit more details about modelling and solving strategies, including mathematical programming, constraint programming and incomplete methods.

1.1.5.1 Mathematical Programming (MP)

In operations research, mathematical programming or mathematical optimization, seeks to solve various types of optimization problems, through the selection of a target (best) element from a set of available alternatives, considering some objective function.

Convex programming concerns the type of problems for which the objective function is convex (*minimization*) or concave (*maximization*) and the set of considered constraints is convex. Many classes of convex optimization problems can be solved in polynomial time by interior point methods for a given precision [103]. **Linear Programming** (LP) deals with the optimization of a linear objective function, subject only to linear equality and linear inequality constraints. An LP algorithm aims to find the point in this feasible region (a convex polytope or polyhedron) where the objective function has the most extreme (smallest or largest) value, if such point exists. LP problems, can be expressed in a canonical form as

$$\text{maximize} \quad \mathbf{c}^{\mathrm{T}}\mathbf{x} \tag{1.1a}$$

$$\text{subject to} \quad A\mathbf{x} \leq \mathbf{b}, \tag{1.1b}$$

$$\mathbf{x} \geq 0. \tag{1.1c}$$

where the objective function is to *maximize* the expression $\mathbf{c}^{\mathrm{T}}\mathbf{x}$ in 1.1a, subject to the constraints specified in the inequalities 1.1b and 1.1c defining the feasible region. Vector $\mathbf{x}$ refers to the vector of variables, matrix A refers to a matrix of known coefficients, and vectors $\mathbf{b}$ and $\mathbf{c}$ refer to vectors of known coefficients. A well known efficient algorithm to address linear programming problems is the Dantzig's simplex algorithm [34], also known as the simplex method.

Integer Programming (IP) deals with the optimization of a problem for which some variables are restricted to take integer values (non-convex). If *all* decision variables of the problem must be integer, it corresponds to a *pure* IP. Otherwise, it corresponds to a **Mixed Integer Programming** (MIP) problem. In general, solving IP models is much more difficult (NP-complete) than solving regular LP models. Even

small problem instances may be hard to solve. An IP model corresponds to an **Integer Linear Programming** (ILP) model if its objective function and *all* of its constraints are linear, otherwise, it is called an **Integer Non Linear Programming** (INLP) model. Similarly, MIP models containing only linear constraints and a linear objective function are called **Mixed Integer Linear Programming** (MILP) problems. Branch-and-bound based algorithms are commonly used to address MILP problems.

Coming back to scheduling problems, three standard general and widely used MIP formulations can be distinguished: the disjunctive formulation [94], the time-indexed formulation [24, 81], and the rank-based formulation [152], which are often combined or adjusted. These methods basically differ in the type of decision variables that are considered, for instance, time-indexed variables or variables reflecting the relative position of jobs on each machine. A computational analysis of such models is presented in [82], a work from which the example below (1.2a to 1.2g) of a disjunctive MIP model is taken, to illustrate a job shop scheduling problem.

As presented in section 1.1.2, a $n \times m$ JSSP contains a set of n jobs (J), and a set of m machines (M). For each job $j \in J$, the list $(\sigma_1^j, \ldots, \sigma_h^j, \ldots, \sigma_m^j)$ denotes the processing order of operations that j must follow through the machines. Each operation in the previous list must be performed in a defined machine among the available set. Operation σ_h^j corresponds to the h-th operation of job j, and σ_m^j to the last operation of job j. Also, for each job $j \in J$ and each machine $i \in M$, $p_{i,j}$ represents the non-negative integer processing time of job j on machine i. Each machine $i \in M$ can process only one job at a time (a disjunctive resource), and once a machine starts to process a job, it must carry on the execution without interruption (non-preemptive tasks). The objective corresponds to the makespan minimization (the maximum completion time of the last operation of any job $j \in J$). The described problem is NP-hard for $n \geq 3$ and $m \geq 2$ [49].

To obtain a mathematical programming formulation of a JSSP, it is possible to consider the following decision variables:

$x_{i,j}$ the start time of job j on machine i,
$z_{i,j,k}$ equal to 1 if job j precedes job k on machine i,

and the disjunctive MIP model is presented below

$$\begin{aligned}
&\text{minimize} && C_{max} && && (1.2a)\\
&\text{subject to} && x_{i,j} \geq 0, && \forall j \in J,\ \forall i \in M, && (1.2b)\\
& && x_{\sigma^j_h,j} \geq x_{\sigma^j_{h-1},j} + p_{\sigma^j_{h-1},j}, && \forall j \in J,\ \forall h = 2,\dots,m, && (1.2c)\\
& && x_{i,j} \geq x_{i,k} + p_{i,k} - V \cdot z_{i,j,k}, && \forall j,k \in J^2,\ \text{s.t. } j < k,\ \forall i \in M, && (1.2d)\\
& && x_{i,k} \geq x_{i,j} + p_{i,j} - V \cdot (1 - z_{i,j,k}), && \forall j,k \in J^2,\ \text{s.t. } j < k,\ \forall i \in M, && (1.2e)\\
& && C_{max} \geq x_{\sigma^j_m,j} + p_{\sigma^j_m,j}, && \forall j \in J, && (1.2f)\\
& && z_{i,j,k} \in \{0,1\}, && \forall j,k \in J^2,\ \text{s.t. } j < k,\ \forall i \in M. && (1.2g)
\end{aligned}$$

The objective function, which corresponds to the makespan minimization, is stated in 1.2a. Constraint 1.2b requires each job start date to be greater than or equal to zero. Constraint 1.2c ensures the respect of the given order of operations for each job. The disjunctive constraints 1.2d and 1.2e, require that no two jobs can be executed simultaneously on the same machine. The large value V corresponds to $\sum_{j \in J} \sum_{i \in M} p_{i,j}$, given that the processing time of any operation cannot be greater than the sum of all the operations processing times. Constraint 1.2f states that the makespan should be at least the largest completion time of the last operation of all jobs. Finally, the binary value of the $z_{i,j,k}$ variables is ensured in 1.2g.

1.1.5.2 Constraint Programming (CP)

In contrast to the formulation of Mathematical Programming models like MILP models, which must satisfy specific mathematical conditions, **Constraint Programming** (CP) models have no limitation on the set of constraints that can be imposed. Constraints in CP relate the decision variables to specify the properties of the solution that must be satisfied for a given problem, which is called a **Constraint Satisfaction Problem** (CSP).

Formally, a CSP on finite domains is defined by a triplet $\langle X, D, C \rangle$ [125], where

$X = \{X_1, \dots, X_n\}$ is the set of variables,
$D = \{D_1, \dots, D_n\}$ is the set of their respective domains ($\forall k \in [1; n],\ X_k \in D_k$)
$C = \{C_1, \dots, C_m\}$ is the set of constraints.

Each constraint $C_i = (S_i, \mathcal{R}_i)$, is defined by a list of variables $S_i = \{X_{i_1}, \dots, X_{i_k}\}$ and a relation $\mathcal{R}_i \subseteq D_{i_1} \times \cdots \times D_{i_k}$. Several types of constraints can be distinguished, including **extensional constraints** which enumerate the combinations of values that satisfy them, **arithmetic constraints** which are defined by arithmetic expressions, or **global constraints** which are defined by explicit semantics such as noOverlap, AtMost, or AllDifferent. These constraints can be stated over specific **finite domains**,

such as **boolean domains** (true/false constraints as in the SAT[1] problem) or **integer domains**.

A solution to a CSP corresponds to a complete assignment of the decision variables to values in their domains, for which all the problem constraints are satisfied. A CSP, when associated with an objective function, becomes a **Constraint Optimization Problem** (COP). Then, a solution to a COP optimizes (minimizes or maximizes) the objective function value.

Solving a CSP can be highly combinatorial. This is why **inference** and **search methods** are used to efficiently explore the search space and try to obtain a solution within reasonable time.

Combinatorial inference and search methods often involve constraint propagation and backtracking techniques [98]. **Constraint propagation** is used to reduce the search space, usually large in practical applications, by means of diverse techniques such as *local consistency* and *rules iteration* approaches. Local consistency techniques include, among others, the well-known procedures of *arc consistency* and *path consistency*. These techniques seek to obtain a simpler problem by reasoning on subsets of constraints to prune values or combinations of values from the variables domains.

Backtracking techniques are used to search recursively for solutions while keeping partial assignments of the values of each variable, until *inconsistent* dead ends. This form of search can be represented by a decision search tree, in which backtracks are systematically performed from the leaves (dead ends), to revert (backtrack) the last variable choice. Different strategies (constraint learning, backjumping, look-ahead) [124] allow to improve search efficiency.

Constraint-Based Scheduling (CBS) is the discipline that seeks to solve scheduling problems using CP [8, 124].

We provide below an example of a CP model (1.3a to 1.3e) for the same JSSP as presented in the previous section. This CP model, also taken from [82], considers the decision variables $x_{i,j}$ which correspond to the start time of job j on machine i, as in the previous disjunctive MIP model (1.2a to 1.2g).

[1]Boolean (or Propositional) Satisfiability Problem

$$\text{minimize} \quad C_{max} \tag{1.3a}$$

$$\text{subject to} \quad x_{i,j} \in [0..V], \qquad \forall j \in J,\ \forall i \in M, \tag{1.3b}$$

$$x_{\sigma^j_h,j} \geq x_{\sigma^j_{h-1},j} + p_{\sigma^j_{h-1},j}, \qquad \forall j \in J,\ \forall h = 2,\ldots,m, \tag{1.3c}$$

$$C_{max} \geq x_{\sigma^j_m,j} + p_{\sigma^j_m,j}, \qquad \forall j \in J, \tag{1.3d}$$

$$\text{disjunctive}(\{x_{i,1},\ldots,x_{i,n}\},\ \{p_{i,1},\ldots,p_{i,n}\}), \qquad \forall i \in M. \tag{1.3e}$$

The objective function stated in 1.3a, corresponds to the makespan minimization. Constraint 1.3b requires each jobs start date to be greater than or equal to zero. Constraint 1.3c ensures the respect of the given order of operations for each job. Constraint 1.3d states that the makespan should be at least the largest completion time of the last operation of all jobs. Finally, the disjunctive constraint 1.3e, involving the start dates and the durations of each of the n jobs on each machine in M, requires that no two jobs can be executed simultaneously on the same machine.

Below, another example of a CP model is given for the same JSSP. Instead of defining constraints over integer variables as the previous CP model (1.3a to 1.3e), the model below (1.4a to 1.4c) uses **interval variables** to model the tasks to perform. In the IBM ILOG CP Optimizer software, interval variables, considered as *first class citizens*, are considered to represent activities [86]. The decision variables $task_{i,j}$ of the CP model presented below correspond to interval variables whose duration correspond to durations $p_{i,j}$ associated with the execution of job j on machine i.

$$\text{minimize} \quad C_{max} = \max(\{\text{endOf}(task_{\sigma^j_m,j}) \mid j \in J\}) \tag{1.4a}$$

$$\text{subject to} \quad \text{endBeforeStart}(task_{\sigma^j_{h-1},j}, task_{\sigma^j_h,j}), \quad \forall j \in J,\ h = 2,\ldots,m, \tag{1.4b}$$

$$\text{noOverlap}(\{task_{i,j} \mid j \in J\}), \qquad \forall i \in M. \tag{1.4c}$$

The objective function stated in 1.4a corresponds to the minimization of the makespan, which is equal to the largest completion time of the last operation among all jobs. Constraint 1.4b ensures the respect of the given order of operations for each job. Finally, the disjunctive *noOverlap* constraint 1.4c requires that no two jobs can be executed simultaneously on any machine. The implementation of these interval variables makes the modelling of scheduling problems easier and enhances constraint propagation [86, 85]. Propagation techniques such as *time-table* constraints, *edge-finding*, *not-first not-last*, among others, are compared in [8].

1.1.5.3 Incomplete methods

Heuristic techniques. Heuristic methods, or simply *heuristics*, are problem-dependent *rules* that seek to take advantage of the specificities of a problem to search for an *acceptable* solution, regarding a trade between accuracy and speed. In general, heuristic methods do not provide a *formal* guarantee of performance on the obtained solution values. They can be really useful to speed up the process of finding an approximate solution (*shortcuts*) in many real-world complex (NP-hard) optimization problems, when exact methods cannot run in polynomial time to find an optimal solution, or when there are no known algorithms to address the problem. For instance, well-known simple heuristics for scheduling problems are the LPT[2] first and SPT[3] first heuristics defining sequencing rules. *Greedy algorithms* are procedures that apply at each step a particular heuristic rule to select the *best available alternative*, regardless to the future consequences that may arise. Commonly used greedy algorithms include, among others, the greedy (serial or parallel) Schedule Generation Schemes (SGS) to solve the RCPSP [80], the well-known *nearest neighbour* greedy algorithm, which is applied in the selection of the next destination in a TSP[4], and the deadline scheduling algorithm known as EDF[5] [137].

Local Search. Local search methods can be considered as a part of a more sophisticated category of heuristics. Local or *neighbourhood* search is an any-time iterative procedure which, starting from any candidate solution, moves to a neighbour solution as long as necessary [25]. In this method, the neighbourhood structure and the set of allowed moves to reach the next solution in the space of candidate solutions (the search space) must be defined. Also, the iterative local moves can be done based on heuristics or basic rules related to the *local* optimization of the criterion (hill-climbing), until a satisfying solution is found, or until a time bound is reached. It is important to note that if no improving move exists in the neighbourhood, local search can get stuck in a local optimum.

Metaheuristic techniques . Metaheuristics are higher-level strategies that guide the solution search process by manipulating, tuning, or adjusting basic heuristic algorithms, local search, greedy strategies, or combinations of them. In a large amount of the recent research, metaheuristics are also combined with learning strategies, and with other techniques such as MP or CP. They are usable for a wide-range of contexts, since they are problem-independent methods making few assumptions on the optimization problem to be solved. Many metaheuristic techniques are commonly used as *partial search algorithms* to guide local search procedures and find globally optimal solutions,

[2]Largest Processing Time
[3]Shortest Processing Time
[4]Travelling Salesman Problem
[5]Earliest Deadline First

such as Simulated Annealing, Tabu Search, VNS[6], ACO[7], ILS [91], or GRASP[8]. Other types of usual metaheuristics are population-based mechanisms that simultaneously improve a set of candidate solutions, such as Particle Swarm Optimization (PSO) and genetic algorithms.

1.2 Hierarchical Planning and Scheduling modelling frameworks

Hierarchical planning and scheduling techniques have been widely used to address large-scale practical applications.

This section first introduces the notion of hierarchy in scheduling, through the concept of Work Breakdown Structures (WBSs). It then describes the hierarchical models used in the related planning field, and in particular, its most representative framework, the Hierarchical Task Network (HTN) planning framework. Finally, variants and extensions of hierarchical planning taking advantage or involving scheduling notions are discussed.

1.2.1 Hierarchical scheduling

The concept of **Work Breakdown Structure** (WBS) [61] was developed together with the PERT[9] [45] as a common framework to be used generally in project management. A WBS is a hierarchical decomposition of a *project*, into several phases or deliverables. It is generally represented vertically, as a tree structure in which parent - child relations are established between the defined project deliverables, enabling a better insight and control on the project execution. A basic WBS allows in a project only one alternative (or decomposition method) to the execution of each of the project tasks. In this case, each task can be directly broken down into a further unique set of leaves or compound subtasks.

The leaves of these structures are the lowest (or terminal) elements which are not further subdivided, and the compound tasks are those that may be further decomposed.

Hierarchical arrangements and WBSs can be used to model task decompositions among different levels in a hierarchy [83, 84], as depicted in the example of Figure 1.1. This example considers two mandatory top level tasks (TLT_1 and TLT_2) decomposed in two and three subtasks respectively, and several precedence constraints (arrows) among

[6] Variable Neighborhood Search
[7] Ant Colony Optimization
[8] Greedy Randomized Adaptive Search Procedure
[9] Program Evaluation and Review Technique

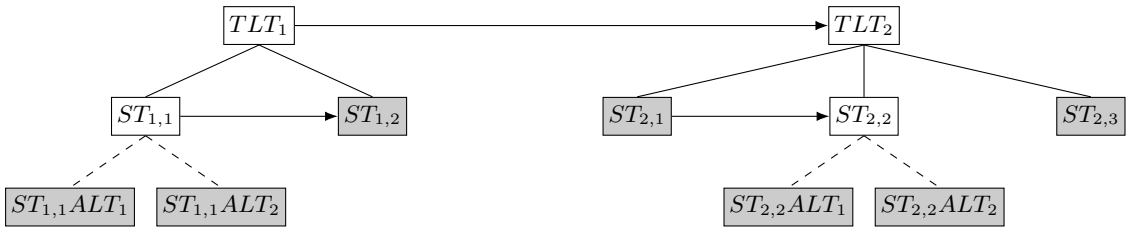

Figure 1.1: Example of a WBS with integrated alternatives for some tasks.

tasks. **Execution alternatives** or **decomposition methods** (dashed lines) for some of the subtasks ($ST_{1,1}$ and $ST_{2,2}$) are integrated to the WBS of this example.

The Listing 1.1, presents an example of a WBS as the one of Figure 1.1, translated to a CP model. This example is brought and adapted from the work of Laborie *et al.*, concerning the IBM ILOG CP Optimizer software [85]. Only the constraints related to the WBS (Lines 11 - 23) are presented below, but any other temporal constraints or resource constraints can be added.

```
1  int n = ...;
2  tuple Dec {int task; {int} subtasks;};
3  int compulsory[1 . . n] = ...;
4  {Dec}  Decs = ...;
5  int nbDecs[i in 1 . . n] = card( {d | d in Decs : d.task == i} );
6  int nbParents[i in 1 . . n] = card( {d | d in Decs : i in d.subtasks} );
7
8  dvar interval task[i in 1 . . n] optional;
9  dvar interval dec[d in Decs] optional;
10
11 constraints {
12   forall(i in 1 . . n) {
13     if(nbParents[i] == 0 && 0 < compulsory[i])
14       presenceOf(task[i]);
15     if(nbDecs[i] > 0)
16       alternative(task[i], all(d in Decs: d.task == i) dec[d]);
17       forall(d in Decs: d.task == i)
18         span(dec[d], all(j in d.subtasks) task[j]);
19     }
20   }
21   forall(d in Decs, j in d.subtasks: 0 < compulsory[j])
22     presenceOf(dec[d]) => presenceOf(task[j]);
23 }
```

Listing 1.1: Translation of a WBS to a CP model (Laborie *et al.* [85]).

In this hierarchical CP model, the n tasks to perform (Line 1) are modelled using **interval variables**. A high-level task may be *decomposed* into a set of lower-level ones (subtasks). Each decomposition of the WBS (Line 2), involves a parent (high-level) task and a set of lower-level subtasks. Each task of the considered set may be

either an **optional** (non-compulsory) task or a **compulsory** one (Line 3). The non-compulsory tasks are those that can remain unperformed, and the compulsory ones must be executed. For a given task, its number of alternative decompositions (Line 5) and its number of parent tasks (Line 6) is computed. The **optional** interval variables associated with the tasks (Line 8) and with the decompositions (Line 9) are defined. The constraint in Line 14 states that the compulsory highest-level tasks must be *present*. The *alternative* constraints for tasks with multiple decompositions are stated in Line 16. To model these decompositions, *span* constraints are stated (Line 18) between the parent tasks and all of its subtasks. The parent task will be constrained to start at the *start time* of the first subtask, and to end at the *end time* of the last subtask performed. Finally, the constraint in Line 22 states that the compulsory subtasks of a given decomposition must be *present*, whenever this alternative decomposition is selected.

1.2.2 Hierarchical planning

Automated Planning (AI Planning), is a field of Artificial Intelligence which studies computational decision making processes performed by intelligent entities (robots, humans, computer programs) to achieve some objectives as best as possible [53, 54]. *Planning* as a decision making process considers a conceptual general model of a dynamic system, called a *state-transition* system [36] or a *discrete-event* system. In contrast to a **scheduling** problem, where the set of tasks is initially known, a **planning** problem considers instead a set of *actions* having an impact on the evolution of the *system states* from an initial state to a goal state through *state transition functions*. Each state transition function consists in a set of planning *operators*, instantiated into *actions*.

Classical planning refers to planning for specific state-transition systems meeting several assumptions (deterministic, fully-observable state-transition system, with restricted goals and implicit time, among others) [53]. More general assumptions on the state-transition systems lead to frameworks such as Markov Decision Processes (MDPs) involving actions that have probabilistic effects, and Partially Observable Markov Decision Processes (POMDPs) where the system state is only partially observable. One of the most commonly used modelling languages in Classical Planning is known as the Planning Domain Definition Language (PDDL) inspired, among others, from the STRIPS[10] planning language [44]. In PDDL, each state is represented by a collection of state variables which can be affected by a set of available actions [52].

[10]Stanford Research Institute Problem Solver

1.2.2.1 Hierarchical Task Network (HTN) planning

HTN planning is an alternative representation of planning problems, in which the objective is not to achieve a goal state, but to perform a set of tasks. Each task of the given set is either a primitive task, executed by a primitive action, or a non-primitive task, broken down into a set of other tasks.

1.2.2.2 Modelling a hierarchical planning problem

The formalization of an HTN planning problem below is taken from the representative works of Erol *et al.* [43] and Ghallab *et al.* [53], and from recent work on the properties of HTN planning and HTN planning systems [18].

As in classical planning, an HTN planning problem considers a set of *states* and *operators* (instantiated as *actions*). Each operator corresponds to a deterministic *state transition function*. Formally, an **operator** o is a triple $(\text{name}(o),\ \text{precond}(o),\ \text{effects}(o))$, defined by a name, a set of preconditions that must be satisfied for the operator to be executed, and a set of effects obtained after execution. The preconditions are related to a *required state* of the system to execute the operator, and the effects define the *final state* of the system after the operator has been executed.

We give below an example of a simple operator for an application involving a robot moving in a certain environment. Various robot deployment applications are discussed and exploited throughout this manuscript. In the following example, the operator $move(r, l, l')$ (Line 1.5a), defines the transfer of a robot r from a location l to an adjacent and unoccupied location l' (Line 1.5b).

$$move(r, l, l') \tag{1.5a}$$
$$;; \quad \text{robot } r \text{ moves from location } l \text{ to an adjacent location } l' \tag{1.5b}$$
$$\text{precond:} \quad \text{adjacent}(l, l'),\ at(r, l),\ \neg\text{occupied}(l') \tag{1.5c}$$
$$\text{effects:} \quad \text{at}(r, l'),\ \text{occupied}(l'),\ \neg\text{occupied}(l),\ \neg\text{at}(r, l) \tag{1.5d}$$

To execute this operator, locations l and l' must be adjacent (Figure 1.2), robot r must be at location l, and location l' must be unoccupied (Line 1.5c). After the execution of this operator, location l' will be occupied by robot r, leaving location l unoccupied (Line 1.5d).

The formalization of an HTN problem also includes *tasks*, *methods* and *task networks*.

In HTN, a **task** can be either a *primitive task* or a *non-primitive* task. If the task

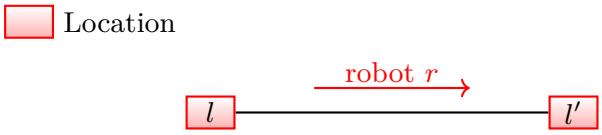

Figure 1.2: Example of operator $move(r, l, l')$ for a primitive task.

can be executed directly by the execution of an action (now, a **primitive action**), it corresponds to a **primitive task**. Otherwise, the task corresponds to a **non-primitive task** (or **compound task**), which needs the execution of a set of further primitive or non primitive tasks to be performed. A non-primitive task is often associated with the execution of a so-called **high-level action** [125].

In HTN, the relations between the problem tasks are described using hierarchical network structures, called task networks (TN). A **task network** w is a pair (U, C), defined by a set U of tasks, and a set C of constraints between them. These constraints can be, for instance, *precedence* constraints requiring the completion of a task (predecessor) to execute another one.

Each non-primitive task is associated with one or several alternatives to be performed. Each alternative is referred to as a **decomposition method**, which denotes a possible task *refinement* into further primitive or non-primitive tasks. Formally, a decomposition method, is a 4-tuple

$$m = (name(m),\ task(m),\ subtasks(m), constr(m)),$$

defined by a unique *name*, an associated non-primitive task ($task(m)$), and a task network ($subtasks(m), constr(m)$).

We give below an example of two decomposition methods (Figure 1.3) associated with a non-primitive task $MOVE(r, l, l'')$ (Figure 1.4), requiring a robot r to move from location l to a *non-adjacent* unoccupied location l''.

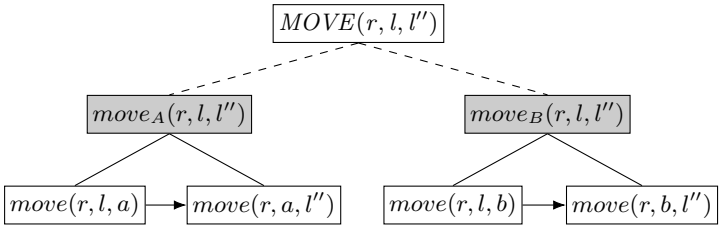

Figure 1.3: Two decomposition methods.

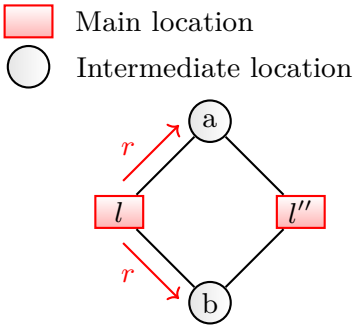

Figure 1.4: A non-primitive task.

$$move_A(r, l, l'') \tag{1.6a}$$
$$;; \quad \text{one method to move robot } r \text{ from location } l \text{ to location } l'' \tag{1.6b}$$
$$\text{task:} \quad MOVE(r, l, l'') \tag{1.6c}$$
$$\text{subtasks:} \quad move(r, l, a),\ move(r, a, l'') \tag{1.6d}$$
$$\text{constr:} \quad move(r, l, a) < move(r, a, l'') \tag{1.6e}$$

$$move_B(r, l, l'') \tag{1.7a}$$
$$;; \quad \text{another method to move robot } r \text{ from location } l \text{ to locat. } l'' \tag{1.7b}$$
$$\text{task:} \quad MOVE(r, l, l'') \tag{1.7c}$$
$$\text{subtasks:} \quad move(r, l, b),\ move(r, b, l'') \tag{1.7d}$$
$$\text{constr:} \quad move(r, l, b) < move(r, b, l'') \tag{1.7e}$$

The alternative methods $move_A$ (Lines 1.6a - 1.6e) and $move_B$ (Lines 1.7a - 1.7e) in the previous example, decompose non-primitive task $MOVE(r, l, l'')$ into the primitive subsequent subtasks ($move(r, l, a)$, $move(r, a, l'')$) and ($move(r, l, b)$, $move(r, b, l'')$), respectively. These methods use the intermediate locations a and b, supposed to be adjacent to both l and l'', to perform task $MOVE(r, l, l'')$. An operator for each of the primitive tasks would resemble the above-described example of an operator (Lines 1.5a - 1.5d).

Finally, an **HTN planning problem** is defined as a 4-tuple $P = (s_o, w, O, M)$, where s_o is the initial state of the system, w is the initial or *root* task network, O is a set of planning operators, and M is a set of decomposition methods. A **solution of an HTN planning problem** corresponds to any plan π containing an *executable* sequence of primitive tasks, obtained from successive decompositions of compound tasks from the root task network. Well-known domain-independent HTN planning systems are, among others, SHOP2 [100, 102] and HTNPlan-P [136].

1.2.2.3 Discussion on hierarchical planning

From the modelling side, HTN planning represents an **expressive** planning paradigm [42], used to deal with generally large-scale problems. The **concept of tasks** makes a wide range of applications easily and *naturally* adaptable to this frame. Even if in HTN planning the domain author needs to specify the set of tasks and decomposition methods *in addition* to the planning operators of classical planning [53], one of the advantages of this hierarchical formalization is related to its **modularity**. The use of hierarchical structures to represent each non-primitive task allows to recall or easily adapt some

previously defined arrangements to new requirements (*recursion*), concerning similar or comparable decompositions of a same type of task. Besides, specific operational procedures can also be modelled through the possible task decompositions.

From an computational point of view, many theoretical classical planning problems can be solved with HTN planners, sometimes more quickly than with classical planners. In HTN planning, significant parts of the search space can be avoided by **reasoning** from the highest level tasks to the primitive ones, speeding up the search through the hierarchical arrangements. In these hierarchical structures, the specified task decompositions may be used to **prune** partially ordered plans of primitive actions [10], providing a sort of control knowledge [7]. For instance, the Conflict-driven Hierarchical Meta-CSP Planner (CHIMP) [138] presents an approach which exploits the restrictions made on the planning search space in hierarchical planning, by means of *pre-defined decomposition rules*. These pre-defined rules aim to efficiently cut a hybrid search space (time, spatial or resources sub-spaces) to generate hybrid robot plans in acceptable time.

Macro-actions [29] (or *macro-operators*) which are actions that decompose directly into a sequence of primitive ones, have also been considered to increase the performance of planning procedures. Through the definition of some offline pruning rules, combinations of macro-actions can be selected or filtered [23]. Mechanisms of **abstraction** have also been studied to exploit the hierarchical structures in HTN planning [78], since early works concerning abstraction hierarchies (ABSTRIPS) [77, 126]. Variable selection **heuristics** might also be used to take advantage of the hierarchical decision levels of the problem structure, defining *branching strategies* usually seeking to address the highest hierarchical level first.

1.2.3 Extensions of the standard HTN framework to consider scheduling features

1.2.3.1 General extensions

In the recent years, more and more HTN planning frameworks have been developed or extended [14]. Some representative HTN planning frameworks designed to mainly address totally-ordered HTN planning problems (those maintaining a total ordering between all actions) are among others, GTOHP [121], HTN2ASP [39], Tot-SAT [15], and Tree-Rex [130]. PANDA [19] and the recent PANDApro [65, 66] combine hierarchical planning with POCL[11] planning and heuristic search strategies. The causal links in POCL planning aim to identify the actions effects that are used to satisfy some preconditions of other actions. Other planning frameworks such as HTN2STRIPS [1, 3] seek to translate HTN planning problems into a sequence of classical planning problems.

[11] Partial-Order Causal-Link

The variants of *standard* HTN planning [18] may also consider subtask sharing among non-primitive tasks, free primitive task insertions as in classical planning, or preconditions and effects added to non-primitive tasks [20]. In an HTN planning framework allowing task insertion, each action may be added to any task network at any position. This planning framework is called Task Insertion HTN or **TIHTN** planning [50] and is usually referred to as *hybrid planning*, since it merges classical and HTN planning.

Even if the HTN planning framework is the best known representation for hierarchical planning, there exist other forms of hierarchical representation frameworks [18]. These forms include the Hierarchical Goal Network (**HGN**) planning [132], in which the hierarchical structure is defined upon a hierarchy of *state variables* instead of a task hierarchy, and the Goal Task Network (**GTN**) planning [2], which merges both task network (HTN) and goal network (HGN) representations. These goal-based reasoning systems seek to overcome the lack of correspondence between tasks in HTN planning and the goals to achieve in classical planning to correctly translate classical planning problems into HTN domain models.

1.2.3.2 Time and resources in HTN planning

One of the *major drawbacks* of the **standard frame** of classical and HTN planning in practical applications is related to the essential implicit concepts of **time and resources** [131]. The representation of real-world applications can turn out to be quite complex without these central notions, since the available resources cannot be directly associated with the execution of the tasks and some temporal interactions may not be easy to express. These central notions are explicit in scheduling, where the resources are available to execute tasks, and allow direct reasoning on the temporal constraints and resource consumptions by the different tasks, among the different levels of the decision hierarchy.

Temporal planning in HTN allows to consider features involved in practical applications such as the duration of the tasks and the temporal constraints to relate concurrent actions and to express their effects, while considering explicit time. In practice, planning problems may involve different types of available **resources** and **complex temporal constraints** which, sometimes, cannot be explicitly handled by existing *standard* classical and HTN planners. Considering temporal and resource constraints in HTN planning can be seen as a sort of integration of planning and scheduling. Some of the existing *extended* HTN planners support explicit reasoning about resources and temporal constraints [128, 134], to some extent. We list below some of these recently developed frameworks.

First, EUROPA[12] [9] is an expressive and flexible platform to integrate advanced planning, scheduling and constraint reasoning into an end-user application. This frame-

[12]Extensible Universal Remote Operations Planning Architecture

work contains a Temporal Network module (efficient constraint propagation and consistency check) and a Resources Management module (data structures and algorithms to support different types of resources) which appear to be particularly useful to deal with practical planning domains. In addition to the temporal and resource knowledge used in frameworks like CHIMP, knowledge provided by external reasoners and other forms of knowledge may be considered thanks to the meta-CSP [13] framework [95], which allows the combination of multiple reasoners. CHIMP is also able to make online reasoning and unify plans by merging tasks under execution with new arriving tasks.

The GSCCB-SHOP2 [118] framework presents an *efficient* HTN planning algorithm to handle multi-capacity discrete resources and complex temporal constraints. It was developed based on SHOP2 [100, 101]. This algorithm integrates three interrelated modules described below, which try to maintain temporal consistency of all the constraints while providing deterministic resource states. While resource reasoning with the state updating rules is conducted in a first Resource module, the Check Consistency and Backtrack (CCB) module aims to retract pruned solutions by maintaining the consistency of temporal constraints, and the Guide Search (GS) module improves the resource utilization to shorten in terms of makespan the generated action plans. A metaheuristic to improve the utilization of resources is also considered in this framework.

The Flexible Acting and Planning Environment (FAPE) [21, 40] introduces a framework based on an expressive language called ANML[14] [135]. The FAPE language combines *timeline representations* with HTN decomposition methods, and supports resource usage. Timelines give a partial view of the evolution of state variables over time. However, the current version of FAPE framework does not support resources. It focusses on the integration of temporal planning and acting, embedded in an autonomous real-time system (a robotic platform). It extends the classical definition of preconditions and effects to a rich temporal hierarchical setting and introduces a constraint-based planning algorithm.

The ASPEN Modeling Language (AML) [32, 48] is also a timeline-based representation. It contains *activities* used to model goals or high level tasks, and *sub-activities* used to refine goals and high level tasks into primitive actions. PLATINUm is another timeline-based planner [144] recently extended to explicitly consider discrete resources and reservoir resources [145], and to integrate them into a general plan refinement procedure.

Despite the multiple recent attempts to efficiently integrate resources into temporal hierarchical planning, there is still no general framework to do so, and the domain modeller must still specify the different types of knowledge and the associated elements (resources, constraints,...) to be considered, according to each different planning

[13] see `metacsp.org`
[14] Action Notation Modelling Language

framework. In an attempt to create a *common input language* to HTN planning problems, some authors have recently defined the Hierarchical Domain Definition Language (**HDDL**) [63, 64], based on the widely used description language of non-hierarchical planning, PDDL 2.1 [47], and inspired, among others, from approaches such as HTN-PDDL [56], HATP [133, 87], and GTOHP [121]. HDDL[15] can be easily extended and allows the comparison of different hierarchical planning frameworks. The syntax is quite similar to PDDL 2.1 [47] and the language allows the problem definition of standard HTN problems using the explicit notions of abstract tasks, actions, decomposition methods with their subtasks and preconditions, among others. Other features of HDDL allow other classes of problems such as TIHTN planning problems to be modelled.

1.3 Multi-stage scheduling through problem decomposition

Returning to the domain of scheduling problems, this section discusses problem decomposition techniques encountered in the recent research, which provide the advantages of not representing a scheduling problem in its entirety.

Problem decomposition techniques have emerged as useful approaches when it comes to solve *computationally expensive* scheduling problems. Instead of modelling and solving a scheduling problem as a whole, a decomposition approach splits it into several different *optimization stages* seeking to decrease the combinatorial complexity. In such a decomposed scheme, the optimization stages may represent for instance, different hierarchical decision levels, two levels dealing respectively with task allocation and scheduling over resources, or simply a set of *easier* sub-problems. This concept of structural decomposition of a scheduling problem must not be mistaken for the concept of *disaggregation*, which mainly concerns resources (tasks or time intervals) being decomposed (or disaggregated) as a way of breaking them into finer units (shorter time periods, for instance) [12].

To decompose scheduling problems, one key step is to exploit the underlying structure of these problems and infer certain information about each of the optimization stages to define an appropriate *reformulation* strategy to implement [147]. A reformulation strategy redefines a problem by introducing a decomposition, for instance, into a master problem and a set of sub-problems. Below we review some widely used reformulation strategies including, among others, Dantzig-Wolfe column generation and techniques derived from Benders Decomposition.

[15] `ipc-2020.hierarchical-task.net`

1.3.1 Problem decomposition and mathematical programming

1.3.1.1 Classical Benders Decomposition

Benders Decomposition [16, 51] is a technique that reformulates and solves an optimization problem as a two-stage problem. The first stage considers a first subset of variables to solve a *master problem* and the second stage considers a *sub-problem* (or auxiliary problem) which involves a second subset of variables and is solved for each of the first stage solutions. This process seeks to iteratively generate new constraints called Benders cuts for the master problem. These Benders cuts prune infeasible (feasibility cuts) or sub-optimal (optimality cuts) first stage decisions.

Classical Benders Decomposition approaches are mainly addressed to deal with linear problems with a special block structure which allows them to naturally decompose the problem. However, classical Benders Decomposition approaches have been also studied to address for instance, CP problem formulations [41], and extended to address several kinds of optimization problems (more details in Section 1.3.3).

1.3.1.2 Column generation algorithms

In practice, some optimization problems involve a huge number of variables. The column generation or *delayed* column generation algorithms [46] work with a subset of variables or *columns* which potentially improve the objective function, and generate missing variables only when they are needed, seeking to improve the tractability of large-scale problems. The global problem is then decomposed into a *master problem* which only considers a subset of columns, and a *sub-problem*, which iteratively identifies new needed columns to generate, based on an objective function that allows to find variable that have a negative reduced cost. Column generation approaches are mostly used to deal with large MILP problems, but they have been also studied in the CP field [75].

The well-known **Dantzig–Wolfe decomposition algorithm** [35] reformulates the global problem into a master problem and a set of n sub-problems to be solved with a column generation approach. It can be easily adapted to certain types of scheduling problems, such as the Multi-Machine Assignment Scheduling Problem (MMASP) [127] and similar problems [31]. Extended frameworks based on Dantzig–Wolfe decomposition include, among others, Decompose-and-Cut [120] and Branch-and-Price [117, 146] approaches.

1.3.2 Problem decomposition in Constraint Satisfaction Problems (CSPs)

A Dynamic CSP [37] is a sequence of CSPs (Section 1.1.5), each of them being a transformation of a previous one, related to constraint or domain changes, or variable additions or removals. All these changes can be expressed in terms of constraint additions or removals (relaxations) [148, 149] over the different problems. Dynamic CSPs can be considered as a sort of a problem decomposition involving several subproblems to solve in sequence. This is specially true for the so-called Conditional CSP (CCSP) framework, sometimes also referred to as a Dynamic CSP (DCSP) proposed by Mittal *et al.* [99]. The CCSP framework addresses problems whose solutions may involve different sets of variables or constraints, *e.g.* configuration problems. It contains two types of constraints: the compatibility constraints or the traditional ones of a CSP, and the activity constraints which define the conditions of activation of a variable in a final solution. These *activity constraints* are activated if the associated variable is considered (active) in the current assignment, similarly to the operating mode of the *optional interval variables* in CP Optimizer, described in Section 1.2.1. In this sense, a Conditional CSP is a *particular case* of the Dynamic CSP (Dechter and Dechter [37]), for which the transformations between the problem stages are functions of these activity constraints. Different categories of solving methods for a dynamic CSP are distinguished and differ in the way the information is transferred between the stages: *heuristic methods*, *local repair* and *constraint recording* techniques [153]. This last strategy of *constraint recording* can be somehow related to the cuts of a Benders Decomposition based approach since at each stage, some new constraints which represent an inconsistent set of decisions, will be transferred to the next CSPs in the sequence. Among the classes of constraint recording methods, we can cite *nogood recording* [129] which identifies sets of inconsistent variable assignments and branching constraints.

The dynamic CSP framework has also been studied in HTN planning to take advantage of the constraint propagation strategies of this encoding, looking for search space reduction [140].

A *different notion* of decomposition-based methods which can be encountered when dealing with a CSP [73] refer to the decomposition (or *clustering*) of constraint networks [38]. Such methods (Tree-Decomposition [123], Hypertree-Decomposition [57], ...), rely on the notion of tree-decomposition of graphs, and seek to deal with *tractable* subclasses of CSP problems by grouping variables into sets, and solving a sub-problem for each of these clustered sets.

1.3.3 Problem decomposition and hybrid optimization

Production planning and scheduling. Bi-level approaches to solve the Integrated Process Planning and Scheduling (IPPS) problems have been extensively conducted in the field of manufacturing systems and production planning. An IPPS problem is a

well-known NP-hard optimization problem, in which the notion of *planning* is not directly related to Automated Planning, but to the process plan (task routing) selection and machine assignment, mostly, addressed in manufacturing environments [11]. In this direction, planning and scheduling correspond to different decision making levels which can be naturally formulated in a bi-level hierarchical decomposition framework. The structure of this framework might contain, for instance, an upper level production planning problem dealing mainly with task allocation, and one or multiple lower level sub-problems dealing with task scheduling over resources [69, 90]. In general, a top-down solving approach is followed. Such approaches deal with the planning problem at a first stage, and use it to define the conditions to solve the scheduling problem at a later stage. To deal with large-scale scheduling problems, a wide range of hybrid methods to address problem decompositions have been studied [67]. Most of these methods concern efficient approaches integrating **Integer Programming** and **Constraint Programming** approaches [22, 60, 62, 72], by exploiting the combination of the strengths of these two methods.

Logic-Based Benders Decomposition (LBBD). Logic-Based Benders Decomposition (LBBD) [68] is a generalization of the Benders Decomposition or Classical Benders Decomposition approach. In contrast to the above-mentioned strategy of column generation, this kind of strategy can be seen as a *row generation*, which instead of iteratively adding variables, adds new constraints to the master problem.

LBBD is not only adapted to linear or non-linear programming problems, but also to a wide variety of large-scale optimization problems [71], including fundamental classes of production planning and scheduling problems. In this last kind of problems, each of the stages is often solved using different methods which are adapted to the structure of each stage. In this context, an LBBD algorithm can be used to link both stages [33, 69, 70] which may correspond for instance, to a first stage MILP formulation to solve the production planning assignment and a second stage CP formulation to solve the scheduling sub-problem [28].

Variants of LBBD include for instance, *Branch-and-Check* [142] frameworks, where the master problem is generally harder than the sub-problems, and is solved only once and *checked* by the subproblems. Similar approaches involve decomposition frameworks able to generate precise *no-goods* in CP [17, 27], which are close to Benders cuts.

1.3.4 Simulation-Based Optimization (SBO)

Simulation Optimization (SO) or Surrogate-Based Optimization (SBO) is a technique used to deal with computationally expensive large-scale problems coming from various fields [6]. It is also known as metamodel-based optimization, black-box optimization, parametric optimization, or Optimization via Simulation (OvS). Contrary to Mathe-

matical Programming, these techniques do not rely in an *algebraic model*, but consider a sort of a *model simulation* available as a black-box [74, 76].

Simulation-based optimization can be considered as a problem decomposition strategy in which a higher level stage (a master problem) might embody a surrogate model of one or several downstream stages (or sub-problems). The finer sub-problems can be used to evaluate objective functions based on a fine grain model, and they might provide the master surrogate problem with relevant information which would allow tuning the simulation after each objective function evaluation for a particular input. Recent research has come to approaches involving multiple surrogate models simultaneously [150], to deal with the lack of model adjustment.

This technique has been applied to applications from different fields and a comprehensive review on Simulation Optimization including a classification of the algorithms available was conducted by Amaran *et al.* [6]. Among the best-known classes of algorithms presented in this survey, we find metaheuristics, gradient-based methods and Response Surface Methodologies (RSMs). This last category might include, among other approximation forms, Kriging, Radial Basis Functions (RBFs), and neural networks.

Nevertheless, the potential of Simulation-Based Optimization has not been extensively studied in the fields of planning or scheduling. Hao *et al.* [59] propose a surrogate-modelling based approach to address a bottleneck stage scheduling problem, which is decomposed into an *expensive-to-evaluate* assignment sub-problem (AP) and a sequencing sub-problem (SP). This approach introduces a surrogate model based differential evolution (SM-DE) to *crudely* estimate the corresponding objective of the *time-consuming* AP. The SM-DE iteratively runs with a branch-and-bound procedure to solve the SP, generating new solutions (job sequences for each machine) further used as sample data to incrementally adjust the surrogate model. Also, an improved adaptive proximity-based mutation strategy balances exploration/exploitation trade-off during the evolutionary process of the SM-DE.

1.4 Hierarchical Scheduling Problems (HSPs)

An HSP can be defined by reusing the previously introduced concepts of WBS and HTN planning.

At each *downstream level* of the hierarchical structure describing an HSP, a group of *subtasks* to be performed can be *abstracted* based on the system granularity (detail of the discriminations about the system [155]), or can be *reduced* to a small number of tasks at the next upper level, decreasing the computational cost of each of the reduced problems [125]. For instance, a classical JSSP can be seen as a simple 2-level hierarchical scheduling problem, in which the jobs to execute (top-level tasks) are decomposed into

sets of operations (subtasks) to perform along the available machines. More details on this point will be given in Chapter 3, in which a first modelling framework of an HSP is introduced (Section 3.2.1).

On its hierarchical structure, an HSP may also involve several execution alternatives or decomposition methods to the execution of the tasks. Thus, as an HTN planning structure, an HSP can be considered as a *more flexible* structure which, contrarily to a WBS, is not fully unfolded. More details on this point can be found in Chapters 4 and 5.

Chapter 2

A hierarchical multi-robot deployment case study

Contents

2.1 Introduction

In this chapter, a case study regarding a Multi-Robot Deployment (MRD) application is introduced. A Hierarchical Scheduling Problem (HSP) based on different formats of this MRD application has been studied during the course of this thesis. This problem involves *a set of robots* which must *navigate through shared areas* of a field and perform *exploration tasks* (or *observations*) at different locations.

The MRD application has been widely addressed to tackle problems mainly related to *situation awareness issues*, such as cooperative sensing using air-ground teams [30], disaster response [122], and exploration-rescue systems in hostile environments [139]. In the OR field, similar applications have been addressed. This is the case of integrated scheduling and routing problems [151], requiring coordination of production activities

and transportation activities (the PTSP[1] problem). Similar problems include extensions of the classical RCPSP which may involve transfer times (RCPSPTT) [119] or routing (RCPSPR) [151]. Other applications coming from different fields might also be adapted and tackled as sorts of MRD problems.

This chapter is organized as follows. Section 2.2 presents an overview of the MRD application including a detailed description of the case studies we consider. Section 2.3 presents the associated Hierarchical Scheduling Problem (HSP) along with an encoding in Constraint Programming. Section 2.4 details several anti-collision management mechanisms to avoid robot conflicts when executing the MRD mission. Finally, Section 2.5 concludes this chapter.

2.2 Overview of the multi-robot deployment application

2.2.1 Description of a general MRD case study

In an MRD problem, a **fleet of robots** must perform a set of **observation requests** over specific areas of a field. The robots cannot perform more than one observation at a time (disjunctive resources). They must also **transfer observation data in real time** to a mission center, and for this purpose each robot uses a specific **emission frequency** related to the nature of the observation performed. To avoid interferences, two robots that use the same frequency cannot transfer observation data in parallel (disjunctive resource). **Redundancy** is also useful in this kind of application, therefore some observation targets (or requests) must be observed by several distinct robots and they must be spaced out by a certain amount of time (**revisit delay**). Moreover, some **precedence constraints** can be imposed over the observations to perform.

To represent the structure of the field which is **shared between all robots**, a graph of connected waypoints as the one in Figure 2.1 is used. In this figure, two **atomic observations** (O_1, O_2) must be performed at the **observation target (area or request)** RQ_1, two **atomic observations** (O_4, O_5) at the observation target RQ_3, and only one observation (O_3) at the target RQ_2.

In the MRD problem, **several candidate paths** are considered to navigate between pairs of observation areas. For each alternative path, the movements of robots between observation locations can be broken down into successive movements on **links** between pairs of adjacent **waypoints**. In a first *simple* anti-collision mechanism to avoid conflicts between the robots in the shared network, each link and each waypoint cannot be occupied by more than one robot at a time (disjunctive resources). Finally, the objective of the mission is to perform **all the observations as quickly as possible**.

[1] Production and Transportation Scheduling Problem

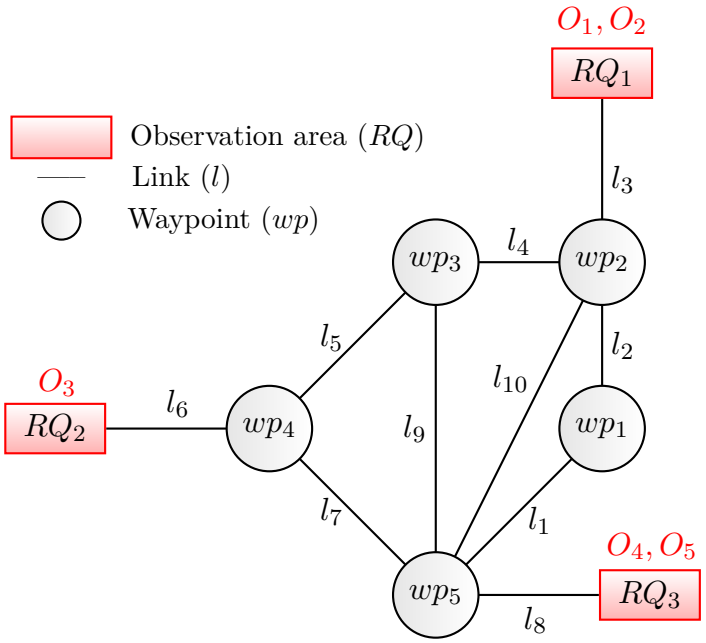

Figure 2.1: Graph of connected waypoints to represent the structure of the field.

2.2.2 Generation of MRD problem instances

In order to test the different strategies that were studied throughout this thesis, benchmarks around the described MRD application have been considered and will be further developed in Chapters 3 and 4.

A first problem instance generator was defined for this type of benchmark and it takes as parameters the following elements:

- **Observations features:**
  - the number of *observation areas* or *observation requests*,
  - the number of redundant *observation activities* required to observe each area,
  - a *minimum temporal distance* between the atomic observations performed in each area,
  - the *duration* of each of the observation requests,
  - the *precedence constraints* which can be imposed over the atomic observations or over the observation requests to perform,
- **Robots features:**
  - the number of (homogeneous or heterogeneous) *robots available* to explore the field and perform area observations, and for each robot an *associated frequency* to transmit data during an observation,

- **Field features:**
  - the total *number of crossing points* (waypoints) in the deployment area,
  - the *number of connections* (links) between each pair of waypoints and a maximum distance (radius) in order to establish them,
  - the maximum *number of alternative paths* to consider between each pair of observation areas,
  - the maximum number of transfer links for a single transfer (move) of a robot between observation areas (the maximum *size of each path*),
  - the maximum speed of each robot, that can depend on the considered transfer link between way-points for each robot.

2.3 Constraint-Based Scheduling for the MRD application

The MRD problem goal is to **allocate each candidate observation** to a robot, to **schedule the sequence of observations** realized by each robot, and to **plan navigation tasks** between observation locations. A first possible approach to solve the above described MRD problem is to develop a Constraint Programming (CP) model covering all the specifications of the mission. We describe thereafter the features of this model comprising the input data, the decision variables, the constraints considered, and the objective function.

2.3.1 Input data

To define the CP model, the following input data is considered:

- a set of *frequencies* $\mathcal{F}$ available for communicating observation data in real time to the mission center;
- a set of *robots* $\mathcal{R}ob$; for each robot $r \in \mathcal{R}ob$, $freq_r \in \mathcal{F}$ is the (unique) frequency used by r to emit data during observations;
- a set of mandatory (by default) *observation areas (or requests)* $\mathcal{R}eq$, corresponding to areas of the field that must be observed; for each area $a \in \mathcal{R}eq$, $duObs_a \in \mathbb{N}$ denotes the duration required to observe a. Each request $a \in \mathcal{R}eq$ is defined by a release date rd_a (after which it can start), a due date dd_a (before which it must end);
- a number *NobsPerArea* of observations required over each observation area; all observations of a given area must be performed by distinct robots for redundancy issues.

- a set of *observations* $\mathcal{O}bs$ to be performed, which contains as many elements as the number of (a, k) pairs in $\mathcal{R}eq \times [1..NobsPerArea]$; for each observation $o \in \mathcal{O}bs$, $ar_o \in \mathcal{R}eq$ denotes the area associated with o;

- for each robot $r \in \mathcal{R}ob$, two specific observations denoted by α_r and β_r which represent virtual observations that must be performed at the beginning and at the end of the plan of r respectively; fictitious observations α_r and β_r allow us to model the initial and goal locations of r;

- a set of shared *waypoints* $\mathcal{W}$. For each robot $r \in \mathcal{R}ob$ and each waypoint $w \in \mathcal{W}$, $duMv_{r,w} \in \mathbb{N}$ is the minimum duration spent in w during a navigation of r through w; the observation areas are not considered as shared waypoints;

- a set of *links* $\mathcal{L}$, which correspond to direct connections between adjacent waypoints or between an observation area and a waypoint. For each robot $r \in \mathcal{R}ob$ and each link $l \in \mathcal{L}$, $duMv_{r,l} \in \mathbb{N}$ is the minimum duration required by r to traverse link l;

- for each robot $r \in \mathcal{R}ob$ and each pair of observations $(o, o') \in (\mathcal{O}bs \times \mathcal{O}bs \cup \{\alpha_r\} \times \mathcal{O}bs \cup \mathcal{O}bs \times \{\beta_r\})$, a set of candidate paths $P_{r,o,o'}$ which can be used by r to go from o to o'; each path p in $P_{r,o,o'}$ is specified by the sequence $[p_1, \ldots, p_Q]$ of successive network resources (waypoints or links) traversed by the path ($p_q \in \mathcal{W} \cup \mathcal{L}$ for every $q \in [1..Q]$); for instance, in Figure 2.1, several paths (path $[l_8, wp_5, l_1, wp_1, l_2, wp_2, l_3]$, path $[l_8, wp_5, l_{10}, wp_2, l_3]$, ...) depicted in Figure 2.2 are available to move from observation O_5 in area RQ_3 to observation O_1 in area RQ_1 (the details about the path notations used in Figure 2.2 are given in Section 2.3.2);

- a *minimum temporal distance* (revisit delay) d between two observations of the same area;

- a large enough *temporal horizon* $H \in \mathbb{N}$ available for the whole mission.

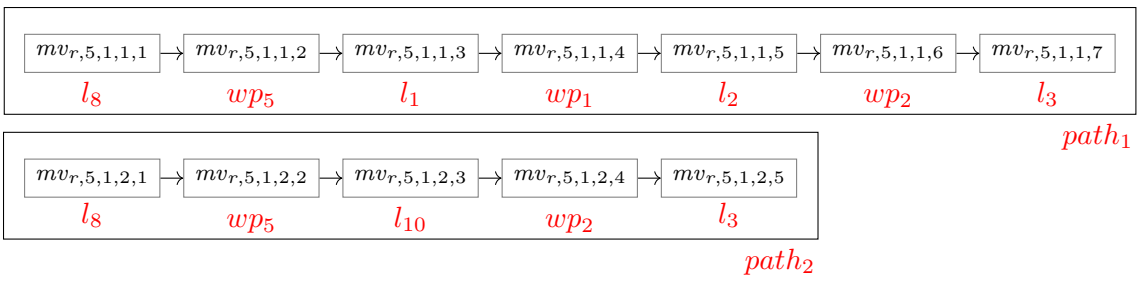

Figure 2.2: Example of sequences of navigation tasks corresponding to alternative paths between observations O_5 and O_1.

2.3.2 Decision variables

From the previous input data, a CBS model is defined and built upon the *interval variables* used in the CP Optimizer tool. If $[Ts, Te]$ denotes the time frame available for realizing the associated task, each interval variable itv is characterized by a start value $\mathbf{startOf}(itv) \in [Ts, Te]$, an end value $\mathbf{endOf}(itv) \in [Ts, Te]$, and a presence $\mathbf{pres}(itv) \in \{0, 1\}$ expressing whether the task is present in the solution schedule. Also, a so-called *sequence variable* $\mathbf{seq}_r$ is associated with each robot $r \in \mathcal{R}ob$. The value of this variable corresponds to a total ordering of all observation tasks realized by r. In the following, Tr denotes the set of all triples (r, o, o') defining a transition for a robot $r \in \mathcal{R}ob$ between a pair of candidate observations $(o, o') \in (\mathcal{O}bs \times \mathcal{O}bs \cup \{\alpha_r\} \times \mathcal{O}bs \cup \mathcal{O}bs \times \{\beta_r\})$. The following decision variables are introduced using several scheduling features available in the CP Optimizer tool:

- for each observation $o \in \mathcal{O}bs$, one *interval variable* obs_o which must be placed during time frame $[0, H]$ and whose duration is $duObs_{ar_o}$, that is the observation duration of the area associated with o;
- for each observation $o \in \mathcal{O}bs$ and each robot $r \in \mathcal{R}ob$, one *optional* interval variable $obs_{o,r}$ used to represent the realization of observation o by robot r;
- for each robot $r \in \mathcal{R}ob$, two (non-optional or mandatory) interval variables $obs_{\alpha_r,r}$ and $obs_{\beta_r,r}$ representing fictitious observations that r must realize at the beginning and end of its plan respectively; we recall that these fictitious observations allow us to model the initial and goal locations of the robots; interval $obs_{\alpha_r,r}$ has a null duration and is placed at time 0, and interval $obs_{\beta_r,r}$ has a null duration and is placed during time frame $[0, H]$;
- for each robot $r \in \mathcal{R}ob$, one *sequence variable* $\mathbf{seq}_r$ which represents an ordering over all present intervals associated with r, *i.e.* over all present intervals in set $\{obs_{o,r} \mid o \in \mathcal{O}bs \cup \{\alpha_r, \beta_r\}\}$;
- for each robot $r \in \mathcal{R}ob$ and each pair of candidate observations (o, o') such that $(r, o, o') \in Tr$, the following variables are considered:
  - for each candidate path $p \in P_{r,o,o'}$, one optional interval variable $mv_{r,o,o',p}$ representing a move along path p;
  - for each candidate path $p = [p_1, \ldots, p_Q] \in P_{r,o,o'}$ and each index $q \in [1..Q]$, one optional interval variable $mv_{r,o,o',p,q}$ representing the usage of the qth network resource of path p (see the example in Figure 2.2).

The previous *variables* represent the decision to be made in the considered problem. Also, *expressions* can be built upon those decision variables: for each robot $r \in \mathcal{R}ob$ and each pair of candidate observations (o, o') s.t. $(r, o, o') \in Tr$, $\mathbf{next}_{r,o,o'} \in \{0, 1\}$

denotes an expression taking value 1 if the interval $obs_{o,r}$ is the predecessor of interval $obs_{o',r}$ in the sequence of intervals $\mathbf{seq}_r$ associated with robot r.

2.3.3 Objective and constraints

The constraints 2.1a to 2.1m introduced below are imposed over the considered variables.

Constraint 2.1a imposes that the first and last observations in a robot sequence must correspond to the initial and final fictitious observations. Constraints 2.1b uses the *alternative* constraint of CP Optimizer and expresses that each observation is realized by a unique robot. Constraint 2.1c imposes that each robot can realize at most one observation for a given area (redundancy requirement). Constraint 2.1d expresses that observation tasks using the same frequency cannot overlap. Constraint 2.1e expresses that observations of a given area cannot overlap, taking into account the minimum delay d defined in the input data. Constraint 2.1f forbids the temporal overlapping of tasks that use the same link. Constraint 2.1g states the equivalence between moves presence and chains of observations in the sequences of each robot, ensuring that exactly one path is used between each pair of successive observations. Constraint 2.1h expresses that each path spans all its elementary moves. Constraint 2.1i states that the presences of elementary moves must be consistent with the presences of the selected paths. Constraints 2.1j to 2.1l enforce the chaining between the successive intervals involved in a chosen navigation path. Constraint 2.1m defines the minimum duration of each elementary move interval. An inequality is considered here since a robot is

allowed to wait on a link or at a waypoint.

$$\forall r \in \mathcal{R}ob,\ \mathit{first}(\mathbf{seq}_r, obs_{\alpha_r,r}) \wedge \mathit{last}(\mathbf{seq}_r, obs_{\beta_r,r}) \quad (2.1a)$$

$$\forall o \in \mathcal{O}bs,\ \mathit{alternative}(obs_o, \{obs_{o,r} \mid r \in \mathcal{R}ob\}) \quad (2.1b)$$

$$\forall a \in \mathcal{R}eq,\ \forall r \in \mathcal{R}ob,\ \textstyle\sum_{o \in \mathcal{O}bs \mid ar_o = a} \mathbf{pres}(obs_{o,r}) \leq 1 \quad (2.1c)$$

$$\forall f \in \mathcal{F},\ \mathit{noOverlap}(\{obs_{o,r} \mid o \in \mathcal{O}bs,\ r \in \mathcal{R}ob,\ \mathit{freq}_r = f\}) \quad (2.1d)$$

$$\forall a \in \mathcal{R}eq,\ \mathit{noOverlap}(\{obs_{o,r} \mid o \in \mathcal{O}bs,\ r \in \mathcal{R}ob,\ ar_o = a\}, d) \quad (2.1e)$$

$$\forall \lambda \in \mathcal{L},\ \mathit{noOverlap}(\{mv_{r,o,o',p,q} \mid (r,o,o') \in Tr) \wedge (p \in P_{r,o,o'}) \wedge (q \in [1..|p|]) \wedge (p_q = \lambda)\}) \quad (2.1f)$$

$$\forall (r,o,o') \in Tr,\ \mathbf{next}_{r,o,o'} = \textstyle\sum_{p \in P_{r,o,o'}} \mathbf{pres}(mv_{r,o,o',p}) \quad (2.1g)$$

$$\forall (r,o,o') \in Tr, \forall p \in P_{r,o,o'},\ \mathit{span}(mv_{r,o,o',p}, \{mv_{r,o,o',p,q} \mid q \in [1..|p|]\}) \quad (2.1h)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \forall q \in [1..|p|],\ \mathbf{pres}(mv_{r,o,o',p}) = \mathbf{pres}(mv_{r,o,o',p,q}) \quad (2.1i)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \forall q \in [2..|p|], \quad (2.1j)$$
$$\mathbf{pres}(mv_{r,o,o',p}) \rightarrow \mathit{endAtStart}(mv_{r,o,o',p,q-1}, mv_{r,o,o',p,q})$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \mathbf{pres}(mv_{r,o,o',p}) \rightarrow \mathit{endBeforeStart}(obs_{o,r}, mv_{r,o,o',p}) \quad (2.1k)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \mathbf{pres}(mv_{r,o,o',p}) \rightarrow \mathit{endBeforeStart}(mv_{r,o,o',p}, obs_{o',r}) \quad (2.1l)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \forall q \in [1..|p|], \quad (2.1m)$$
$$\mathbf{pres}(mv_{r,o,o',p,q}) \rightarrow (\mathbf{endOf}(mv_{r,o,o',p,q}) - \mathbf{startOf}(mv_{r,o,o',p,q}) \geq duMv_{r,p_q})$$

Finally, the objective is to minimize the makespan, defined as the time at which each robot reaches its goal position:

$$\text{minimize} \max_{r \in \mathcal{R}ob} \mathbf{endOf}(obs_{\beta_r,r}) \quad (2.2)$$

2.4 Anti-collision mechanisms

As stated before, one specificity of the network defining the observations field is that it is **shared between all robots**. To *avoid collisions* during traversals of the network, or *at least to reduce the need to deal with collision situations online*, we have considered in the previous model that each link can be occupied by at most one robot at a time (disjunctive resources). A finer approach could be done by considering a non-unit capacity, but this would require handling cumulative resources which are not considered at this stage in this thesis.

Different alternatives for the modelling of the MRD problem have been considered,

according to the network resources that can be shared and according to the time frame during which each robot move monopolizes those shared resources. The three different approaches proposed are discussed thereafter.

They differ in terms of robustness and in terms of required synchronization between the robots at execution time, where duration of robot moves can be shorter or longer that expected.

2.4.1 The simplest case: link isolation

In this first anti-collision approach, the locations associated with observation areas and the waypoints are considered as shareable. The rationale for this assumption is that the robots have a smaller (or even null) speed when performing observations or turning (respectively) at these locations, so the online management of collisions is easier and less hazardous in this case than when the robot moves on links. This approach is the one used in the previous CP encoding.

2.4.2 Link and waypoint isolation and minimum handover

In this second approach, the only locations that are considered as shareable are those associated with observation areas, considering, as in the previous approach, the ease of online collision management since the robots have a smaller (or null) speed when performing observations at these locations.

This second approach is illustrated in Figure 2.3, where a robot r successively consumes the network resources involved in a path for a transition between observations o and o' successively using link l_1, waypoint wp, and link l_2. The resource consumption for each of those network resources is illustrated as well in the figure. One specificity in this case is that there exists a positive time lapse, called the **handover duration**, during which a robot switches between network resources (move between a link and a waypoint for example). During each handover period, the robot moves to the next resource on its path, but must also remain "connected" to the previous one, to prevent robots from "jumping" between network resources. The goal is to forbid inconsistent solutions where two robots would instantaneously cross each other over the network (*e.g.* solutions where at a given time t, one robot instantaneously moves for link l to waypoint w while another one instantaneously moves from w to l).

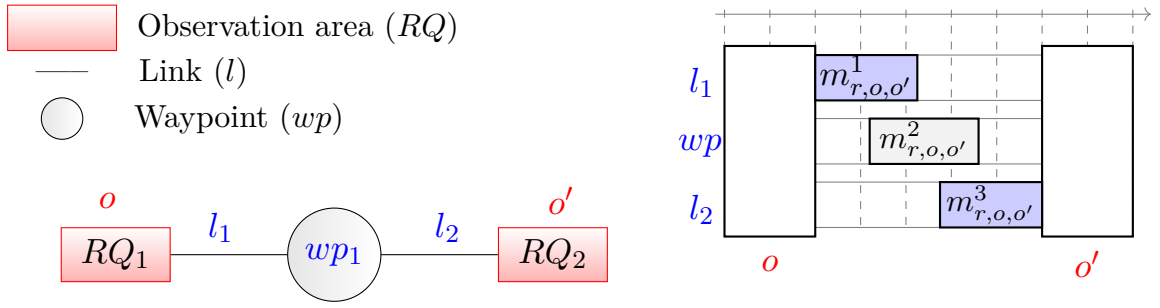

Figure 2.3: Network resources consumption for the Minimum Handover case.

2.4.3 Path isolation

In this third approach, for a transition of robot r between two observations o and o', all path resources used during the transition are reserved for the whole move duration, as illustrated in Figure 2.4.

With the previous strategy (Minimum Handover), the usage of network resources is more finely optimized but there is a need to synchronize the robots at each basic move. On the contrary, the Path Isolation strategy takes more margins to get a collision-free deployment, but it only requires synchronizing the robots at the start and at the end of the global moves between observation tasks, that is when the speed of robots is low. The Path Isolation approach is inspired by works dealing with inter-core interferences due to shared hardware resources in multi-core processors [55, 154] to temporally isolate hard real-time applications [112].

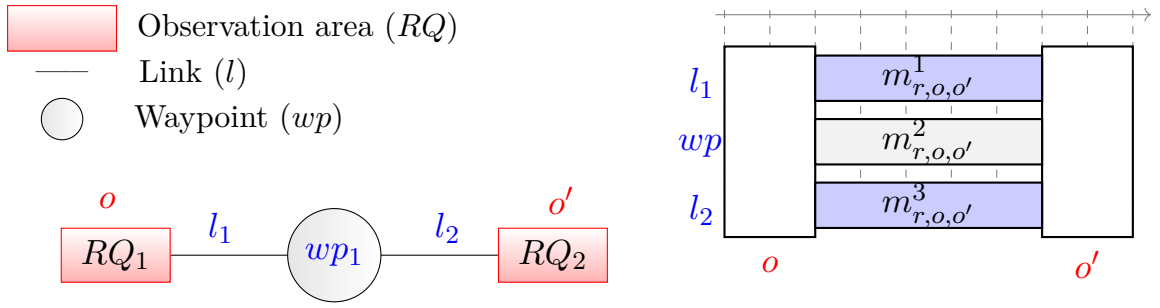

Figure 2.4: Network resources consumption for the Path Isolation case.

2.5 Conclusion

This chapter introduced a Multi-Robot Deployment (MRD) case study, in which a set of robots must navigate through shared areas of a field and perform exploration tasks (or observations) at different locations as fast as possible.

The general case study was presented, along with the associated Hierarchical Scheduling Problem (HSP) and an encoding in Constraint Programming. Finally, several alternative representations for the MRD problem comprising anti-collision mechanisms were introduced. These different approaches will be more extensively discussed in the following chapters.

Chapter 3

A first decision method based on task abstraction and iterative task decomposition

Contents

3.1 Introduction

This chapter aims to provide a first framework to model some classes of Hierarchical Scheduling Problems (HSPs), along with preliminary strategies to address such problems. This novel framework reuses some notions from the previously described concepts of WBS and HTN planning including compound tasks that cover several subtasks. It is introduced in Section 3.2 along with the formal definition of the type of HSPs considered in this chapter, and a first interpretation of this framework in terms of Constraint Programming. To deal with HSPs that can be modelled under this framework, a first *decision method* based on abstraction and decomposition of the problem tasks is introduced in Sections 3.3 and 3.4, together with several associated heuristic rules and configuration parameters. The experimental results obtained from the implementation of the proposed refinement strategy for different types of representative problem instances, including well known benchmarks and instances of a Multi-Robot Deployment (MRD) application, are discussed in Section 3.5. Finally, Section 3.6 provides the conclusions of the work presented in this chapter and gives some insights from this preliminary results to be used in further related research.

The contributions introduced in this chapter were presented at the beginning of the thesis in the French conference "Journées Francophones de Programmation par Contraintes (JFPC)" [106].

3.2 Hierarchical Scheduling Problems

In this section we introduce a first modelling framework for handling Hierarchical Scheduling Problems defined by **(1)** a set of tasks that can be atomic (*i.e.* directly performable) or compound (*i.e.* decomposed into a subset of tasks), **(2)** a set of disjunctive resources available to perform them, and **(3)** the minimization of the makespan while taking into account temporal constraints and constraints on the availability of the required resources.

3.2.1 A first framework to model Hierarchical Scheduling Problems

3.2.1.1 Definition

A first modelling framework of a Hierarchical Scheduling Problem can be stated by reusing the concepts of WBS and HTN planning introduced in Chapter 1. A Hierarchical Scheduling Problem P is defined as a tuple $(\mathcal{R}, U)$, where

- $\mathcal{R}$ is a set of **disjunctive resources** which can only perform one activity at a

time,

- U is a set of tasks,

The set of tasks U is partitioned into two subsets $\mathcal{A} \subseteq U$ and $\mathcal{C} \subset U$.

- Subset $\mathcal{A} \subseteq U$ corresponds to a subset of **atomic tasks** or activities to perform. Each atomic task $a \in \mathcal{A}$ is defined by
  - a **duration** du_a,
  - a set of required **resources** $\mathcal{R}_a \subseteq \mathcal{R}$ consumed during the task execution.
- Subset $\mathcal{C} \subset U$ corresponds to a set of **compound tasks**. Each compound task $c \in \mathcal{C}$ is defined by
  - a set of **subtasks** $SubTask_c \subset \mathcal{C} \cup \mathcal{A}$,
  - a set of required **resources** $\mathcal{R}_c \subseteq \mathcal{R}$ consumed as long as the compound task is active, that is, from the start date of the first subtask associated with c to the end date of the last subtask associated with c,
  - a set of acyclic **precedence constraints** $\mathcal{P}_c \subset SubTask_c \times SubTask_c$ between the subtasks of c.

In the following, for any resource $r \in \mathcal{R}$, U_r denotes the set of tasks τ (atomic or compound) that use r, *i.e.* such as $r \in \mathcal{R}_\tau$.

3.2.1.2 Additional assumptions

The Hierarchical Scheduling Problems addressed in this chapter are assumed to be *well formed*. We assume that:

- the task decomposition graph is acyclic. Formally, this graph corresponds to the directed graph whose nodes are the tasks of $\mathcal{A} \cup \mathcal{C}$, and which contains for each compound task c and each subtask $\tau \in SubTask_c$ an arc $c \rightarrow \tau$,
- each atomic or compound task appears as a subtask of at most one task. This implies that the task decomposition graph defines a forest of tasks,
- a resource consumed by a compound task c is not consumed by a task τ descendent of c. Formally, for any compound task c and any task τ such that there is a path $c \rightarrow \tau$ in the task decomposition graph, $\mathcal{R}_c \cap \mathcal{R}_\tau = \emptyset$.

3.2.1.3 Scope of this modelling framework

The resulting model involves hierarchical decompositions. It allows to easily represent problems such as the Job Shop Scheduling Problem (JSSP) or the Open Shop Scheduling Problem (OSSP) [113], with in this case only one level of decomposition (one compound task per job). In the case of the JSSP, the HSP modelling contains the precedence constraints defining the sequences of activities associated with the different jobs. In the case of the OSSP where the order of tasks to perform in a job is left free, the HSP modelling also involves an additional dummy resource to prevent tasks in the same job from being performed in parallel.

Finally, it is worth pointing out that the resource consumptions can be directly associated with the compound tasks, which allows to model scenarios involving resources monopolized during the whole duration of a set of subtasks, *e.g.* an scenario involving an operator having to perform a set of works *alone*, but having to wait for tools shared between several operators to be available.

3.2.1.4 Multi-Robot Deployment example

As stated earlier, a Hierarchical Scheduling Problem inspired by a Multi-Robot Deployment (MRD) application has been studied during the course of this thesis. In this application, robots must *navigate through a shared field* and perform observations in sequence. The complete description of this kind of problem was presented in the previous chapter.

In this chapter that concerns the first research works of this thesis, we consider a simple case in which only the links of the field are considered as disjunctive network resources, thus they cannot be occupied by more than one robot at a time. A toy example associated with a variant of this application is illustrated in Figure 3.1, in which:

- two zones Z_1 and Z_2 must be respectively explored by two robots, r_1 and r_2,
- to observe zone Z_1, robot r_1 must first carry out observation O_1 and retransmit it on frequency f_1, then navigate (movement M_{12}) to carry out O_2 while transmitting data on frequency f_2,
- move M_{12} is broken down into 3 steps: move M_{12}^1 on link l_1, move M_{12}^2 on link l_2 and move M_{12}^3 on link l_3,
- a similar breakdown is made for zone Z_2.

With the above defined framework, this toy example is modelled as follows:

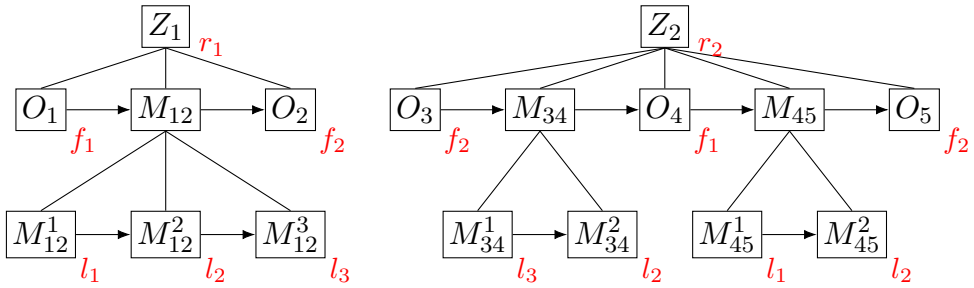

Figure 3.1: Multi-robot deployment application.

- The set $\mathcal{R}$ of disjunctive resources corresponds to
  - $\mathcal{R} = \{r_1,\ r_2,\ f_1,\ f_2,\ l_1,\ l_2,\ l_3\}$,
- the sets $\mathcal{A}$ and $\mathcal{C}$ of atomic and compound tasks are
  - $\mathcal{A} = \{O_1(2),\ O_2(3),\ O_3(3),\ O_4(1),\ O_5(2),\ M^1_{12}(1),\ M^2_{12}(2),\ M^3_{12}(1),\ M^1_{34}(2),\ M^2_{34}(1),\ M^1_{45}(2),\ M^2_{45}(1)\}$ (the duration of each activity is specified in parenthesis),
  - $\mathcal{C} = \{Z_1,\ Z_2,\ M_{12},\ M_{34},\ M_{45}\}$,
- For each zone, the subtasks correspond to
  - $SubTask_{Z_1} = \{O_1,\ M_{12},\ O_2\}$,
  - $SubTask_{Z_2} = \{O_3,\ M_{34},\ O_4,\ M_{45},\ O_5\}$,
- and the precedence constraints correspond to
  - $\mathcal{P}_{Z_1} = \{(O_1,\ M_{12}),\ (M_{12},\ O_2)\}$,
  - $\mathcal{P}_{Z_2} = \{(O_3,\ M_{34}),\ (M_{34},\ O_4),\ (O_4,\ M_{45}),\ (M_{45},\ O_5)\}$.
- For each compound navigation task (movement), the subtasks correspond to
  - $SubTask_{M_{12}} = \{M^1_{12},\ M^2_{12},\ M^3_{12}\}$,
  - $SubTask_{M_{34}} = \{M^1_{34},\ M^2_{34}\}$,
  - $SubTask_{M_{45}} = \{M^1_{45},\ M^2_{45}\}$,
- and the precedences correspond to
  - $\mathcal{P}_{M_{12}} = \{(M^1_{12},\ M^2_{12}),\ (M^2_{12},\ M^3_{12})\}$,
  - $\mathcal{P}_{M_{34}} = \{(M^1_{34},\ M^2_{34})\}$,
  - $\mathcal{P}_{M_{45}} = \{(M^1_{45},\ M^2_{45})\}$.
- Finally, the sets of tasks consuming each disjunctive resource are
  - $U_{r_1} = \{Z_1\}$ and $U_{r_2} = \{Z_2\}$,
  - $U_{f_1} = \{O_1,\ O_4\}$ and $U_{f_2} = \{O_2,\ O_3,\ O_5\}$,
  - $U_{l_1} = \{M^1_{12},\ M^1_{45}\}$, $U_{l_2} = \{M^2_{12},\ M^2_{34},\ M^2_{45}\}$ and $U_{l_3} = \{M^3_{12},\ M^1_{34}\}$.

3.2.1.5 Solution of a Hierarchical Scheduling Problem

A solution of an HSP associates a start date s_τ and an end date e_τ with each task τ of $\mathcal{A} \cup \mathcal{C}$, so that several constraints are satisfied. For each activity (or atomic task), the end date is given by the sum of its start date and its duration (Equation 3.1a), and the start and end dates of a compound task c must cover exactly its subtasks (Equations 3.1b and 3.1c). Also, two tasks consuming the same resource cannot overlap in time (Equation 3.1d). Finally, a precedence between two tasks implies that the end date of the first task must be less than or equal to the start date of the second one (Equation 3.1e).

$$\forall a \in \mathcal{A},\ e_a = s_a + du_a \tag{3.1a}$$
$$\forall c \in \mathcal{C},\ s_c = \min\{s_\tau \,|\, \tau \in SubTask_c\} \tag{3.1b}$$
$$\forall c \in \mathcal{C},\ e_c = \max\{e_\tau \,|\, \tau \in SubTask_c\} \tag{3.1c}$$
$$\forall r \in \mathcal{R}, \forall \tau \neq \tau' \in U_r^2,\ (e_\tau \leq s_{\tau'}) \vee (e_{\tau'} \leq s_\tau) \tag{3.1d}$$
$$\forall c \in \mathcal{C},\ \forall (\tau, \tau') \in \mathcal{P}_c,\ s_{\tau'} \geq e_\tau \tag{3.1e}$$

A solution is said to be optimal if it minimizes the *makespan*, defined as the end date of the last task in the plan ($\max\{e_\tau \,|\, \tau \in \mathcal{A} \cup \mathcal{C}\}$).

Figure 3.2 illustrates the resource consumptions associated with a solution of the toy example, meeting the considered constraints. On this figure, tasks M_{12}, M_{34} and M_{45} do not appear because they do not directly consume resources. Their start and end dates are: $s_{M_{12}} = 2$, $e_{M_{12}} = 6$, $s_{M_{34}} = 3$, $e_{M_{34}} = 6$, $s_{M_{45}} = 7$, $e_{M_{45}} = 10$.

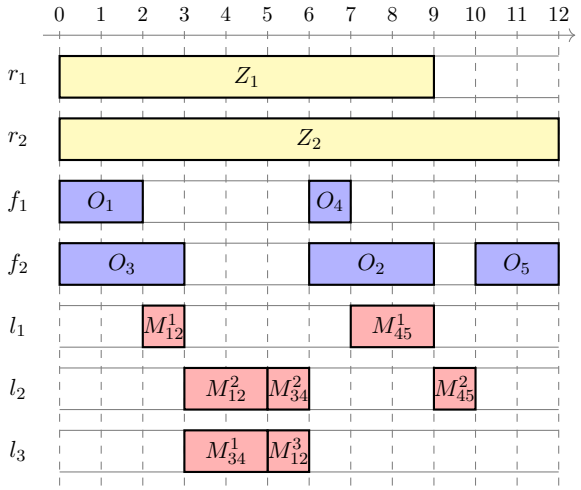

Figure 3.2: Resource consumptions for the MRD example.

3.2.2 Encoding in Constraint Programming

The Constraint Programming (CP) encoding of the defined HSP is relatively straightforward. For example, it is possible to reuse the scheduling primitives available in the IBM ILOG tool CP Optimizer, some of which can be used to define task hierarchies [83, 84].

The simple resulting model is presented below. First, for each task τ of the problem, a time interval itv_τ (an *interval variable* in CP Optimizer) is used. We recall that, each interval variable itv is defined by a value **startOf**(itv) representing the start date of the interval, a value **endOf**(itv) representing the end date of the interval, and a length *lengthOf*(itv) providing the distance between the start and end of the interval. For the interval variables which are associated with atomic tasks, this distance is known and is equal to the duration of the activity (see Equation 3.2a). For the intervals associated with compound tasks, this distance is not initially known (see Equation 3.2b), and it is only possible to specify that these intervals must end before a considered maximum time horizon T. The above described interval variables correspond to

$$\forall a \in \mathcal{A}, \textbf{ dvar interval } itv_a \text{ size } du_a \text{ in } [0..\mathrm{T}], \tag{3.2a}$$

$$\forall c \in \mathcal{C}, \textbf{ dvar interval } itv_c \text{ in } [0..\mathrm{T}], \tag{3.2b}$$

The model also contains the different constraints that were listed previously in Equations 3.1a to 3.1e. The formalization contains in particular the *span* constraint specifying that an interval must cover exactly one set of intervals, the *noOverlap* constraint imposing a non-overlap in time between a set of intervals, and the *endBeforeStart* constraint imposing that the end date of an interval must be less than or equal to the start date of another interval. This model of the HSP can then be solved using the CP Optimizer tool. The above described constraints correspond to

$$\forall c \in \mathcal{C},\ span(itv_c, \{itv_\tau \,|\, \tau \in SubTask_c\}), \tag{3.3a}$$

$$\forall r \in \mathcal{R},\ noOverlap(\{itv_\tau \,|\, \tau \in U_r\}), \tag{3.3b}$$

$$\forall c \in \mathcal{C},\ \forall (\tau, \tau') \in \mathcal{P}_c,\ endBeforeStart(itv_\tau, itv_{\tau'}). \tag{3.3c}$$

3.3 Abstraction of a compound task

The CP method presented above is based on a **completely unfolded version** of the hierarchical scheduling problem. Such complete unfolding can be quite **costly in terms of computation time** as the number of atomic and compound tasks increases. To ease scaling and guide the resolution on higher levels, a method to avoid the systematic unfolding of all tasks at the beginning of the search is proposed. This method reasons on *abstractions of compound tasks* in a first instance, and then *refines* these abstractions

step by step when necessary, gradually returning to a non-abstract modelling for some of the compound tasks.

In this context, *abstracting* a compound task means representing, in a non-detailed way, all its subtasks, and reasoning more broadly by **estimating the overall impact of these subtasks** on the scheduling problem to be solved. More precisely, each compound task c is abstracted by an atomic task denoted by $Abs(c)$. The latter is defined by a duration $du_{Abs(c)}$ and by a set of resources $\mathcal{R}_{Abs(c)}$ consumed during the whole duration of $Abs(c)$.

Abstractions are computed starting from atomic tasks (for any atomic task $a \in \mathcal{A}$, $Abs(a) = a$) and propagated progressively to the higher level tasks in the hierarchy of tasks, *i.e.* they are built following a *bottom-up* traversal of the task hierarchy. The techniques used to define the abstraction features $du_{Abs(c)}$ and $\mathcal{R}_{Abs(c)}$ are defined thereafter.

3.3.1 Duration of the abstraction of a compound task

To define the duration $du_{Abs(c)}$ associated with the abstraction of a compound task $c \in \mathcal{C}$, it is possible to schedule the abstractions of the subtasks of c independently of the other compound tasks in the problem. This scheduling subproblem Pb contains atomic task $Abs(\tau)$ for each subtask $\tau \in SubTask_c$, and a precedence constraint $Abs(\tau) \rightarrow Abs(\tau')$ for each precedence constraint $\tau \rightarrow \tau' \in \mathcal{P}_c$.

It is assumed that there is an available procedure capable of **quickly producing a solution** S to problem Pb. This procedure could be, for instance, the use of a *heuristic rule* that inserts the tasks one after the other in the plan following an order of insertion function of the features of the tasks of the problem. When problem Pb is a *simple* problem, it can also be solved using a complete solution technique able to produce a schedule that minimizes the *makespan*. If the precedence constraints associated with c are such that all subtasks of c must be performed in sequence, obtaining a solution S that minimizes the makespan is immediate (in this case, the optimal makespan is $\sum_{\tau \in SubTask_c} du_{Abs(\tau)}$)

The solution S found by scheduling the abstractions of the subtasks of the compound task c can then be used to associate with the abstraction of c a duration $du_{Abs(c)}$ equal to the *makespan* of solution S. Intuitively, this abstraction is pessimistic in the sense that it considers a duration that is anyway sufficient to complete the entire compound task c.

Figure 3.3 shows an initial compound task and a plan minimizing the *makespan* that can be obtained directly, given the precedence constraints. The solution depicted in Figure 3.3(b) would give a duration equal to 4 time units with the chosen abstraction.

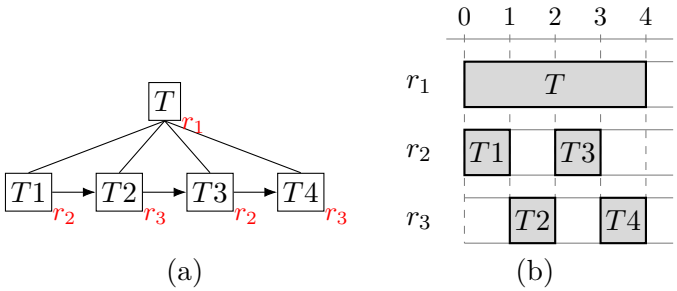

Figure 3.3: (a) initial compound task T with 4 subtasks of duration 1, for a scheduling problem involving 3 disjunctive resources r_1, r_2, r_3; (b) solution schedule for the problem.

3.3.2 Resource consumption for the abstraction of a compound task

To abstract the resource consumptions of the subtasks of a compound task c, two alternatives are considered:

- a first alternative, denoted as $C0$, is given in Equation 3.4a; it includes in the consumptions of $Abs(c)$ only the resources of $\mathcal{R}_c$ that are **consumed directly** by c;
- a second alternative, denoted as $C1$, is given in Equation 3.4b; it adds to the consumptions of $Abs(c)$ the set of resources **consumed by the abstractions** of the subtasks of c.

$$C0 \ : \mathcal{R}_{Abs(c)} = \mathcal{R}_c \tag{3.4a}$$
$$C1 \ : \mathcal{R}_{Abs(c)} = \mathcal{R}_c \cup (\cup_{\tau \in SubTask_c} \mathcal{R}_{Abs(\tau)}) \tag{3.4b}$$

Intuitively, the consumption abstraction version $C0$ is a rather optimistic version which considers that in any case, the resource consumptions of the subtasks will not be limiting for scheduling (on the example of the last level of hierarchy for the robot deployment problem, this approach is equivalent to considering as a first approximation that the network resources will not slow down robot movements).

The consumption abstraction version $C1$ is more pessimistic (or *robust*) in the sense that it reserves all the resources that could be consumed by the subtasks during the execution of the task abstraction.

Thus, two abstraction methods noted respectively Abs^{C0} and Abs^{C1} are obtained. These abstraction methods are illustrated in Figure 3.4. The Abs^{C1} approach leads to task schedules that do not contain any *resource conflict* between subtasks of different compound tasks. It also ensures that the schedule built on the basis of abstractions

can be extended to a complete schedule at any time. In this sense, abstraction Abs^{C1} is *robust*. In contrast, abstraction Abs^{C0} is *optimistic*.

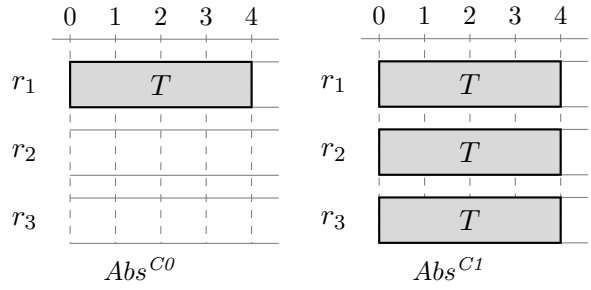

Figure 3.4: Resource consumption of the different forms of abstraction obtained from the solution displayed in Figure 3.3.

Finally, these resulting abstractions reason implicitly as if subtasks of different tasks could not be *interleaved* with each other on resources, i.e. by not considering a schedule in which a resource is used first by the subtasks of one compound task c_1, then by the subtasks of another compound task c_2, and then again by the subtasks of c_1.

3.4 Iterative decomposition (refinement) strategy

A global **decomposition or refinement strategy** is used to progressively go from a plan built for a problem P_0 that contains only abstractions of high-level tasks, to a plan built for a problem P_n that corresponds to the original $(\mathcal{R}, U)$ problem. To move from a problem P_i to a refined problem P_{i+1}, the basic principle is to select at each step of the algorithm a non-empty set of compound tasks that are present in P_i in their abstract form, and to refine these tasks so that the new problem P_{i+1} obtained contains at least one less abstract task.

3.4.1 Parameters to refine a Hierarchical Scheduling Problem

Different settings need to be established to define exactly how to go from problem P_i to problem P_{i+1}.

3.4.1.1 Number of tasks to refine

The first setting to be made concerns the number of abstract tasks that are refined at each step, with possible settings being for instance:

- the selection of a single abstract task at each step,

- the selection of K abstract tasks to be refined simultaneously.

The definition of this setting will have an impact on the number of iterations to completely decompose the problem, to reach P_n.

3.4.1.2 Heuristics for the selection of tasks to be refined

The second setting to be made concerns the choice of a heuristic rule that favours the refinement of the most interesting abstract tasks. Three selection heuristics denoted respectively by H1, H2 and H3 have been studied. These heuristics are the following.

H1 This first heuristic rule focuses on abstract **tasks that have a minimal start date**. This heuristic is particularly well suited for *online approaches* in which the execution of the plan is carried out in parallel with the planning process, and where the next primitive actions of the plan must be initiated at some point in a near future.

H2 This second heuristic focuses on abstract tasks at the **highest level of the hierarchy**, *i.e.* abstract tasks associated with compound tasks whose height is maximal (where the height $h(c)$ of a compound task c is given by $h(c) = 1 + \max_{\tau \in SubTask_c}(h(\tau))$), and where the height of an atomic task a is $h(a) = 0$. This approach is to some extent analogous to the principles of a classical approach to deal with HSPs, where the higher-level decision problem is fully addressed first, before considering the finer specifications of the problem,

H3 This last heuristic focuses on abstract **tasks that appear on a critical path** in the solution plan S found for problem P_i, *i.e.* abstract tasks that have zero temporal flexibility in S. The objective here is to first focus the research on the bottleneck of the problem.

For all these heuristics, several abstract tasks might have the same heuristic value. In this case ties are broken randomly.

3.4.1.3 Information transferred between iterations

In order to make the **step-by-step resolution** of the problem of interest rather than directly reasoning about the complete problem, it is useful to **transfer information between successive resolutions**. In other words, it is useful to define methods for the first resolutions to guide subsequent ones, since the latter may be more easily trapped in certain areas of a potentially much larger search space as the refinement process is executed. Two types of information to transfer were studied:

- the transfer of **a makespan constraint** imposing that the *makespan* obtained for problem P_{i+1} must be less than or equal to the *makespan* of the solution found for problem P_i. This constraint is effectively used only when the abstractions chosen are pessimistic, because otherwise it is not guaranteed that the optimal *makespan* of problem P_{i+1} can be less than or equal to the *makespan* of problem P_i;
- the transfer of ***start-to-start* precedence constraints** between tasks, the intuition being that coarse-grain resolutions have potentially made it possible to synthesize good orders of resource usage by tasks. More precisely, starting from the solution plan found for a problem P_i, for each resource $r \in \mathcal{R}$, all the tasks of P_i which use r are extracted. These tasks are sorted by increasing start date, and for any pair of tasks τ, τ' that succeed each other in this order, a constraint $s_\tau \leq s_{\tau'}$ is added to P_{i+1}. This constraint imposes that τ starts before τ' if this constraint was not already present in P_i.

3.4.2 Iterative decomposition algorithm

Algorithm 3.1 describes the overall method that is proposed.

Algorithm 3.1: Iterative decomposition for a HSP

```
   Data: (R, U, MaxTimeIter)
   Result: S
1  P ← fullAbstractProblem(R, U);
2  S ← solve(P, MaxTimeIter);
3  while getAbsTasks(P) ≠ ∅ do
4  |  ToDecompose ← selectAbsTasks(P);
5  |  Constraints ← extractConstraints(S);
6  |  P ← update(P, ToDecompose, Constraints);
7  |  S ← solve(P, MaxTimeIter);
8  return S
```

An initial construction phase of the problem where all compound tasks are abstract is found at line 1. This problem together with a maximum computation time *MaxTimeIter* are given as inputs to the algorithm at line 2. Then, iterative decompositions are performed as long as there are abstract tasks left (lines 3 to 7), to finally return a solution plan S (line 8).

At each decomposition step, the algorithm selects a set of abstract tasks to decompose or refine (line 4), extracts constraints from the current solution S (line 5), computes an updated problem P (line 6) and solves this new problem with a maximum computation time (line 7). The solution found on the last problem is finally returned (line 8). This pseudo-code allocates the same computation time to each iteration of the refinement, but others distributions of the global computation time could be explored.

3.5 Experimental results

Preliminary results were obtained based on different types of problem instances, applying the iterative decomposition strategy presented above.

3.5.1 Problem instances

3.5.1.1 Multi-robot deployment instances

In order to test the different strategies, we generated benchmarks around the multi-robot deployment application previously described in Section 3.2.1 which is a specific variant of the general MRD problem introduced in Chapter 2. A problem instance generator was developed for this type of benchmark, which takes as parameters the following elements:

- the number of *observation areas*,
- the number of *observation activities* required to observe each (compound) area,
- the number of *available robots* to perform area observations,
- the number of *available frequencies* to transmit data during an observation,
- the total number of shared links between crossing points in the deployment area (the *size of the field*),
- the maximum number of transfer links for a single transfer (move) of a robot (the *size of each path* between observation areas).

It is also possible to set the maximum duration of the different atomic tasks of the problem. The results are presented for two representative multi-robot instances. The first one is denoted by 5Z. It is a small instance involving 5 zones to be observed in a randomly generated field, 2 robots, 3 observations required per zone, 3 crossing points at most per robot move, and 5 available frequencies. The second instance is named 50Z and corresponds to a larger instance involving 50 zones to be observed in a randomly generated field, 4 robots, 10 observations per zone, 3 crossing points at most per move and 3 available frequencies.

3.5.1.2 Job Shop and Open Shop Scheduling Problems

We also tested the methods on well-known benchmarks of the literature, namely the *Open Shop* Scheduling Problem (OSSP) and the *Job Shop* Scheduling Problem (JSSP) instances of E. Taillard [141].

For the *Open Shop* problems, we considered the 10×10 and the 20×20 instances. More precisely, we transformed these two instances into a two-level hierarchical problem involving respectively 10 and 20 compound tasks, 10 (respectively 20) subtasks per compound task, and 10 (respectively 20) disjunctive resources to perform them.

For the *Job Shop* problems, we considered the 20×20 and the 100×20 instances. Similarly, we transformed these instances into a two-level hierarchical problem involving 20 (respectively 100) compound tasks, 20 subtasks per compound task, and 20 disjunctive resources.

3.5.2 Experiments

3.5.2.1 Experimental results

Table 3.1 presents the results for both instances of the multi-robot deployment problem, 5Z and 50Z. For solving these problems, a maximum computation time *MaxTime* is set to 1 minute for the 5Z instance and 5 minutes for the 50Z instance. When solving with the abstraction mechanisms, the number of iterations *MaxTimeIter* described in Algorithm 3.1 corresponds to the *MaxTime* distributed equally among all iterations. In these experiments, the information transferred between iterations correspond to the *start-to-start* precedence constraints.

Tables 3.2 and 3.3 present respectively the results for the OSSP and the JSSP, with a maximum computation time of 10 seconds. Since several problem instances from the set of E. Taillard [141] have the same sizes, we present in these tables only the first instance of a given size since the results were comparable for the rest of them.

Instance	Abstraction	H (%)	Best Makespan	TF	TB	TT
5Z	No abstraction		**158***	0.30	0.30	17.4
	Abs^{C0}	H1 (5)	158	-	-	31.2
		H1 (20)	158	-	-	30.9
		H2 (5)	158	-	-	34.3
		H2 (20)	158	-	-	44.2
		H3 (5)	158	-	-	15.7
		H3 (20)	158	-	-	31.8
	Abs^{C1}	H1 (5)	158	0.59	17.0	29.7
		H1 (20)	158	0.87	12.4	37.6
		H2 (5)	158	0.97	24.9	46.2
		H2 (20)	158	0.67	31.9	43.2
		H3 (5)	158	0.64	31.0	43.3
		H3 (20)	158	0.77	24.3	45.3
50Z	No abstraction		**3785**	0.79	28	*MaxTime*
	Abs^{C0}	H1 (5)	3919	-	-	*MaxTime*
		H1 (20)	3910	-	-	*MaxTime*
		H2 (5)	3919	-	-	*MaxTime*
		H2 (20)	3910	-	-	*MaxTime*
		H3 (5)	3919	-	-	*MaxTime*
		H3 (20)	3910	-	-	*MaxTime*
	Abs^{C1}	H1 (5)	3919	2.56	297	*MaxTime*
		H1 (20)	3910	2.79	299	*MaxTime*
		H2 (5)	3919	2.11	300	*MaxTime*
		H2 (20)	3910	2.29	298	*MaxTime*
		H3 (5)	3919	2.79	298	*MaxTime*
		H3 (20)	3910	3.15	299	*MaxTime*

Table 3.1: Experimental results for the multi-robot exploration application.

Instance	Abstraction	H (%)	Best Makespan	TF	TB	TT
10×10	No abstraction		**637***	0.11	0.67	1.16
	Abs^{C0}	H1 (20)	637	-	-	1.88
		H2 (20)	637	-	-	1.89
		H3 (20)	637	-	-	1.86
	Abs^{C1}	H1 (20)	637	0.81	8.82	9.35
		H2 (20)	637	0.54	8.86	9.52
		H3 (20)	637	0.69	8.68	9.68
20×20	No abstraction		**1155***	0.23	0.66	1.57
	Abs^{C0}	H1 (20)	1155	-	-	1.9
		H2 (20)	1155	-	-	1.91
		H3 (20)	1155	-	-	1.82
	Abs^{C1}	H1 (20)	1155	1.27	9.30	9.75
		H2 (20)	1155	1.08	9.40	9.81
		H3 (20)	1155	1.16	9.35	9.79

Table 3.2: Results for the OSSP.

Instance	Abstraction	H (%)	Best Makespan	TF	TB	TT
20 × 20	No abstraction		**1217***	0.20	0.20	1.87
	Abs^{C0}	H1 (20)	1217	-	-	2.38
		H2 (20)	1217	-	-	2.48
		H3 (20)	1217	-	-	2.35
	Abs^{C1}	H1 (20)	1217	1.17	9.47	9.85
		H2 (20)	1217	1.56	9.54	9.91
		H3 (20)	1217	1.32	9.48	9.98
100 ×20	No abstraction		**5464***	0.99	0.99	3.09
	Abs^{C0}	H1 (20)	5464	-	-	6.73
		H2 (20)	5464	-	-	7.05
		H3 (20)	5464	-	-	6.74
	Abs^{C1}	H1 (20)	5464	2.21	9.56	10.0
		H2 (20)	5464	2.94	9.57	10.0
		H3 (20)	5464	2.78	9.47	10.0

Table 3.3: Results for the JSSP.

For each resolution of the abstract problem using the decomposition heuristics described above, one must choose the number K of abstract tasks to be simultaneously decomposed. This number is expressed here as a percentage p of the total number of compound tasks. Formally, $K = \lceil p \cdot \mathbf{card}(\mathcal{C}) \rceil$. Percentage p is indicated in parenthesis next to the heuristic rule which is used. For the displayed results, either 5% or 20% of the abstract tasks are refined at each iteration.

The elapsed time until the first solution is found is indicated in Column **TF** (Time To First) and is less than 1 second for nearly all the small problem instances, and less than 4 seconds over all tested instances. It is important to emphasize that for the tested instances that didn't turn out to be significantly large, the basic approaches without consideration of the proposed techniques (line No abstraction in the tables) easily manage to tackle the problem. Column **TB** (Time to Best) indicates the time in seconds to find the best solution and Column **TT** (Total Time) indicates the total elapsed time in seconds. In Algorithm 3.1, **TB** corresponds to the time at which the *makespan* of the final solution is found. In other words, the *makespan* of the solution is not improved between **TB** and **TT**. Since for the abstraction Abs^{C0}, the *makespan* obtained at each iteration cannot be considered as a true upper bound, the resulting **TB** is not shown in the tables.

The line **No abstraction** corresponds to the resolution of the translation of the problem into Constraint Programming by the CP Optimizer tool. When CP Optimizer manages to prove that the bound was the optimum, the symbol $*$ is added in the tables to the makespan value in **green**. Otherwise, the makespan appears in **red** without the symbol $*$.

3.5.2.2 Discussion on the experiments

The obtained results reveal that the optimization tool CP Optimizer manages to find good solutions quite quickly for all the problem instances and performs better than our iterative decomposition algorithm. Indeed, even for the biggest problem that was tested, a first solution is found in 28 seconds, and it is not improved during the resolution time left.

On small problems (5Z and for the small instances of E. Taillard), our method is able to find the optimal *makespan* found by CP Optimizer. On the other hand, on larger problems, by adding the *start-to-start* precedence constraints between successive resolutions, the search space is restricted and there is a risk to potentially remove the optimal solution from the search space. Note that the solutions are close (3.5%) to the **Best Makespan** found for the problem without abstraction (**No abstraction** line).

Although both abstractions have similar results in the tables presented here, it turns out that their behaviour in the search phase is very different. For example, using the H3(5%) decomposition heuristic for instance 50Z, the first bound obtained for the Abs^{C0} abstraction is 3713, and it is considerably higher, 8083, for the Abs^{C1} abstraction (additional results that are not shown in the tables).

The differences between the resolution of the problem instances without abstraction and using the decomposition and abstraction-based method certainly comes from the fact that the abstractions used are very *aggressive* in the way they define resource consumption.

3.6 Conclusion

In this chapter, a **first decision method** to deal with Hierarchical Scheduling Problems was introduced, along with a **novel framework** for modelling some classes of such problems. To carry out this first study, this chapter only deals with relatively simple classes of HSPs, but more general classes will be covered in the remaining chapters. The modelling framework was inspired by some notions coming from the concepts of WBS and HTN planning, described in Chapter 1. A straightforward Constraint Programming interpretation of this framework was also presented.

The introduced decision method is based on task abstraction and task decomposition mechanisms. The overall approach attempts to use the **task decomposition hierarchy to guide search**. The decomposition hierarchy is thus exploited at the *algorithmic level*, and *not only for modelling* purposes. Other methods that could be considered in further research may include for instance automatically grouping problem tasks that are strongly constrained in relation to each other, or performing task abstractions by categories of the consumed resources.

The introduced method explores the **use of simple abstractions** by defining a duration and a set of consumptions for each compound task. The latter forms could be greatly improved in further research by **considering finer abstraction strategies**. For instance, one approach would be to consider abstractions using cumulative resources (resources with a certain capacity), that could represent a percentage of consumption of a resource by a subtask. Finally, it should be noted that the abstractions are computed on the basis of a schedule that corresponds to an effective solution for each compound task, as opposed to an approach that would reason solely by constraint propagation to obtain lower bounds on the duration of these tasks. The **information transferred between iterations** could be also reassessed so that the search space doesn't turn out to be significantly reduced as is the case with the *start-to-start* precedence constraints.

The results obtained are **preliminary results** in the context of the research objectives of this thesis. In Chapter 5, more iterative decision strategies will be introduced. These novel strategies, instead of considering the iterative partial decomposition of task abstractions, use an *iterative surrogate based strategy* and *an iterative cut generation approach* allowing to consider a wider class of resources and execution alternatives (decomposition methods) for some tasks as in the HTN framework.

Chapter 4

Generic models for Hierarchical Scheduling Problems

Contents

4.1 Introduction

This chapter aims to provide a generic decomposition-based framework to model Hierarchical Scheduling Problems (HSPs). In the previous chapter we have considered partial decomposition of task abstractions. In this chapter, the proposed framework is still based on decomposition but allows to consider a wider class of resources, as well as execution alternatives (decomposition methods) for some tasks, as in the HTN framework.

The considered HSPs are those which can be modelled as problems involving resources that must perform complex setup operations between the main tasks they realize. For such problems, our goal is to compute optimal execution dates for the tasks, taking into account temporal constraints and constraints on the availability of the required resources.

Some of the contributions of this chapter were presented in 2019 at the International Joint Conferences on Artificial Intelligence (IJCAI) [107] and at the International Conference on Principles and Practice of Constraint Programming (CP) [108]. Short resumes of the main contributions were also presented in 2019 and 2020 at the French ROADEF annual conference [109, 110].

This chapter is organized as follows. Section 4.2 describes Hierarchical Scheduling Problems dealing with setup times and Section 4.3 introduces a first specific approach to the decomposition of scheduling problems illustrated on the MRD application. Section 4.4 and Section 4.5 introduce generic models for HSPs, including a global model (a single-layer framework) and a decomposition-based model (a multi-layer framework). Section 4.6 illustrates the generic decomposition framework in the MRD case study. These generic frameworks can be extended to several scheduling problems involving setup operations, and cover all the anti-collision approaches introduced in Chapter 2. Finally, Section 4.7 concludes about the main contributions presented in this chapter.

4.2 Hierarchical Scheduling Problems involving setup operations

The Hierarchical Scheduling Problems considered in this chapter involve **resources for which setup times** (or **transition times**) might be considered between the tasks they must realize.

As already mentioned in Chapter 1, a setup time between two tasks j and k represents the amount of time that must be elapsed between the end of task j and the start of task k if k immediately follows j on a given resource [124]. The setup times usually depend on the order of execution of pairs of tasks j, k on a resource ($setup(j, k)$ or $s_{j,k}$) and they are called **sequence dependent setup times** [104]. They may also depend on each given resource i ($setup(i, j, k)$ or $s_{i,j,k}$), and they can also be associated with groups (families) of tasks and considered zero when processing successive tasks in a same family.

Setup times can also be used to represent **abstractions of potentially complex setup operations**. For instance, in an MRD[1] application (see Chapter 2) where several robots must be deployed on a field to make observations of some areas (see Figure 4.1),

[1]Multi-Robot Deployment

a setup operation that requires a robot to go from a waypoint A to a waypoint B can be approximated by a constant setup time obtained by a simple shortest path computation. In practice however, these setup operations correspond to actual robot moves, and the fact that the network of links between waypoints is **a resource that is shared between the robots** must be taken into account to evaluate the actual efficiency of a schedule. In this case, at a detailed level, there can be several candidate navigation paths to move between A and B, and each path alternative corresponds to a set of moves on links of the way-point graph.

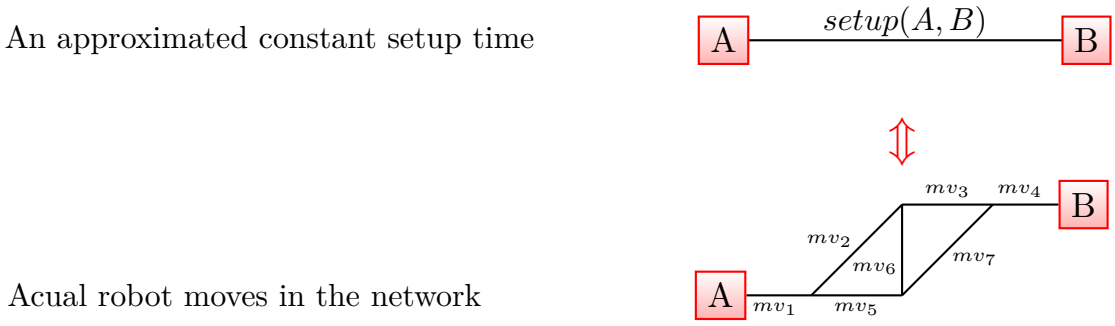

Figure 4.1: Example of setup times approximation in an MRD application.

Another application in which setup times might be considered is the placement of embedded functions over many-core processors [111]. In this application, embedded functions placed on distinct cores must potentially exchange data. In this case, the time required for each data exchange can be approximated by a constant setup time, but at a detailed level data transfers correspond to packet exchanges concurrently realized on a shared network. Also, in logistics, transferring an object from one location to another can be modelled as a simple setup time, but at a detailed level it might require using a shared fleet of vehicles whose activities must also be scheduled.

The goal pursued in this chapter is to propose a generic framework to deal with such Hierarchical Scheduling Problems involving complex setup operations.

4.3 Decomposition into a Two-Layer Scheduling Problem: an example

4.3.1 Multi-layer decomposition

To illustrate the decomposition-based approach proposed in this chapter, we refer back to the MRD application. To deal with a Hierarchical Scheduling Problem such as the MRD application, a global approach such as the one introduced in Chapter 2 must handle **a large (sometimes huge) number of tasks**. To decrease the computational complexity, a classic approach is to **explicitly break down the problem into**

several sub-problems. In our case, the MRD problem can be split into two parts: **(1)** one part which decides on the successive observations realized by each robot based on a coarse-grain model of navigation operations (so-called **layer L1**), and **(2)** one part responsible for detailing the navigations of robots within the network of shared waypoints and links (so-called **layer L2**).

These subproblems can be integrated for instance in a **two-layer decision strategy** that first synthesizes a high-level schedule based on a **coarse-grain model** of setup operations (decision layer L1), and then details this schedule based on a **fine-grain model** (decision layer L2).

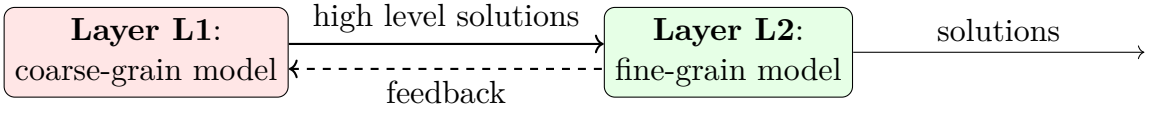

Figure 4.2: Model decomposition into two decision layers.

The advantage of this approach is that decision layer **L2 only needs to consider setup operations which are actually used in the coarse-grain solution** produced by decision layer L1.

Top-down approaches are commonly used in practice for hierarchical decision making, but as high-level decisions are computed from a coarse-grain model, they **can fail to reach the highest quality solutions**. This is why **hierarchical decision strategies that iteratively use the two scheduling layers** (with feedback from decision layer L2) to deal with complex setup operations can be used, as it is done with approaches presented in the next two chapters.

4.3.2 Coarse-grain scheduling model: Layer L1

In the high-level scheduling model of layer L1, the navigation between two given observations o and o' is abstracted in a very coarse way as a simple integer setup time required between the end of the realization of o and the start of the realization of o'.

In addition to some inputs already mentioned in Section 2.3 (the set of *frequencies*, the set of *robots*, the set of *observation areas*, the set of *observations* to be performed, and the *temporal horizon*), an additional input of the high-level MRD scheduling problem considers:

- a constant setup time $setup_{r,o,o'} \in \mathbb{N}$ for each robot $r \in \mathcal{R}ob$ and each pair of observations (o, o') that could be successively realized by r; this setup time represents the minimum duration required for r between the end of o and the start of o'; it is obtained from the length of the shortest path available to go from the location of o to the location of o'; this length is computed in polynomial time in

preprocessing, and it corresponds to an optimistic evaluation assuming that r is alone over the network during its navigation from o to o'.

We also define $setup_{r,\alpha_r,o}$ as the shortest duration required to move from the initial location of r to observation o, and $setup_{r,o,\beta_r}$ as the shortest duration required to move from observation o to the goal location of r. For each robot $r \in \mathcal{R}ob$, the associated function to obtain the setup time between each pair of observations, is denoted $setup_r$.

To define a CP model for layer L1, we again use several scheduling constructs available in the CP Optimizer tool. More precisely, a subset of the decision variables previously introduced in Section 2.3.2 is considered, including:

- for each observation $o \in \mathcal{O}bs$, one *interval variable* obs_o which must be placed during time frame $[0, H]$ and whose duration is $duObs_{ar_o}$, that is the observation duration of the area associated with o;
- for each observation $o \in \mathcal{O}bs$ and each robot $r \in \mathcal{R}ob$, one *optional* interval variable $obs_{o,r}$ used to represent the realization of observation o by robot r;
- for each robot $r \in \mathcal{R}ob$, two (non-optional) interval variables $obs_{\alpha_r,r}$ and $obs_{\beta_r,r}$ representing fictitious observations that r must realize at the beginning and end of its plan respectively; we recall that these fictitious observations allow us to model the initial and goal locations of the robots; interval $obs_{\alpha_r,r}$ has a null duration and must be placed at time 0, and interval $obs_{\beta_r,r}$ has a null duration and must be placed during time frame $[0, H]$;
- for each robot $r \in \mathcal{R}ob$, one *sequence variable* $\mathbf{seq}_r$ which represents an ordering over all present intervals associated with r, *i.e.* over all present intervals in set $\{obs_{o,r} \mid o \in \mathcal{O}bs \cup \{\alpha, \beta\}\}$.

The set of decision variables of the high-level model is therefore

$$V^1 = (\cup_{o \in \mathcal{O}bs}\{obs_o\}) \cup (\cup_{r \in \mathcal{R}ob,\ o \in \mathcal{O}bs \cup \{\alpha_r, \beta_r\}}\{obs_{o,r}\}) \cup (\cup_{r \in \mathcal{R}ob}\{\mathbf{seq}_r\}) \tag{4.1}$$

Constraints 4.2a to 4.2f are imposed over these variables. Constraint 4.2a imposes that the first and last observations in a robot sequence must correspond to the initial and final fictitious observations. Constraints 4.2b uses the *alternative* constraint of CP Optimizer and expresses that each observation is realized by a unique robot. Constraint 4.2c imposes that each robot can realize at most one observation for a given area (redundancy requirement). Constraint 4.2d expresses that observation tasks using the same frequency cannot overlap. Constraint 4.2e expresses that observations of a given area cannot overlap, taking into account the minimum delay d defined in the input

data. Constraint 4.2f enforces that observation tasks using the same robot must not overlap, taking into account the approximated setup durations required to move from one observation to the next for each robot. Symmetry breaking constraints could also be added.

$$\forall r \in \mathcal{R}ob,\ \mathit{first}(\mathbf{seq}_r, obs_{\alpha_r,r}) \wedge \mathit{last}(\mathbf{seq}_r, obs_{\beta_r,r}) \tag{4.2a}$$

$$\forall o \in \mathcal{O}bs,\ \mathit{alternative}(obs_o, \{obs_{o,r} \mid r \in \mathcal{R}ob\}) \tag{4.2b}$$

$$\forall a \in \mathcal{R}eq,\ \forall r \in \mathcal{R}ob,\ \textstyle\sum_{o \in \mathcal{O}bs \mid ar_o = a} \mathbf{pres}(obs_{o,r}) \leq 1 \tag{4.2c}$$

$$\forall f \in \mathcal{F},\ \mathit{noOverlap}(\{obs_{o,r} \mid o \in \mathcal{O}bs,\ r \in \mathcal{R}ob,\ \mathit{freq}_r = f\}) \tag{4.2d}$$

$$\forall a \in \mathcal{R}eq,\ \mathit{noOverlap}(\{obs_{o,r} \mid o \in \mathcal{O}bs,\ r \in \mathcal{R}ob,\ ar_o = a\}, d) \tag{4.2e}$$

$$\forall r \in \mathcal{R}ob,\ \mathit{noOverlap}(\{obs_{o,r} \mid o \in \mathcal{O}bs \cup \{\alpha_r, \beta_r\}\}, \mathit{setup}_r) \tag{4.2f}$$

The objective is to minimize the makespan, defined as the time at which each robot reaches its goal position:

$$\text{minimize} \max_{r \in \mathcal{R}ob} \mathbf{endOf}(obs_{\beta,r}) \tag{4.3}$$

Output. A high-level *solution schedule* σ^1 for layer L1 is an assignment of the values of all the decision variables in V^1 that satisfies all the problem constraints. A solution schedule σ^1 is said to be optimal if it minimizes the makespan.

4.3.3 Fine-grain scheduling model: Layer L2

The low-level scheduling model of layer L2 takes into consideration all the navigation paths available in the waypoint graph representing the observation field, to detail the routing of robots in the shared network and manage navigation conflicts. This decision layer only considers the navigation paths between the observation tasks present in the coarse-grain solution σ^1 produced by L1 (much less navigation options compared to the global CP model).

In addition to the inputs mentioned in Section 2.3 for the global MRD problem, the additional inputs of the low-level multi-robot scheduling problem are:

- the high-level solution schedule σ^1 produced by layer L1; in this solution, we keep for each robot r the value of sequence $\mathbf{seq}_r$, which defines the successive observations planned for r; to have flexibility in L2, we do not keep the exact dates found by layer L1 for present intervals in σ^1; in the following, to make some notations easier, we denote by Tr the set of all triples (r, o, o') such that in solution σ^1, observation o' is realized just after observation o for r;

- for each robot r and each pair of successive observations (o, o') realized by r (*i.e.* $(r, o, o') \in Tr$), a set of candidate paths $P_{r,o,o'}$ which can be used by r to go from o to o'; this set contains all paths whose length is not longer than the duration between the end of $obs^{L1}_{o,r}$ and the start of $obs^{L1}_{o',r}$ in the plan generated by layer L1, so that layer L2 considers not too pessimistic path scenarios to propose solutions. Each path p in $P_{r,o,o'}$ is specified by the sequence $[p_1, \ldots, p_Q]$ of successive network resources (waypoints and links) traversed by path ($p_q \in \mathcal{W} \cup \mathcal{L}$ for every $q \in [1..Q]$).

In the scheduling problem built for layer L2, the detailed routing between observation tasks must be defined. Considering the anti-collision mechanisms, only the model for the "Link and waypoint isolation and Minimum Handover" case (see Section 2.4.2) is given, since the ones for the simple "Link isolation" case (see Section 2.4.1) and for the "Path Isolation" case (see Section 2.4.3) are a bit simpler. For each robot $r \in \mathcal{R}ob$ and each observation o realized by r, the CP model contains one interval variable $obs_{o,r}$ as in L1. For each robot $r \in \mathcal{R}ob$ and each pair of observations (o, o') successively realized by r in the solution found by L1, we also consider the following variables, which together with the set of variables defined in Section 4.3.2 somehow cover the set of variables introduced in the global model of Section 2.3.2:

- one (mandatory) interval variable $mv_{r,o,o'}$ representing the global move of r from o to o';

- for each candidate path $p \in P_{r,o,o'}$, one optional interval variable $mv_{r,o,o',p}$ representing a move along path p;

- for each candidate path $p = [p_1, \ldots, p_Q] \in P_{r,o,o'}$ and each index $q \in [1..Q]$, one optional interval variable $mv_{r,o,o',p,q}$ representing the usage of the qth network resource of path p.

Together, these interval variables make up the set of decision variables V^2 of layer L2.

Fine-grain constraints associated with the "Minimum Handover" case are given

below.

$$\forall \gamma \in \mathcal{W} \cup \mathcal{L},\ noOverlap(\{mv_{r,o,o',p,q} \mid ((r,o,o') \in Tr) \wedge (p \in P_{r,o,o'}) \wedge (q \in [1..|p|]) \wedge (p_q = \gamma)\}) \quad (4.4a)$$

$$\forall (r,o,o') \in Tr,\ alternative(mv_{r,o,o'}, \{mv_{r,o,o',p} \mid p \in P_{r,o,o'}\}) \quad (4.4b)$$

$$\forall (r,o,o') \in Tr, \forall p \in P_{r,o,o'},\ span(mv_{r,o,o',p}, \{mv_{r,o,o',p,q} \mid q \in [1..|p|]\}) \quad (4.4c)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \forall q \in [1..|p|],\ \mathbf{pres}(mv_{r,o,o',p}) = \mathbf{pres}(mv_{r,o,o',p,q}) \quad (4.4d)$$

$$\forall (r,\alpha_r,o') \in Tr,\ endAtStart(obs_{\alpha_r,r}, mv_{r,\alpha_r,o'}) \quad (4.4e)$$

$$\forall (r,o,\beta_r) \in Tr,\ endAtStart(mv_{r,o,\beta_r}, obs_{\beta_r,r}) \quad (4.4f)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \forall q \in [2..|p|],\ \mathbf{pres}(mv_{r,o,o',p}) \rightarrow (\mathbf{startOf}(mv_{r,o,o',p,q}) = \mathbf{endOf}(mv_{r,o,o',p,q-1}) - 1) \quad (4.4g)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \mathbf{pres}(mv_{r,o,o',p}) \rightarrow (\mathbf{startOf}(mv_{r,o,o',p,1}) \geq \mathbf{endOf}(obs_{o,r})) \quad (4.4h)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \mathbf{pres}(mv_{r,o,o',p}) \rightarrow (\mathbf{endOf}(mv_{r,o,o',p,|p|}) \leq \mathbf{startOf}(obs_{o',r})) \quad (4.4i)$$

$$\forall (r,o,o') \in Tr,\ \forall p \in P_{r,o,o'},\ \forall q \in [1..|p|],\ \mathbf{pres}(mv_{r,o,o',p,q}) \rightarrow (\mathbf{endOf}(mv_{r,o,o',p,q}) - \mathbf{startOf}(mv_{r,o,o',p,q}) \geq duMv_{r,p_q} + 2) \quad (4.4j)$$

$$\forall f \in \mathcal{F},\ noOverlap(\{obs_{o,r} \mid o \in \mathcal{O}bs,\ r \in \mathcal{R}ob,\ freq_r = f\}) \quad (4.4k)$$

Constraint 4.4a forbids the temporal overlapping of tasks that use the same link or waypoint. Constraint 4.4b ensures that exactly one path is used between each pair of successive observations. Constraint 4.4c expresses that this path spans all its elementary moves. Constraint 4.4d states that the presences of elementary moves must be consistent with the presences of the selected paths. Constraints 4.4e and 4.4f define the start and end times of the moves from and to the first and last fictitious observations respectively. Constraints 4.4g enforce a handover period between the successive intervals involved in a chosen navigation path. Constraints 4.4h to 4.4i define the successive intervals involved in a transition between observations and chosen navigation paths. Constraint 4.4j defines the minimum duration of each elementary move interval, taking into account the handover period. We consider an inequality here since in the minimum handover configuration, a robot is allowed to wait on a link or a waypoint. For the anti-collision mechanism "link isolation" case (without minimum handover period), these two constraints would resemble to their counterparts, presented in Section 2.3.3. For the anti-collision mechanism "path isolation", *no overlap* constraints involving the network resources of each of the paths would be considered. Constraint 4.4k forbids the temporal overlapping of tasks that use the same frequency (the ordering of observations over frequency resources is not transferred from L1 to L2

to keep more flexibility in L2).

Finally, the goal is still to minimize the makespan (the same expression as in Equation 4.3).

Output. A low-level *solution schedule* σ^2 for layer L2, is an assignment of all variables in V^2 that satisfies all the problem constraints. It corresponds to a solution of the global MRD problem. A solution schedule σ^2 is said to be optimal if it minimizes the makespan (end time of the fictitious last observation tasks performed by the robots).

4.3.4 Towards a generic approach

The models given in Sections 4.3.2 and 4.3.3 correspond to a "manual" decomposition of the scheduling problem associated with the MRD application. In the two following sections, Section 4.4 and Section 4.5, more generic mechanisms are introduced. Two specific points are addressed, namely **(1)** how can we define a generic framework to have a compact modelling for each layer, and **(2)** how can we obtain a generic framework to handle the interaction between layers.

4.4 A generic single-layer framework to model Hierarchical Scheduling Problems

In this section a *generic* modelling framework to deal with HSPs with setup times is presented. This generic framework concerns the hierarchical model of a scheduling problem within a single layer. This model generalizes the framework previously presented in Chapter 3, first by considering disjunctive resources with setup times, and second by considering several possible decompositions for compound tasks.

4.4.1 A first global modelling framework to model HSPs with setup times

We consider a set of disjunctive resources $\mathcal{R}$, which cannot be used by several tasks simultaneously. This set is partitioned between the set $\mathcal{R}^d$ of simple disjunctive resources and the set $\mathcal{R}^s$ of disjunctive resources with setup times, for which a transition duration is required between successive tasks realized by the resource. Each resource $r \in \mathcal{R}^s$ has S possible running states and each task consuming r requires a particular resource state $s \in [1..S]$. For every pair of resource states $(s, s') \in [1..S]^2$, a minimum setup duration function $setup_r$ gives the duration $setup_r(s, s') \in \mathbb{N}$ required between the end of a task using r in state s and the start of a task using r in state s'.

Resources in $\mathcal{R}$ are used to execute *tasks*. A task t is defined by:

- a release date rd_t after which it can start and a due date dd_t before which it must end;
- the set of resources $R_t \subseteq \mathcal{R}$ it consumes all along its execution (from the start of the task to its end), with for every resource with setup $r \in R_t \cap \mathcal{R}^s$ the state $st_{t,r}$ required for r during the execution of t.

A task is either *primitive* or *compound*:

- a *primitive task* a, also called an *operation*, has a fixed duration du_a;
- a *compound task* c has a list of possible *decomposition methods* $M_c = [m_{c,1}, \ldots, m_{c,k}]$ usable for realizing c; each decomposition method $m \in M_c$ corresponds to a so-called *task network* (U_m, C_m).

Formally, a task network $w = (U, C)$ is composed of a set of tasks U and a set of temporal constraints C over these tasks.

In this section, we only consider a set of acyclic precedence constraints between pairs of tasks, but the approach can deal with any minimum and maximum distance constraint between the start and end time-points of pairs of tasks in U, which is required to deal with the "Minimum Handover" anti-collision mechanism;

The *generic* Hierarchical Scheduling Problem P is then defined as a tuple $(\mathcal{R}, w_0)$, where $\mathcal{R}$ is the set of disjunctive resources with setup times and w_0 corresponds to the *task network* (U_0, C_0), called the *root task network*, which represents the set of high-level tasks to be realized. In the following, we denote by $\mathcal{A}$, $\mathcal{C}$, and $\mathcal{M}$ respectively the set of all operations, compound tasks, and methods involved in the full hierarchical decomposition of the root task network. Moreover, for any resource $r \in \mathcal{R}$, U_r denotes the set of tasks t (primitive or compound) that consume resource r, *i.e.* such as $r \in R_t$.

Additionally, we assume that the addressed HSPs are *well formed*. More precisely, it is assumed that each task belongs to a unique task network and that the total number of tasks in an HSP is finite, which implies that the set of candidate decompositions of the root tasks is finite. It is also assumed that given a task t consuming a resource with setup times r, its release date rd_t is greater than or equal to the setup duration required to put resource r in state $st_{t,r}$ from the beginning of the schedule.

4.4.2 Constraint-based encoding

Given the previous formalization of an HSP, it is possible to generate a *flat* CP encoding for minimizing the makespan.

In this encoding, we introduce the following variables:

- for each task $t \in \mathcal{A} \cup \mathcal{C}$, an interval variable $itv_t \in [rd_t..dd_t]$;
- for each compound task $c \in \mathcal{C}$ and each decomposition method $m \in M_c$, an interval variable $itv_m \in [rd_c..dd_c]$.

In the flat CP encoding, we define Constraints 4.5a to 4.5i given below, which use several constraints available in the CP Optimizer tool. They express that all root tasks must be executed (4.5a) and that every compound task is realized using exactly one of its decomposition methods (4.5b). Also, each decomposition interval must cover all its subtasks (4.5c). The presence of subtasks of a method must also coincide with the presence of the method (4.5d). Constraints 4.5a to 4.5d actually correspond to already existing CP encodings of work breakdown structures.

The other constraints introduced are used to represent the scheduling and decomposition constraints of the problem. First, every operation has a fixed duration (4.5e). Constraints 4.5f to 4.5g enforce that tasks realized by disjunctive resources must not overlap, taking into account setup durations for resources with setup operations. For every task network (U, C) used in the model, all temporal constraints in C must be satisfied (4.5h). The latter use features such as the *endBeforeStart* constraint of CP Optimizer, which imposes that the end of a first interval must precede the start of a second one.

$$\forall t \in U_0,\ \mathbf{pres}(itv_t) = 1 \quad (4.5a)$$
$$\forall c \in \mathcal{C},\ alternative(itv_c, \{itv_m \mid m \in M_c\}) \quad (4.5b)$$
$$\forall m \in \mathcal{M},\ span(itv_m, \{itv_t \mid t \in U_m\}) \quad (4.5c)$$
$$\forall m \in \mathcal{M},\ \forall t \in U_m,\ \mathbf{pres}(itv_m) = \mathbf{pres}(itv_t) \quad (4.5d)$$
$$\forall o \in \mathcal{A},\ duration(itv_o, du_o) \quad (4.5e)$$
$$\forall r \in \mathcal{R}^d,\ noOverlap(\{itv_t \mid t \in U_r\}) \quad (4.5f)$$
$$\forall r \in \mathcal{R}^s,\ noOverlap(\{(itv_\tau, st_{t,r}) \mid t \in U_r\}, setup_r) \quad (4.5g)$$
$$\text{for every task network } (U, C),\ \forall \psi = (t, t') \in C_{temp},\ endBeforeStart(itv_t, itv_{t'}) \quad (4.5h)$$
$$\text{for every task network } (U, C), \text{ constraints in } C_{decomp} \quad (4.5i)$$

This encoding can be simplified in the following specific cases. If there is a unique decomposition method m for a compound task c, intervals itv_m and itv_c are equal. In this case, only one interval instance is defined and Constraint 4.5b is not generated. Similarly, if a decomposition method m has a unique subtask t (*i.e.* $U_m = \{t\}$), intervals itv_m and itv_t are equal. In this case, only one interval instance is defined and Constraints 4.5c to 4.5d are not generated.

When the goal is to minimize the makespan, the objective function generated consists in minimizing the maximum end time of a task in the root network, that is to minimize

$$C_{max} = \max_{t \in U_0} \mathbf{endOf}(itv_t). \tag{4.6}$$

Note that the model and the encoding could be generalized to cumulative resources having a finite capacity or to non renewable resources. This would allow to deal with Resource Constrained Project Scheduling Problems (RCPSPs) with modes, and more generally with kinds of hierarchical RCPSPs. In the end, the representation framework obtained is both usable by non-CP experts and quite expressive.

4.5 A generic multi-layer framework to model Hierarchical Scheduling Problems

As mentioned before, our goal is to consider two scheduling layers L1 and L2 that **represent setup operations at a different level of abstraction**. To model each of the decision layers individually, we consider two HSP models as described before.

The layers L1 and L2 consider respectively the HSPs $P_1 = (\mathcal{R}_1, w_1)$ and $P_2 = (\mathcal{R}_2, w_2)$, as defined in Section 4.4.1. For making these two layers interact, the **formal relationship between the scheduling problems they tackle** must be stated. To do this, a **generic scheme** is defined for automatically generating P_2 starting from a solution σ_1 found for P_1.

The main idea is to identify in set $\mathcal{R}_1$ (the resources of the coarse-grain model) a set of disjunctive resources with setup times $\mathcal{R}_1^{ref} \subseteq \mathcal{R}_1$ to be **refined by layer L2** (the fine-grain model). For each resource $r \in \mathcal{R}_1^{ref}$, we assume that there exists a problem-specific function $setupRefine_r$ which details the setup operations needed for r. More precisely, for each pair of tasks (t, t') successively realized by r in a solution σ_1 to P_1, a call to **function** $setupRefine_r(t, t')$ **returns a task network** w_r^{ref} (see Equation 4.7) defined by a set of tasks denoted by $U_r^{ref}(t, t')$ and a set of constraints over these tasks denoted by $C_r^{ref}(t, t')$.

$$w_r^{ref} = (U_r^{ref}(t, t'),\ C_r^{ref}(t, t')) \tag{4.7}$$

It is assumed that set $U_r^{ref}(t, t')$ contains one particular task, referred to as $setupOp_r(t, t')$, which spans all other tasks in $U_r^{ref}(t, t')$. For a feedback phase in the iterative deci-

sion strategy, once a solution σ_2 is found for the scheduling problem of layer L2, it is possible to **return to layer L1** quantities $\sigma_2(\mathbf{du}(setupOp_r(t,t')))$ and to use them **to update the abstract setup times** $setup_r(t,t')$ in L1 (see Section 5.2) or to **add new constraints** to layer L1 (see Section 5.3).

In layer L2, the **set of resources** $\mathcal{R}_2$ (see Equation 4.8a) contains $\mathcal{R}_1 \setminus \mathcal{R}_1^{ref}$ (the resources of layer L1 that do not need to be refined) plus all resources consumed by the tasks created by the setup refinement functions denoted by $\mathcal{R}_2^*$.

$$\mathcal{R}_2 = \mathcal{R}_1 \setminus \mathcal{R}_1^{ref} \cup \mathcal{R}_2^* \tag{4.8a}$$

$$U_2 = U_1 \cup \left(\bigcup_{r \in \mathcal{R}_1^{ref}, (t,t') \in \sigma_1(\mathbf{seq}_r)} U_r^{ref}(t,t') \right) \tag{4.8b}$$

$$\begin{aligned} C_2 = {} & C_1 \cup \left(\bigcup_{r \in \mathcal{R}_1^{ref}, (t,t') \in \sigma_1(\mathbf{seq}_r)} C_r^{ref}(t,t') \right) \\ & \cup \{\mathbf{pres}(t) = \sigma_1(\mathbf{pres}(t)) \mid t \in U_1\} \\ & \cup \{\mathbf{seq}_r = \sigma_1(\mathbf{seq}_r) \mid r \in \mathcal{R}_1^{ref}\} \\ & \cup \{\mathbf{endOf}(t) \leq \mathbf{startOf}(setupOp_r(t,t')) \mid r \in \mathcal{R}_1^{ref}, (t,t') \in \sigma_1(\mathbf{seq}_r)\} \\ & \cup \{\mathbf{endOf}(setupOp_r(t,t')) \leq \mathbf{startOf}(t') \mid r \in \mathcal{R}_1^{ref}, (t,t') \in \sigma_1(\mathbf{seq}_r)\} \end{aligned} \tag{4.8c}$$

Layer L2 takes as **constraints** the values of the presence variables and sequence variables obtained in solution σ_1. The precise dates found by layer L1 for present tasks are not transmitted to L2, to keep some temporal flexibility for layer L2. Theoretically speaking, the scheduling problem of layer L2 also **contains all tasks** in U_1 (see Equation 4.8b) and all the constraints in C_1 (see Equation 4.8c), even if in practice these sets of tasks and constraints can be pruned to keep only the specifications associated with present tasks. Last, L2 contains precedence constraints capturing the ordering of tasks and setup operations over resources in $\mathcal{R}_1^{ref}$. More formally, from the solution σ_1 found for layer L1, the scheduling problem for layer L2 is $P_2 = (\mathcal{R}_2, w_2)$ where $w_2 = (U_2, C_2)$.

In the above described scheme, the only elements which must be manually defined for L2 are the $setupRefine_r$ functions for resources $r \in \mathcal{R}_1^{ref}$.

4.6 Decomposition in a Multi-Robot Deployment case study

The proposed generic approach is illustrated below by considering an MRD application example as the one described in Chapter 2.

4.6.1 Coarse-grain scheduling model: Layer L1

The HSP built for layer L1 is illustrated in Figure 4.3. In the model, the root compound tasks are a set of *observation requests* $\mathcal{R}eq$, corresponding to areas of the field that must be observed. Each request $RQ_j \in \mathcal{R}eq$ is decomposed into several (primitive) *observation tasks* made by N_j distinct robots. Each request actually has several possible decompositions, corresponding to the candidate combinations of N_j distinct robots that can perform the corresponding observations. In the example of Figure 4.3, request RQ_1 can be realized through primitive observation tasks made by either $\{r_1, r_2\}$, or $\{r_1, r_3\}$, or $\{r_2, r_3\}$.

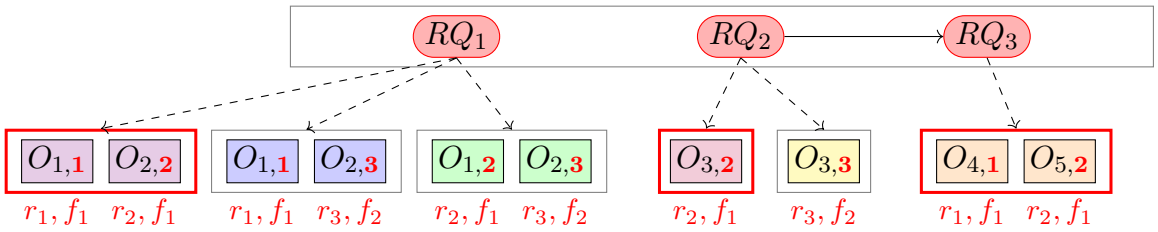

Figure 4.3: Hierarchical Scheduling Problem for layer L1.

Next, the model contains two kinds of resources, namely the set of *robot* resources $\mathcal{R}ob$ used to realize the observation tasks and the set of frequency resources $\mathcal{F}$ used to transfer observation data. Each primitive observation task simultaneously requires one robot resource and one frequency resource. All frequency resources are simple disjunctive resources (constant setup time equal to 0), while robots are disjunctive resources with setup times. For each robot $r \in \mathcal{R}ob$ and each pair of candidate observations (i, i'), $setup_r(i, i')$ returns the duration required by r to move from the location of observation i to the location of observation i' through the waypoint network (see Figure 4.4). We extend $setup_r$ so that $setup_r(0, i')$ gives the duration required to move from the initial location of r to observation i'. In the set of constraints of the model, we also consider a set of acyclic precedence constraints $\mathcal{P} \subseteq \mathcal{R}eq \times \mathcal{R}eq$ between requests. As explained previously, a CP model can be directly generated from the HSP illustrated in Figure 4.3. Note that the problem solved in layer L1 is a kind of Sequence Dependent Setup Time Job Shop Scheduling Problem (Section 1.1.2.2).

4.6.2 Fine-grain scheduling model: Layer L2

Layer L2 is responsible for detailing the routing of robots and managing routing conflicts on the navigation graph. For the "Link Isolation" case, the latter is defined by a set of *links* $\mathcal{L}$ available between adjacent waypoints, and two robots cannot simultaneously use the same link. For L1, the set of resources to refine is the set of robots ($\mathcal{R}_1^{ref} = \mathcal{R}ob$),

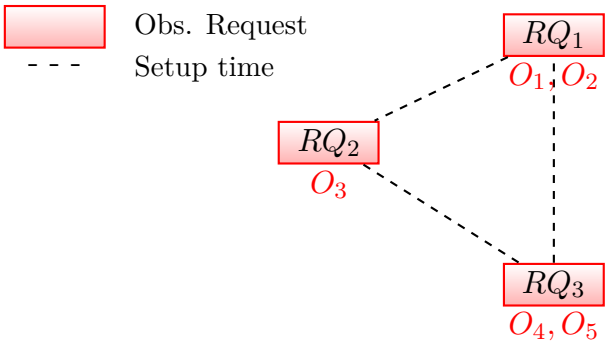

Figure 4.4: Setup distances through the waypoint network.

and for L2 the set of resources to consider is $\mathcal{R}_2 = \mathcal{L} \cup \mathcal{F}$ (the set of frequencies is kept in the set of resources since it is not refined by L2, *i.e.* $\mathcal{F} = \mathcal{R}_1 \setminus \mathcal{R}_1^{ref}$).

The HSP built for L2 is illustrated in Figure 4.5, for an example involving two robots and for the "Link Isolation" approach, the model can be easily adapted to deal with the other anti-collision mechanisms. For instance, in the "Path Isolation" approach, the network resources involved in a path "p", are reserved by all tasks associated with the traversal of p.

In the "Minimum Handover" approach, the handover periods must be considered. In each case, such considerations are expressed through the $setupRefine_r$ function associated with each resource r to be refined from L1 to L2.

Figure 4.5 shows the sequence of high-level moves and observations realized by each robot in the solution σ_1 produced by layer L1. The sequences of observations of robots are $obsSeq_1 = [1, 4]$ and $obsSeq_2 = [3, 5, 2]$. Robot 1 must therefore realize a compound move $MV_{1,0,1}$ from its initial location to observation number 1, then a primitive observation task O_1, then a move $MV_{1,1,4}$ from observation 1 to observation 4, and last a primitive observation task O_4. Precedence constraints are imposed to guarantee that each robot realizes a move activity between two successive observations. The problem also contains precedence constraints between some observations (coming from the precedences required between observation requests). Each compound move $MV_{r,i,i'}$ between two observations i, i' is the root of a task network. It has as many decomposition methods as the number of possible paths between the location of i and the location of i' (two possible decompositions in the case of compound move $MV_{2,3,5}$). A decomposition using the pth path points to a task network which specifies a sequence of atomic moves $mv_{r,i,i',p,k}$ required on links of the waypoint graph. Each atomic move consumes one link resource. For instance, the first path for $MV_{2,3,5}$ traverses the sequence of links $[l_6, l_7, l_8]$, and each subtask $mv_{2,3,5,1,k}$ consumes the kth link of the sequence. It is important to note that the traversals on the waypoints must also be considered in the anti-collision mechanism "Minimum Handover". Such a decomposition of moves between observations realized by each robot r is provided by the problem-specific func-

tion $setupRefine_r$ that allows to generate on-the-fly a model for layer L2 that contains only the global moves used in the last solution found by layer L1. The setup operation $setupOp_r(i, i')$ associated with each move from i to i' for robot r corresponds to $MV_{r,i,i'}$ in the example provided.

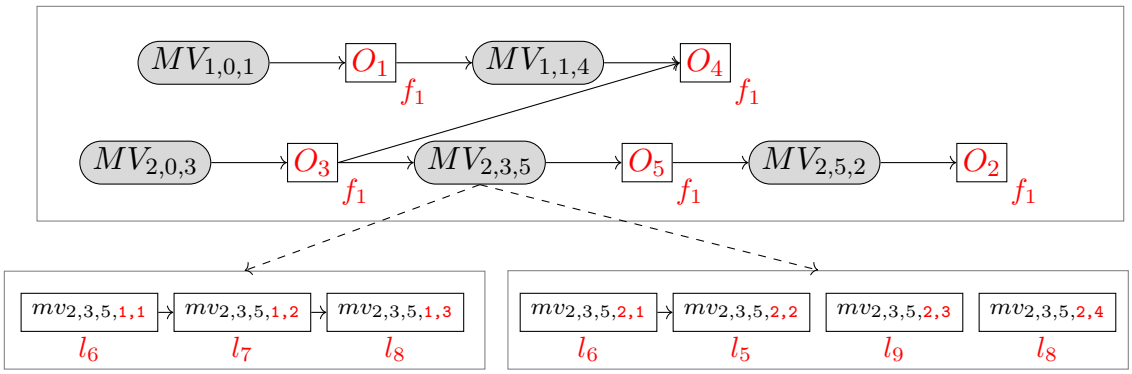

Figure 4.5: Hierarchical Scheduling Problem for layer L2.

4.7 Conclusions

In this chapter, we have introduced *generic decomposition-based mechanisms* to model scheduling problems involving resources for which there exists complex setup operations between the main tasks they must perform. This generic framework, instead of considering the iterative partial decomposition of task abstractions as in the previous chapter, introduces a problem decomposition allowing to consider a wider class of resources with setup times as well as execution alternatives (decomposition methods) for some tasks, as in HTN planning. This generic framework can be extended to several scheduling problems involving complex setup operations.

Chapter 5

Two-layer decision strategies for iterative hierarchical scheduling

Contents

5.1 Introduction

This chapter introduces several multi-layer iterative strategies to deal with the generic Hierarchical Scheduling Problems defined in the previous chapter, in particular those which can be modelled as problems involving resources that must perform complex setup operations between the main tasks they realize. For such problems our goal is to compute execution dates for the tasks, taking into account temporal constraints and constraints on the availability of the required resources.

The strategies introduced are built upon the two-layer models defined in the previous chapter, and combine the fast synthesis of high-level schedules (based on a coarse-grain model of setup operations) and the production of detailed schedules (based on a fine-grain model of setup operations).

As high-level decisions are computed from a coarse-grain model, **the highest quality solutions might be missed**. Henceforth, an iterative decision strategy, in which the solution found by the second layer is used to update the model of the first layer, is presented below.

The main idea of the **iterative process** between the **interactive layers** is shortly depicted in Figure 5.1.

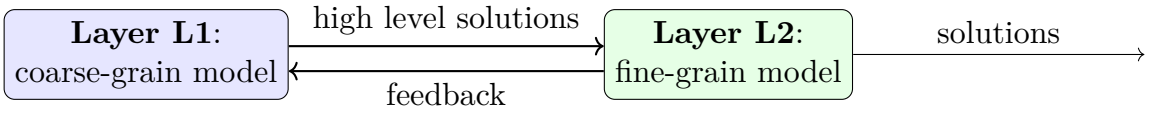

Figure 5.1: Interaction between two decision layers.

Two kinds of interactions between both layers are proposed thereafter.

The first one uses **a flexible surrogate-based iterative approach** in which the high level decision layer encapsulates a surrogate model of one or several downstream layers. During iterations, these finers layer might provide the surrogate model with new relevant information allowing to improve its quality.

This first contribution was presented in 2019 at the International Joint Conferences on Artificial Intelligence (IJCAI) [107], and a short resume was presented in the French ROADEF annual conference [109].

The second multi-layer approach uses **a cut generation strategy** in which a downstream layer contains an explanation module able to generate constraints holding on high-level decision variables. For the MRD problem, these constraints (or cuts) account for interferences found in the low-level solutions and which the high-level scheduler should consider to minimize the makespan. Four variants of the cut generation strategy were studied.

This second contribution was presented in 2019 at the International Conference on Principles and Practice of Constraint Programming (CP) [108], and a short resume was presented in 2020 at the French ROADEF annual conference [110].

To evaluate the approaches presented in this chapter, representative benchmarks of the MRD application will serve as a basis. In this cases, the two proposed approaches seek to decrease the computational complexity by considering the navigation interferences or conflicts only in detailed solution plans. In most of the existing references in the robotics field, these interferences are not taken into account at a planning stage, and the use of anti-collision mechanisms is supposed at execution time. Both **the surrogate-based approach** and **the cut generation approach** were tested over different types of problem instances and the efficiency of the decomposition approaches is compared to a global CP model. The aim is to find out to what extent default CP solvers can be exploited based on the proposed approaches and not to compare them with other resolution techniques such as MILP [79], PDDL [143], or Greedy-based algorithms.

This chapter is organized as follows.

Section 5.2 presents the surrogate-based iterative approach to make interact the scheduling layers of the decomposed modelling framework and discusses the relation of this approach with SBO[1]. Section 5.3 presents the different variants of the cut generation strategy. Both of the two previous sections also illustrate the proposed strategy on the MRD case study and present the experimental results based on this application.

Section 5.4 presents a comparison between the surrogate-based strategy and the different variants of the cut generation approach proposed. It presents the main outcomes of the experiments over different MRD problem instances for the three anti-collision strategies presented in Chapter 2. Finally, Section 5.5 provides conclusions on the contributions and gives some insights for further related research.

5.2 A surrogate-based decision strategy for iterative hierarchical scheduling

5.2.1 Iterative resolution approach

In this first multi-layer approach, a higher level stage might embody a surrogate model of one or several downstream stages. These finer stages (hierarchical levels) might provide the higher level stage with relevant information that allow to tune some parameters of the surrogate model. In our case, the setup times manipulated by the first layer of the problem are seen as a surrogate model of the setup operations modeled by

[1]Surrogate-Based Optimization

the second layer.

Then, each time **a new detailed schedule is produced** by layer L2, **input data of the imperfect coarse-grain model of layer L1 is updated** and **a new high-level solution is looked for**. Doing so, layer L1 **iteratively learns a better approximation of the content of layer L2**, the goal being **to converge very quickly towards better full solutions**.

It is important to note that in the two-layer mechanism introduced, **layer L1 learns an approximation which is not necessarily a lower bound on the fine-grain model of L2**, and the approach proposed can be extended even if layer L2 is a black-box simulator which does not provide critical path explanations on the schedules it produces.

More precisely, on one hand layer L1 transmits to layer L2 a set of present tasks and a sequence of tasks realized by each resource, and on the other hand layer L2 returns the current duration obtained for the detailed setup operations. This process is illustrated below in Figure 5.2. The iterations between the layers are performed until a given CPU time is reached. Note that the purpose of this process is not to obtain an optimal solution for the full problem but to get solutions of good quality within a short computation time, which is more crucial than finding optimality in many case studies.

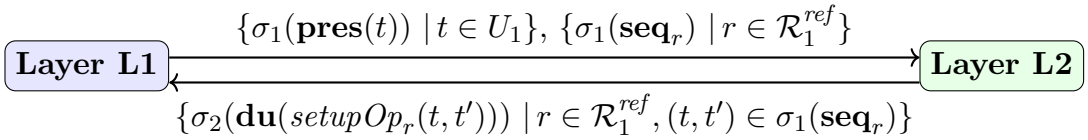

Figure 5.2: Iterative process using both layers.

It is worth noting that as the solutions produced by layer L2 are constrained by the solutions of layer L1 and as layer L1 works with an approximation of the global model, **the iterative process has no guarantee to find an optimal solution**. We can only say that if the initial setup times used by layer L1 are lower bounds of the real durations of setup operations, then **the solution produced by layer L1 at the first iteration gives a lower bound of the optimal makespan**. That lower bound can be used to evaluate the distance between the final solution of the iterative process and the optimal solution.

Algorithm 5.1 presents a generic pseudo-code of a process involving two layers that use each others solutions to minimize an objective function. We detail first the main

lines of this pseudo-code and then the implementation of each function.

Algorithm 5.1: Iterative hierarchical scheduling

Data: $cpuMax, nLoops$
Result: σ^* // *the best solution found*

```
1  σ1, σ2, σ* ← null, obj* ← +∞, it ← 0;
2  initL1();
3  while CpuTimeElapsed() < cpuMax do
4      if shouldRestart(σ1, σ2) then perturbL1() ;
5      else if σ2 ≠ null then updateL1(σ2) ;
6      (R1, w1) ← createPbL1();
7      σ1 ← solve((R1, w1), timeL1(it, nLoops));
8      if σ1 ≠ null then
9          (R2, w2) ← createPbL2(σ1);
10         σ2 ← solve((R2, w2), timeL2(it, nLoops));
11         if σ2 ≠ null ∧ objective(σ2) < obj* then
12             obj* ← objective(σ2), σ* ← σ2;
13     it ← it + 1;
14 return σ*;
```

The first step of the algorithm is to initialize several elements, including the solutions σ_1 and σ_2 respectively found by layers L1 and L2 at the last iteration, and the best solution σ^* found by layer L2 over all iterations. The best objective value found by layer L2 is denoted by obj^* (line 1). The input data for layer L1 is initialized through function **initL1** (line 2).

The process runs until a maximum CPU time is reached (line 3). To escape from local optima, we follow a restart strategy modelled by function **shouldRestart** that takes as parameters the last solutions found by layer L1 and layer L2. If a restart is required, function **perturbL1** reinitializes a given percentage of the input data of layer L1 (line 4). Otherwise, if layer L2 has found a solution at the previous iteration, the latter is used by function **updateL1** to update setup times for layer L1 (line 5). The problem to solve by layer L1 is created (line 6) and the associated solution is obtained through function **solve** to which a maximum CPU time is given. In our case, this CPU time is computed by a function **timeL1** that uses the number of iterations *nLoops* desired for the whole process and the number of loops performed so far (line 7). If a solution exists for layer L1, then it is used to create the problem $(\mathcal{R}_2, w_2)$ that is solved by layer L2 (lines 8-10). If this problem has a solution, we compare its objective value to the best one found so far and update the latter if needed (lines 11 - 12). The best solution found over all iterations is finally returned (line 14).

We detail below how the functions and parameters defined previously are instantiated:

- **createPbL1 and createPbL2.** Creates the scheduling problems as described previously in Section 4.5.
- **solve.** Solves a scheduling problem and returns the best solution found within the maximum CPU time allocated.
- **objective.** In our case, the objective function is the minimization of the makespan.
- **initL1.** The data required to initialize layer L1 corresponds to the approximate setup time matrices for the resources in L1. Depending on this initialization, the algorithm can behave differently. If the matrices are initialized with lower bounds of the real setup durations, then layer *L1 tends to provide optimistic solutions* to layer L2 and the makespan of solutions are generally greater for layer L2 than for layer L1. On the contrary, if the matrices are initialized with upper bounds of the real durations, then layer *L1 provides pessimistic solutions to layer L2.*
- **shouldRestart.** Restarts are performed if layer L2 has the same makespan during a given number of consecutive iterations. Restarts are also performed whenever layer L1 produces a solution whose makespan is greater than the best makespan found so far by layer L2. For layer L2, *we allow lower quality solutions* so that layer L1 can improve its approximation model.
- **perturbL1.** This function randomly reinitializes to their initial value a percentage *rateReinit* of the setup time matrix of each resource.
- **updateL1.** To update the abstract setup times of layer L1 based on the solution given by layer L2 at the previous iteration, we use a reinforcement learning rate $\alpha \in]0, 1]$ that represents the *influence of the actual setup times produced by layer L2* on the input values of layer L1. Formally, for each resource $r \in \mathcal{R}_1^{ref}$, if layer L1 sends to layer L2 a solution σ_1 where tasks t, t' are successively realized over r, then the solution σ_2 produced by layer L2 is used to update the value $setup_r(t, t')$ at the level of layer L1 by $(1 - \alpha) \cdot setup_r(t, t') + \alpha \cdot \sigma_2(\mathbf{du}(setupOp_r(t, t')))$. More *insightful* strategies are left for future research.

5.2.2 Related works

The two-layer scheduling approach proposed before can first be related to Logic-Based Benders Decomposition (LBBD, see Section 1.3.1.1), where the solutions produced by a first-stage problem are sent to a second-stage problem that might return new constraints (Benders cuts) which are violated by the current solution of the first-stage problem. In our case, layer L2 does not need to compute cuts. Instead, layer L2 just returns a fine-grain plan from which the parameters of the coarse-grain model of layer L1 are updated. This can be seen as a *light-weight interaction alternative* to LBBD where our goal is to update the "light" information contained in the setup time values (raw input data of layer L1), and not to generate constraints (cuts) holding on several decision

variables of layer L1. An approach based on the generation of cuts will be introduced in the next section.

We emphasize again that the purpose of our process is not to obtain an optimal solution but to get solutions of good quality within a short computation time. Also, in LBBD, the cuts returned are usually valid cuts, *i.e.* cuts that only prune suboptimal solutions. The approach we propose is less strict since at a given step, each setup time considered by L1 can be any approximation of the real setup times. We might have $setup_r(t, t') = 10$ at a given iteration, then $setup_r(t, t') = 12$ later in the process, and then $setup_r(t, t') = 8$, meaning that we update the model of L1 instead of accumulating a conjunction of cuts.

The interaction between layer L1 and layer L2 is actually **closer to works on approximation (or surrogate-based) models** to deal with computationally expensive large-scale problems [74, 76]. As stated previously in Section 1.3.4, these methods use a problem decomposition strategy in which a higher level stage (a master problem) might embody a surrogate model of one or several downstream stages (or sub-problems). The finer sub-problems might provide the master surrogate problem with relevant information which would allow tuning the approximation model after each precise objective function evaluation for a particular input.

In our case, for layer L1, the computation of a solution through the combinatorial model of layer L2 can be seen as the computation of a complex evaluation function. The latter is summarized in L1 by a matrix of minimum setup times between tasks. Each time a new detailed schedule is available, this surrogate model is updated based on the real setup times that take into account ***interferences*** (conflicts in the utilization of the network resources) between all detailed setup operations.

In surrogate models, a key point is the choice of the next parameters for which the complex evaluation function (layer L2) must be computed. In our case, **the more promising high-level schedules according to layer L1 are evaluated at each step**. This also differs from some iterative incomplete search techniques like Iterated Greedy Search and Large Neighborhood Search, where at each step a part of the current solution is modified.

Also, some works have already introduced two-stage decompositions involving CP models. In particular, Tran *et al.* [143] address a robot deployment application where the decomposition of a CP model also seeks to improve upon a Full-Model involving a number of tasks that potentially increases with the number of robots and locations. In a *master problem*, it simplifies the objective function (whereas our layer L1 simplifies the setup operations), and it does not exploit any feedback from the subproblem to converge towards better solutions (whereas our layer L2 sends feedback to layer L1). Note that the authors of this work state that it is not very clear whether the structure of their problem allows for a decomposition such as LBBD to be applied.

Last, several hierarchical planners have been developed in the planning community (see Section 1.2 for more details), such as CHIMP [138] and related Meta-CSP techniques [95, 96], HiPOP [13], FAPE [21, 40], ASPEN [32], EUROPA [9], or PLATINUm [144]. These planners use hybrid domain knowledge mixing symbolic, temporal and resource reasoning. All these ingredients are integrated in an iterative flaw resolution search technique that tries to repair at each step some flaws in the plan such as resource over-consumptions. The **iterations involved in our search scheme are not used to resolve flaws but to improve the quality of the coarse-grain model** used by the high-level decision layer. The recent planner GSCCB-SHOP2 [118] explicitly integrates task hierarchies, resources, and temporal constraints, but it is however not available for comparing results.

5.2.3 Experimental results

5.2.3.1 MRD problem instances

Experiments were performed over several multi-robot problem instances generated randomly, considering the anti-collision strategy named "Link Isolation". These instances contain from 1 to 15 *observation requests*, each request requiring observations from 1 to 3 robots. From 1 to 3 *frequencies* are available to transfer observation data, and 3 *robots* are available to carry out the observations. Each robot has a different speed, which determines the duration needed to traverse a link. The field structure contains $3 \times |\mathcal{R}eq|$ waypoints connected to their closest neighbors within a fixed range. Function **updateL1** is implemented with a learning rate α ranging from 0.2 to 1 and the reinitialization rate *rateReinit* for **perturbL1** is 0.2. The scheduling problems were solved using IBM ILOG CP Optimizer 12.5 on an Intel Xeon E5-1603, 2.80GHz 8GB RAM, setting $cpuMax = \{5, 30\}$ minutes and an adequate iterations number *nLoops*, depending on the number of observations (problem size) and on *cpuMax*.

5.2.3.2 Interactions between layers L1 and L2

To illustrate the interactions between the two layers, makespan values obtained during iterations of the proposed approach are presented in Figure 5.3 for a problem instance containing 5 *observation requests* (each one must be performed by 2 different robots). In order to accentuate the behavior (to obtain Figure 5.3), we have specifically implemented a very optimistic **initL1** function to initialize $setup_r$ values to zero, using a learning rate $\alpha = 1$.

The best makespans successively obtained by L2 during the iterations are marked with a filled dot. Figure 5.3 shows that makespans of both layers tend to converge quickly. When solutions of layer L2 cannot be improved, restarts are realized (vertical

lines in the figure). More precisely, the first two restarts occurred when L1 did not find a solution better than the best one found so far, and the third restart when the makespan of the solution in L2 remained the same during several iterations. Since **initL1** is very optimistic, the makespan of the solutions returned by L1 tends to increase, given that values of coarse-grain setup times are also increasing when receiving feedback from layer L2.

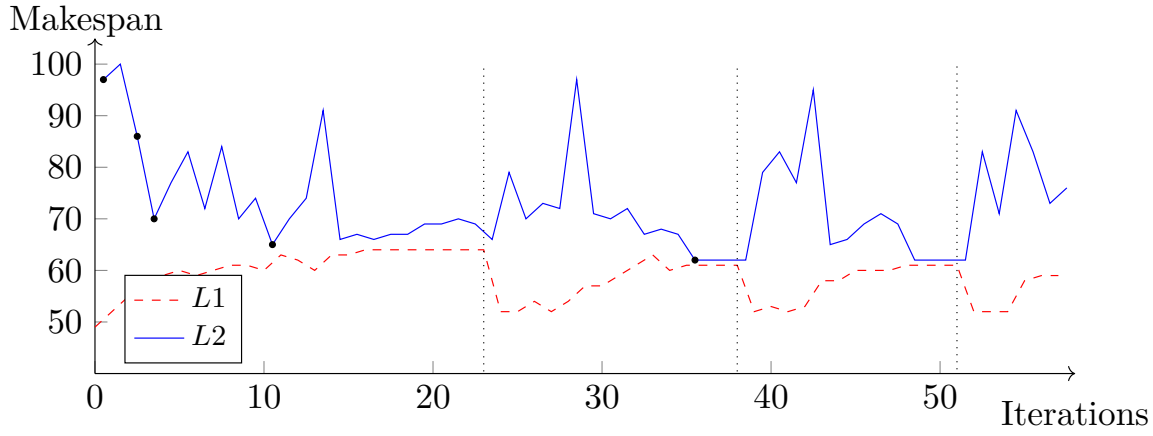

Figure 5.3: Evolution of makespan values during the interaction between both layers.

5.2.3.3 Comparison between the two-layer process and a global one-shot resolution

Full Model. To compare the two-layer decision process with a global one-shot resolution strategy, we implemented the global CP model described in Chapter 2. In theory, this model can be used to find an optimal solution. We recall that this global model contains (1) the model of layer L1 and (2) the model of layer L2 duplicated for each possible transition between observations. More precisely, it involves, for each robot r and each pair of distinct observations i, i', one optional task $MV_{r,i,i'}$ representing a global move of robot r from i to i', plus a huge number of optional intervals $mv_{r,i,i',p,k}$ modeling the move of r on the kth link of the pth path available to go from i to i'. The model contains a fine-grain no overlap constraint that takes into account all observations and all move intervals over links, but to boost constraint propagation we also add the coarse-grain no overlap constraint of layer L1 which takes into account all observation tasks and minimum setup times between them. Last, a constraint is added to ensure that the successive fine-grain activities realized by each robot have consistent types, *i.e.* that an observation interval associated with observation i is preceded by a move interval of the form $MV_{r,i',i}$.

Comparison with the decomposition approach. All generated instances have been solved using the two approaches. Representative results are given in Figures 5.4 and 5.5,

where the circle marks correspond to a makespan lower bound obtained from layer L1. For the instances of Figure 5.4, only one robot must observe each request, and there are $\lceil 0.2 \times |\mathcal{R}eq|^2 \rceil$ randomly generated precedences between requests. For Figure 5.5, two distinct robots must observe each request and there are no precedences. Since the same makespan is reached for most of the experiments with several α values, we only present results for $\alpha = 0.7$. Similarly, giving more CPU time to the two-layer process does not significantly improve the best solution, hence we only present the results for $cpuMax = 5$ minutes. For the Full Model, the generated instances contain from 14 tasks in the smallest instance to 139543 tasks in the largest one.

For the largest instances, the Full Model does not find any solution, even with a CPU time of 30 minutes. The two-layer approach achieves better makespan results in a significantly shorter time, and the makespan values obtained are very close to the makespan lower bounds. It provides first solutions of good quality in less than 3 seconds (not represented on the figures), even for the largest instances in which the complete solver is not able to reach any solution after several minutes. For the smallest instances, the two-layer approach manages to find the optimal solution (without proving its optimality). These results demonstrate that the proposed iterative approach is both simple and much more effective than the Full Model for solving large-size instances. It allows to quickly get good quality solutions no matter the problem size, and intuitively the iterative process used allows L1 to propose very quickly different promising solutions to L2, based on approximations which can freely manipulate both lower and upper bounds on setup times, differently from approaches which would only manipulate valid cuts.

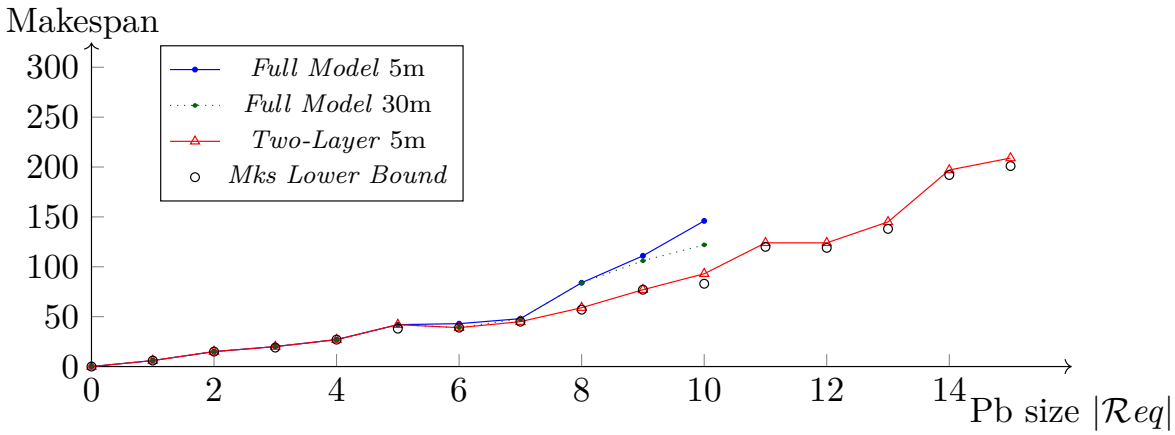

Figure 5.4: Makespan results with 1 robot per request and precedences.

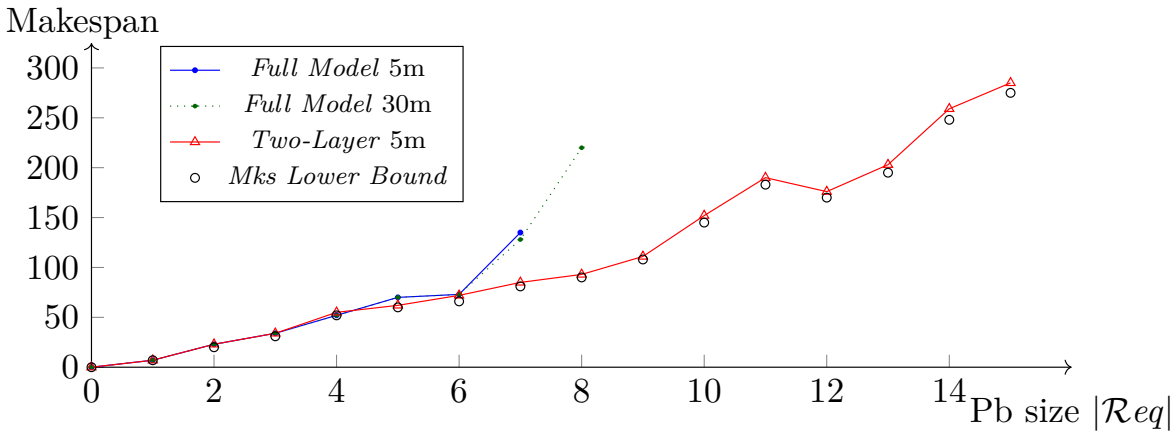

Figure 5.5: Makespan results with 2 robots per request and no precedences.

5.3 Cut generation strategies for iterative hierarchical scheduling

For the second strategy introduced, the MRD application will again serve as a basis to clearly define the approach. This strategy alternates between:

- quickly obtaining high-level schedules based on a coarse-grain CP model which approximates setup operations (navigation tasks in the MRD application) as setup times (setup times between observations in the MRD application),
- generating more accurate schedules based on a fine-grain CP model which takes into account all *resource usage conflicts* (interferences during traversals of the shared network in the MRD application).

This time, the low-level layer also contains *an explanation module able to generate constraints holding on high-level decision variables*. These constraints (or cuts) account for interferences found in the low-level solutions and which the high-level scheduler should consider to minimize the makespan. The proposed variants of the cut generation strategy differ in the way they allow to diversify search. Similarly to the surrogate-based strategy of the previous section, the aim is still to obtain good quality solutions within a short time, and not necessarily to get optimal solutions.

5.3.1 Iterative resolution approach

As mentioned before, when using only a top-down approach, the highest quality solutions may be missed since high-level decisions are computed from a coarse-grain model. We introduce here another iterative resolution strategy related to Logic-Based Benders

Decompositions (LBBDs), where a master solver iteratively proposes solutions to a slave solver which generates new constraints called *cuts*. Iterations between the master and the slave solvers are realized until convergence or until a maximum CPU time is reached. In our case, layer L1 first transfers to layer L2 the sequence of tasks realized by each robot. Then, layer L2 obtains a consistent solution schedule σ^2 for the low-level scheduling problem.

L2 also contains an explanation module which detects interferences between tasks consuming the shared network resources. As shown in Figure 5.6, this explanation module synthesizes cuts which are sent as a feedback to L1.

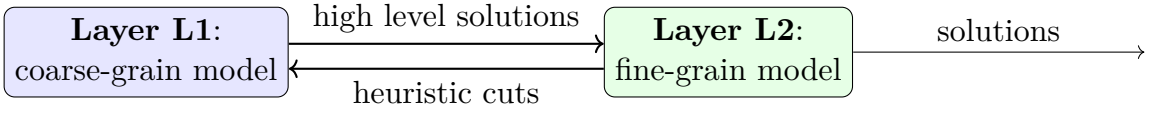

Figure 5.6: Interaction between two decision layers.

Compared to standard LBBD, one specificity of the technique proposed is that, as shown later, the explanation module generates cuts that are not necessarily valid in the sense that they might prune optimal solutions. The purpose of these cuts, which could be called *heuristic cuts*, is not to converge towards an optimal solution, but to speed the search for good solutions by forbidding in a coarse way some observation sequence patterns which might lead to interferences on detailed navigation activities. These patterns can be more or less precise and the generated cuts range from cuts usable to intensify search around the best known solution to cuts usable for exploring completely different regions of the search space.

Last, to perform several iterations between L1 and L2, we do not solve each problem in L1 or L2 to optimality. Instead, each run of L1 and L2 has a maximum allocated CPU time, which depends on the problem instance considered.

From a technical point of view, the surrogate-based approach also considers a two-layer interaction scheme but without any generation of cuts. Instead, the feedback from L2 to L1 corresponds to a simple update of the abstract setup durations of L1 by formula $setup_{r,o,o'} \leftarrow (1-\alpha) \cdot setup_{r,o,o'} + \alpha \cdot du$, where α corresponds to a learning rate and du corresponds to the duration of transition $o \rightarrow o'$ obtained for robot r in layer L2. On the opposite, the approach presented in this section exploits more detailed information and is closer to works on LBBD.

5.3.2 Cut generation in the explanation module of layer L2

The explanation module of L2 returns information about interferences found in the low-level solution and that have a negative impact on makespan minimization. These interferences are detected by examining conflicts related to the usage of network re-

sources during path traversals. More precisely, for a given robot r_1, if the duration required to traverse a path between two successive observations o_1, o_1' is strictly greater than the duration obtained in the solution of L1, this means that there is a resource precedence constraint which creates an interference at some point during the transition from o_1 to o_1'. The goal of the explanation module of L2 is to detect interferences related to network resource usages in the obtained sequence from σ^2. In the following, the approach is illustrated based on the "Link and waypoint isolation and Minimum Handover" strategy, where Figure 5.7 illustrates a scenario where two robots are in conflict for using waypoints wp_1 and wp_2, and link l_2 to traverse the paths needed to perform the sequences of observations shown in Figures 5.8 and 5.9 (handover duration not represented). In this case, the duration of the transition is longer for robot r_1 since it must wait for some network resources to be released by r_2. The explanations of these longer transition durations are depicted in red in Figures 5.8 and 5.9. In the general case, the explanation module of L2 detects through critical path analysis all triples (r_2, o_2, o_2') such that there is a transition from observation o_2 to observation o_2' for robot r_2 and such that at some point between o_1 and o_1', robot r_1 waits for a network resource to be released by r_2 during its transition from o_2 to o_2'. In terms of scheduling, we identify the critical resource precedence constraints associated with the network resources.

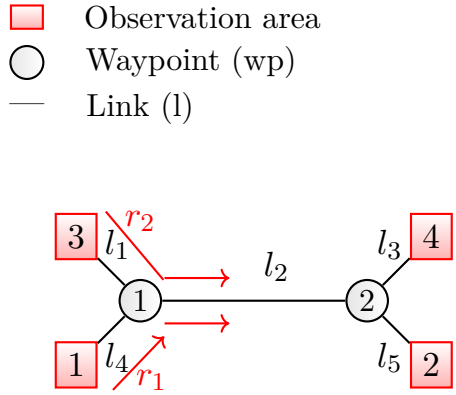

Figure 5.7: Interference between robots.

In the end, each interference produced by the explanation module is defined by a 6-tuple $(r_1, o_1, o_1', r_2, o_2, o_2')$. In the following, the set of all interferences synthesized from a low-level solution σ^2 is denoted by $\mathbf{Itf}(\sigma^2)$.

Four categories of cuts that can be generated through the explanation module are introduced below, by increasing order of refinement. In the following, we respectively denote by x^{L1} and x^{L2} the variables manipulated by layers L1 and L2. For instance, $obs_{o,r}^{L1}$ denotes the observation interval of o by robot r manipulated by L1, while $obs_{o,r}^{L2}$ denotes the observation interval $obs_{o,r}$ manipulated by L2 for representing the same task. Moreover, to get more concise expressions, we denote by $\mathbf{next}_{r,o,o'}^{L1} \in \{0,1\}$ the variable taking value 1 if interval $obs_{o,r}^{L1}$ is the predecessor of interval $obs_{o',r}^{L1}$ in the se-

quence of intervals $\mathbf{seq}_r^{L1}$ associated with robot r in layer L1. Also, $\sigma^2(\mathbf{startOf}(obs_{o,r}^{L2}))$ and $\sigma^2(\mathbf{endOf}(obs_{o,r}^{L2}))$, denote respectively the start and the end date of $obs_{o,r}^{L2}$ in the low-level solution σ^2 produced by L2.

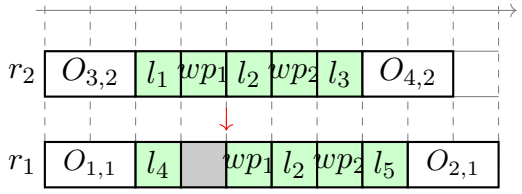

Figure 5.8: Minimum handover interferences.

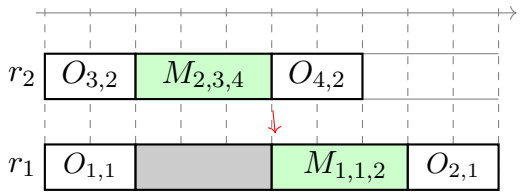

Figure 5.9: Path isolation interferences.

5.3.2.1 Broad Cuts: Setup Times (Cuts C1)

From the set of interferences $\mathbf{Itf}(\sigma^2)$, temporal constraints holding on high-level decision variables can be added to the scheduling problem of layer L1. A first possible approach is to return the following cuts:

$$\forall (r_1, o_1, o_1', r_2, o_2, o_2') \in \mathbf{Itf}(\sigma^2), \quad (5.1)$$
$$\mathbf{startOf}(obs_{o_1',r_1}^{L1}) - \mathbf{endOf}(obs_{o_1,r_1}^{L1}) \geq \sigma^2(\mathbf{startOf}(obs_{o_1',r_1}^{L2})) - \sigma^2(\mathbf{endOf}(obs_{o_1,r_1}^{L2}))$$

Such cuts are not valid since the initial abstract setup duration between o_1 and o_1' for r_1 (the setup duration considered by L1) could be met by updating the sequences of observations realized by other robots. However, these cuts can allow to quickly diversify search by penalizing, at the level of L1, a transition $o_1 \to o_1'$ for r_1 which *might* lead to an interference at the level of network resources.

Note that cuts C1 are equivalent to using the solution σ^2 from layer L2 to update the inputs of layer L1 (the coarse-grain duration of the setup operations between locations for each robot). Remember that $setup_{r,o,o'} \in \mathbb{N}$ corresponds to the high-level approximation of the duration required by r to move from the location of observation o to the location of observation o' over all possible paths of the waypoint network. The previous cuts amount to update $setup_{r,o,o'}$ by:

$$\forall (r_1, o_1, o_1', r_2, o_2, o_2') \in \mathbf{Itf}(\sigma^2), \quad (5.2)$$
$$setup_{r_1,o_1,o_1'} \leftarrow \mathbf{max}(setup_{r_1,o_1,o_1'},\ \sigma^2(\mathbf{startOf}(obs_{o_1',r_1}^{L2})) - \sigma^2(\mathbf{endOf}(obs_{o_1,r_1}^{L2})))$$

5.3.2.2 Moderate Cuts: Setup Times and Sequencing (Cuts C2)

In contrast to the previous cut generation strategy, we can consider another category of cuts which take into consideration the precise transitions creating the interference in σ^2. Such cuts are defined by:

$$\forall (r_1, o_1, o'_1, r_2, o_2, o'_2) \in \mathbf{Itf}(\sigma^2), \quad (5.3)$$
$$(\mathbf{next}^{L1}_{r_1,o_1,o'_1} \wedge \mathbf{next}^{L1}_{r_2,o_2,o'_2}) \rightarrow$$
$$(\mathbf{startOf}(obs^{L1}_{o'_1,r_1}) - \mathbf{endOf}(obs^{L1}_{o_1,r_1}) \geq \sigma^2(\mathbf{startOf}(obs^{L2}_{o'_1,r_1})) - \sigma^2(\mathbf{endOf}(obs^{L2}_{o_1,r_1})))$$

and can be added to the scheduling problem of layer L1. These cuts impose longer setup times in the high-level approximation whenever the successive observations involved in the interferences are successive again in a new sequence considered by L1. Cuts of type C2 are weaker than cuts of type C1, meaning that C2 prunes less solutions than C1.

5.3.2.3 Refined Cuts: Setup Times, Sequencing, and Temporal Positioning (Cuts C3)

More refined cuts coming from the solution analysis of layer L2 can be sent to layer L1. Unlike the previous cut generation strategies, these new cuts consider the time frame during which the setup tasks between observations are performed. Basically, they add high-level constraints which impose longer coarse-grain setup times only in case of temporal overlapping between the transitions involved in the interference. More precisely, let $overlap^{L1}(r_1, o_1, o'_1, r_2, o_2, o'_2)$ denote an expression taking value true when transitions $o_1 \rightarrow o'_1$ and $o_2 \rightarrow o'_2$ overlap in time, that is:

$$overlap^{L1}_{r_1,o_1,o'_1,r_2,o_2,o'_2} = \quad (5.4)$$
$$(\mathbf{endOf}(obs^{L1}_{o_1,r_1}) < \mathbf{startOf}(obs^{L1}_{o'_2,r_2})) \wedge (\mathbf{endOf}(obs^{L1}_{o_2,r_2}) < \mathbf{startOf}(obs^{L1}_{o'_1,r_1}))$$

The detailed cuts are then given by:

$$\forall (r_1, o_1, o'_1, r_2, o_2, o'_2) \in \mathbf{Itf}(\sigma^2), \quad (5.5)$$
$$(\mathbf{next}^{L1}_{r_1,o_1,o'_1} \wedge \mathbf{next}^{L1}_{r_2,o_2,o'_2} \wedge overlap^{L1}_{r_1,o_1,o'_1,r_2,o_2,o'_2}) \rightarrow$$
$$(\mathbf{startOf}(obs^{L1}_{o'_1,r_1}) - \mathbf{endOf}(obs^{L1}_{o_1,r_1}) \geq \sigma^2(\mathbf{startOf}(obs^{L2}_{o'_1,r_1})) - \sigma^2(\mathbf{endOf}(obs^{L2}_{o_1,r_1})))$$

which means that if transitions $o_1 \rightarrow o'_1$ and $o_2 \rightarrow o'_2$ appear again in a solution for L1 and if these transitions overlap in time, then a higher setup time must be used at the level of L1. Cuts of type C3 are weaker than cuts of type C2, meaning that C3 prunes less solutions than C2.

5.3.2.4 Valid Global Cut (Cut C4)

A quite simple valid cut consists in forbidding the entire sequence obtained for L1 at the previous step. This cut is defined by:

$$\neg \left[\bigwedge_{(r,o,o') \in Tr} \mathbf{next}^{L1}_{r,o,o'} \right] \tag{5.6}$$

It can be added to the scheduling problem of L1 as a global scheduling constraint. This cut will only force to seek for a different high-level solution, bypassing the synthesized information about the interferences found.

5.3.3 Experimental results

Benchmarks The two-layer approach and the four cut generation strategies proposed were evaluated over several MRD problem instances containing from 1 to 15 observation areas, connected through a network of shared links and waypoints. Several randomly generated observation scenarios were tested, considering from 1 to 3 frequencies available to transfer observation data, and from 2 or 3 homogeneous robots available to carry out the observations. The fields generated are regular grids of size $N \times M$ containing waypoints which are connected to their 4 adjacent neighbors. Random fields such as the one in Figure 2.1, and other grid configurations were also tested, leading to the same experimental conclusion. Observation areas are randomly positioned so as to be connected to one waypoint of the grid, and for most observation pairs o, o' there are several navigation paths of minimum length from o to o'. Each area requires observations from 1 to 3 robots (redundancy). The generated instances were all tested on IBM ILOG CP Optimizer 12.5 on an Intel Xeon CPU E5-1603, 2.80GHz 8GB RAM, setting an adequate number of iterations depending on the problem size and on cpuMax. Experiments were performed for both the minimum handover and path isolation configurations, to test the algorithms on instances which are more or less constrained in terms of usage of the shared network.

Results. Representative results of the tested configurations are given in Figure 5.10 and Figure 5.11, where two different robots must observe each observation area. For nearly all problem instances, the four proposed strategies for the two-layer approach achieve better makespan results than the global CP approach, in a significantly shorter computation time. Intuitively, the makespan would be greater for the “Path isolation” (see Figure 5.9) than for the ‘Minimum handover” approach (see Figure 5.8). The four proposed strategies provide good quality solutions in just a few seconds, even for the largest instances for which the global CP approach is not able to reach any solution with a CPU time of 30 minutes. For the smallest instances, most of the strategies of the two-layer approach manage to find the optimal solution, but without proving its

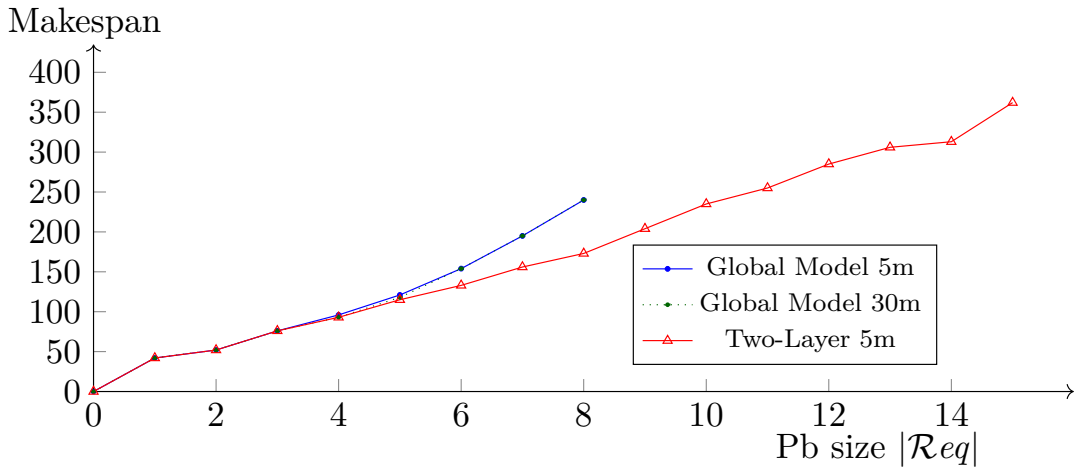

Figure 5.10: Makespan results for the Minimum Handover approach.

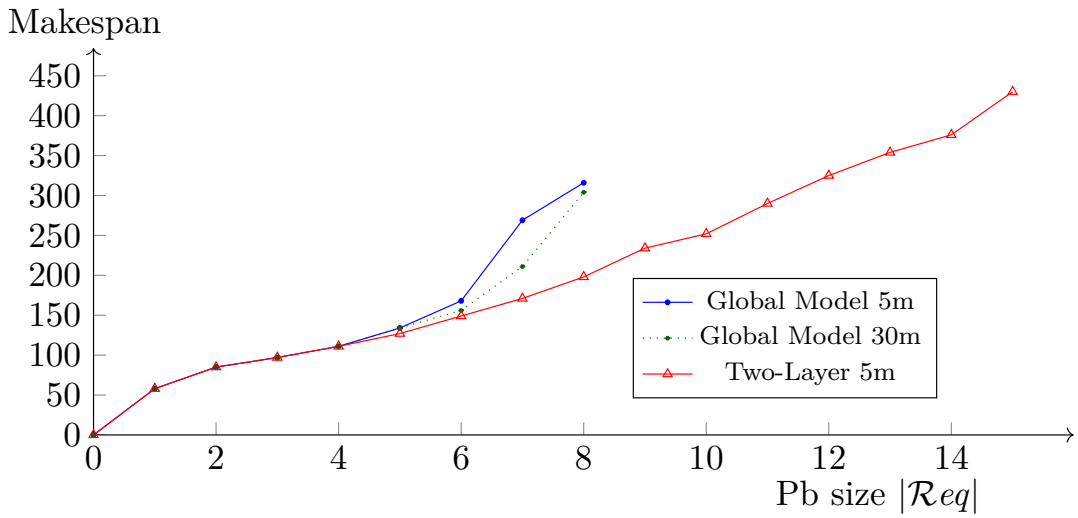

Figure 5.11: Makespan results for the Path Isolation approach.

optimality. As shown in Table 5.2, the results also demonstrate that over the set of benchmarks tested, there is not a single winner among the four cut generation strategies proposed. One explanation is that for some instances, it may be more advantageous to diversify the exploration of the search space by generating moderate cuts (strategy C2) or coarse-grain cuts updating the entire set of setup times (strategy C1), while for other instances it may be more convenient to explore a search space not so far from the current problem by generating fine-grain cuts (strategies C3 and C4). In other words, there is a kind of exploration/exploitation trade-off depending on the instance, leading to a disparity in the number of added cuts and in the elapsed time until the best solution is found, averaging between 1 and 2 minutes for the different strategies. To take advantage of all cuts, the next step would be to define a portfolio solver exploiting the different kinds of cuts, the goal being to outperform each individual cut generation

	Cut Strategies				Global	
$\|\mathcal{R}eq\|$	C1	C2	C3	C4	5m.	30m.
1	**42** [0]	**42** [0]	**42** [0]	**42** [0]	**42**	**42**
2	**52** [0]	**52** [0]	**52** [0]	**52** [0]	**52**	**52**
3	**76** [2]	77 [0]	**76** [2]	**76** [1]	**76**	**76**
4	**93** [0]	**93** [0]	**93** [0]	**93** [0]	96	**93**
5	**115** [3]	**115** [3]	**115** [3]	**115** [1]	121	118
6	**133**[13]	**133** [5]	**133** [8]	**133** [0]	154	154
7	**156**[37]	162 [8]	**156**[31]	159[11]	195	195
8	185[17]	**173**[42]	185[34]	182 [1]	240	240
9	205[38]	**204**[13]	211 [9]	205 [8]	-	-
10	**235**[41]	**235** [6]	**235** [9]	**235**[10]	-	-
11	256[40]	260[55]	**255**[25]	257 [7]	-	-
12	296 [0]	**286**[51]	296 [0]	291 [3]	-	-
13	312[33]	**306**[12]	320[13]	312 [7]	-	-
14	326 [5]	333 [3]	332 [3]	**313** [4]	-	-
15	373[11]	**362**[11]	**362** [7]	377 [4]	-	-

Table 5.1: Makespan found along with the number of cuts added until the best solution is found (in brackets) for different sizes of the set of observation areas $\mathcal{R}eq$ and for the Minimum Handover configuration; results are given for cut generation strategies C1 to C4, with a 5-minute time limit (cpuMax), and for the global CP model, with 5 and 30-minute time limits.

strategy. Portfolio approaches combine different solvers to get a globally better one, and their efficiency was already shown in the CP field [4, 5, 105].

5.4 Comparison: surrogate-based strategy vs. cut generation approach

5.4.1 Problem instances

Table 5.3 presents the indices of tables containing the experimental results discussed thereafter. Each line of the table refers to a group of problem instances containing instances of different sizes (number of observation areas) and based on different field configurations.

Column **ACM** refers to the Anti-Collision Mechanism adopted for each group of tests, as follows:

- **ACM1:** Link isolation,
- **ACM2:** Link and waypoint isolation and Minimum Handover,

	Cut Strategies				Global	
$\|\mathcal{R}eq\|$	C1	C2	C3	C4	5m.	30m.
1	**58** [0]	**58** [0]	**58** [0]	**58** [0]	**58**	**58**
2	96 [0]	**85** [2]	96 [0]	96 [0]	**85**	**85**
3	**97** [0]	**97** [0]	**97** [0]	**97** [0]	**97**	**97**
4	**111** [4]	**111** [8]	178 [0]	178 [0]	**111**	**111**
5	130 [4]	130[11]	**129** [6]	130 [7]	134	134
6	**150**[11]	153[10]	153[10]	**150**[13]	168	156
7	**174**[11]	**174** [4]	177[19]	187[17]	269	211
8	212 [6]	212[21]	**198**[18]	208[15]	316	304
9	240[16]	237[24]	**234**[10]	238[15]	-	-
10	288 [5]	270[13]	268[24]	**252** [9]	-	-
11	306[14]	**290**[20]	317[12]	309 [7]	-	-
12	326[13]	**325**[26]	335[10]	343 [5]	-	-
13	**354** [4]	374[16]	380 [7]	364 [5]	-	-
14	389 [0]	**376**[11]	389 [0]	389 [0]	-	-
15	**430** [4]	**430**[16]	432 [9]	**430** [4]	-	-

Table 5.2: Makespan found along with the number of cuts added until the best solution is found (in brackets) for different sizes of the set of observation areas $\mathcal{R}eq$ and for the Path Isolation configuration; results are given for cut generation strategies C1 to C4, with a 5-minute time limit (cpuMax), and for the global CP model, with 5 and 30-minute time limits.

- **ACM3:** Path Isolation.

Three different configurations for the observation fields were generated (column **Field**). These field configurations are not related to the number of observation areas but to the distance ratios between observation areas which are related to the length of the available paths or the number of connecting network resources between main waypoints. The three field configurations that were tested are denoted as **Small**, **Medium** and **Large**.

Column **RT** refers to three different Randomness Tests (1, 2 and 3) performed for each group of problem instances of different sizes. This randomness is derived from the fact that the setup time matrices for each robot are randomly reset in the surrogate-based approach or from the fact that some of the constraints that have been added to layer L1 are randomly removed in the cut generation approach. Column **QT** refers to the number of tables that will be shown in this chapter or in the annexes of this document (see Annexe A) for each row (group of problem instances) of this table.

ACM	Field	RT	QT	Description	Table names
ACM1	Small	1	3	Time / Mks / Dif [%]	5.4, 5.5, 5.6
		2	2	Mks / Dif [%]	A.1, A.2
		3	2	Mks / Dif [%]	A.3, A.4
	Medium	1	3	Time / Mks / Dif [%]	A.5, A.6, A.7
		2	2	Mks / Dif [%]	A.8, A.9
		3	2	Mks / Dif [%]	A.10, A.11
	Large	1	3	Time / Mks / Dif [%]	A.12, A.13, A.14
		2	2	Mks / Dif [%]	A.15, A.16
		3	2	Mks / Dif [%]	A.17, A.18
ACM2	Small	1	3	Time / Mks / Dif [%]	5.7, 5.8, 5.9
		2	2	Mks / Dif [%]	A.19, A.20
		3	2	Mks / Dif [%]	A.21, A.22
	Medium	1	3	Time / Mks / Dif [%]	A.23, A.24, A.25
		2	2	Mks / Dif [%]	A.26, A.27
		3	2	Mks / Dif [%]	A.28, A.29
	Large	1	3	Time / Mks / Dif [%]	A.30, A.31, A.32
		2	2	Mks / Dif [%]	A.33, A.34
		3	2	Mks / Dif [%]	A.35, A.36
ACM3	Small	1	3	Time / Mks / Dif [%]	5.10, 5.11, 5.12
		2	2	Mks / Dif [%]	A.37, A.38
		3	2	Mks / Dif [%]	A.39, A.40
	Medium	1	3	Time / Mks / Dif [%]	A.41, A.42, A.43
		2	2	Mks / Dif [%]	A.44, A.45
		3	2	Mks / Dif [%]	A.46, A.47
	Large	1	3	Time / Mks / Dif [%]	A.48, A.49, A.50
		2	2	Mks / Dif [%]	A.51, A.52
		3	2	Mks / Dif [%]	A.53, A.54

Table 5.3: Indices of tables containing experimental results to compare the surrogate-based approach and the cut generation approach.

The tables presented in this section concern **time** results (**Time** in the **Description** column), **makespan** results (**Mks** in the **Description** column) and the **gap to the best known solution** (**Dif** [%] in the **Description** column).

The tables concerning time results (in this section, Tables 5.4, 5.7, and 5.10) give the time in seconds until the first feasible solution (columns in **First Feasible Sol**) and the best solution (columns **Best Solution**) are reached.

The comparison of the results is made for all the presented approaches (using a time limit of 5 minutes), namely:

- Column **S**: Surrogate-based approach.

- Column **C1**: Cut generation approach adding broad cuts: setup times.
- Column **C2**: Cut generation approach adding moderate cuts: setup times and sequencing.
- Column **C3**: Cut generation approach adding refined cuts: setup times, sequencing, and temporal positioning.
- Column **C4**: Cut generation approach adding a valid global cut.
- Column **Full5m**: Full Model using a 5-minute time limit.
- Column **Full30m**: Full Model using a 30-minute time limit.

The column **L2** presents, according to the table, either the time or the makespan results of the *first feasible solution* for all the presented approaches. This value corresponds to the solution of the layer L2 on the *first iteration*, when no cuts and no inputs are updated. Since for each of the tested problem instances, the allowed time per iteration was the same, this value is the same for all of the proposed approaches.

The results are presented in a similar way for the tables concerning makespan results (in this section, Table 5.5, 5.8, and 5.11) and the gap to the best known solution (in this section, Table 5.6, 5.9, and 5.12). The best makespan for each problem instance is displayed in **bold blue** font.

5.4.2 Results

5.4.2.1 The simplest case: Link isolation

To compare the approaches presented in this chapter, Tables 5.4, 5.5, and 5.6 present respectively the **time** results, the **makespan** results and the **gap to the best known solution** for the *first* random test for the anti-collision mechanism *Link isolation*. These tables present the results for several *small* field problem instances involving from 1 to 15 observation requests.

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,15	0,19	0,19	0,15	0,15	0,30	0,24	0,20	0,19	0,19
2	0,25	0,34	0,34	0,25	0,25	0,35	0,37	0,27	0,71	0,71
3	15,14	1,76	1,76	3,15	15,14	15,39	15,07	15,93	4,00	4,00
4	15,22	10,75	10,75	15,25	15,22	15,25	15,38	15,25	27,67	425,77
5	10,25	47,91	47,91	10,29	10,25	10,22	10,66	10,87	275,13	242,83
6	10,19	30,07	30,07	10,18	10,19	10,27	10,89	10,27	73,18	73,18
7	10,21	59,70	59,70	10,22	10,21	42,49	64,97	83,98	59,70	59,70
8	12,23	176,08	176,08	252,91	252,72	265,67	51,78	51,67	176,08	176,08
9	15,30	229,65	229,65	15,27	15,30	45,34	15,26	45,56	229,65	313,00
10	21,08	-	-	63,45	21,08	278,74	278,30	131,02	-	-
11	25,72	-	-	275,73	25,72	225,42	75,60	75,47	-	-
12	30,61	-	-	151,09	151,06	271,31	210,66	150,82	-	-
13	39,00	-	-	187,64	187,60	263,46	263,04	112,89	-	-
14	50,10	-	-	50,51	50,10	51,02	50,11	50,58	-	-
15	75,84	-	-	226,59	226,19	225,32	226,24	75,13	-	-

Table 5.4: Time results for the *first* random test for anti-collision mechanism *Link isolation* on *small* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	40	40	40	**40**	**40**	**40**	**40**	**40**	**40**	**40**
2	51	62	62	**51**	**51**	**51**	**51**	**51**	**51**	**51**
3	74	80	80	**74**	**74**	**74**	**74**	**74**	**74**	**74**
4	95	101	101	95	95	95	95	95	99	**93**
5	114	133	133	**114**	**114**	**114**	**114**	**114**	116	116
6	131	166	166	**131**	**131**	**131**	**131**	**131**	154	154
7	158	186	186	158	158	**155**	156	**155**	186	186
8	182	230	230	176	176	**174**	178	177	230	230
9	200	274	274	200	200	198	200	**196**	274	273
10	235	-	-	225	235	220	**217**	225	-	-
11	251	-	-	**245**	251	247	247	247	-	-
12	277	-	-	**270**	**270**	283	276	276	-	-
13	313	-	-	300	**289**	293	301	305	-	-
14	317	-	-	**317**	**317**	323	**317**	**317**	-	-
15	356	-	-	**338**	343	341	350	356	-	-

Table 5.5: Makespan results for the *first* random test for anti-collision mechanism *Link isolation* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	40	0,00%	0,00%	0,00%	0,00%	0,00%
2	51	0,00%	0,00%	0,00%	0,00%	0,00%
3	74	0,00%	0,00%	0,00%	0,00%	0,00%
4	93	2,15%	2,15%	2,15%	2,15%	2,15%
5	114	0,00%	0,00%	0,00%	0,00%	0,00%
6	131	0,00%	0,00%	0,00%	0,00%	0,00%
7	155	1,94%	1,94%	0,00%	0,65%	0,00%
8	174	1,15%	1,15%	0,00%	2,30%	1,72%
9	196	2,04%	2,04%	1,02%	2,04%	0,00%
10	217	3,69%	8,29%	1,38%	0,00%	3,69%
11	245	0,00%	2,45%	0,82%	0,82%	0,82%
12	270	0,00%	0,00%	4,81%	2,22%	2,22%
13	289	3,81%	0,00%	1,38%	4,15%	5,54%
14	317	0,00%	0,00%	1,89%	0,00%	0,00%
15	338	0,00%	1,48%	0,89%	3,55%	5,33%
Average Gap		**0,98%**	**1,30%**	**0,96%**	**1,19%**	**1,43%**

Table 5.6: Gap to the best known solution for the *first* random test for anti-collision mechanism *Link isolation* on *small* field problem instances.

5.4.2.2 Link and waypoint isolation and Minimum handover

To compare the approaches presented in this chapter, Tables 5.7, 5.8, and 5.9 present respectively the **time** results, the **makespan** results and the **gap to the best known solution** for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover*. These tables present the results for several *small* field problem instances involving from 1 to 15 observation requests.

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,17	0,18	0,20	0,05	0,17	0,17	0,17	0,17	0,18	0,20
2	0,38	0,54	0,47	0,22	0,38	0,50	0,62	0,40	0,54	0,47
3	15,33	0,86	0,84	15,32	30,41	15,33	30,43	30,40	2,32	2,07
4	15,30	2,88	2,67	15,22	15,30	15,30	15,30	15,30	148,59	1236,99
5	10,23	5,37	4,92	10,16	30,42	30,40	30,41	40,85	252,64	1226,36
6	10,33	42,63	42,63	10,19	113,62	22,61	117,84	20,99	71,58	71,58
7	10,42	24,21	23,24	159,64	52,63	146,83	103,60	253,13	47,58	46,00
8	7,38	14,73	14,02	8,27	67,88	223,61	66,30	66,68	201,87	196,42
9	7,64	-	-	12,05	30,54	90,59	90,41	90,57	-	-
10	7,93	-	-	175,37	202,38	201,65	69,10	150,51	-	-
11	7,41	-	-	213,52	150,99	185,16	90,63	50,90	-	-
12	10,61	-	-	168,78	270,20	150,20	111,68	150,49	-	-
13	15,50	-	-	112,21	90,97	187,95	262,82	50,90	-	-
14	15,54	-	-	34,18	271,60	111,89	186,69	30,34	-	-
15	15,92	-	-	189,57	150,62	112,20	150,68	151,18	-	-

Table 5.7: Time results for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *small* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	42	42	42	**42**	**42**	**42**	**42**	**42**	**42**	**42**
2	52	52	52	**52**	**52**	**52**	**52**	**52**	**52***	**52***
3	77	112	112	**76**	**76**	77	**76**	**76**	**76**	**76**
4	93	102	102	96	**93**	**93**	**93**	**93**	96	**93**
5	148	129	129	**115**	**115**	**115**	**115**	**115**	121	118
6	139	166	166	**133**	**133**	136	**133**	**133**	154	154
7	162	202	202	**156**	**156**	**156**	**156**	**156**	195	195
8	195	246	246	182	**173**	179	176	176	240	240
9	213	-	-	201	201	199	198	**195**	-	-
10	277	-	-	224	229	225	**223**	226	-	-
11	292	-	-	255	249	252	**246**	250	-	-
12	296	-	-	280	**265**	276	279	**265**	-	-
13	336	-	-	291	300	299	**285**	297	-	-
14	333	-	-	328	322	316	320	**313**	-	-
15	381	-	-	357	**329**	347	**329**	**329**	-	-

Table 5.8: Makespan results for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	42	0,00%	0,00%	0,00%	0,00%	0,00%
2	52	0,00%	0,00%	0,00%	0,00%	0,00%
3	76	0,00%	0,00%	1,32%	0,00%	0,00%
4	93	3,23%	0,00%	0,00%	0,00%	0,00%
5	115	0,00%	0,00%	0,00%	0,00%	0,00%
6	133	0,00%	0,00%	2,26%	0,00%	0,00%
7	156	0,00%	0,00%	0,00%	0,00%	0,00%
8	173	5,20%	0,00%	3,47%	1,73%	1,73%
9	195	3,08%	3,08%	2,05%	1,54%	0,00%
10	223	0,45%	2,69%	0,90%	0,00%	1,35%
11	246	3,66%	1,22%	2,44%	0,00%	1,63%
12	265	5,66%	0,00%	4,15%	5,28%	0,00%
13	285	2,11%	5,26%	4,91%	0,00%	4,21%
14	313	4,79%	2,88%	0,96%	2,24%	0,00%
15	329	8,51%	0,00%	5,47%	0,00%	0,00%
Average Gap		**2,45%**	**1,01%**	**1,86%**	**0,72%**	**0,59%**

Table 5.9: Gap to the best known solution for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *small* field problem instances.

5.4.2.3 Path isolation

To compare the approaches presented in this chapter, Table 5.10, 5.11, and 5.12 present respectively the **time** results, the **makespan** results and the **gap to the best known solution** for the *first* random test for anti-collision mechanism *Path isolation*. These tables present the results for several *small* field problem instances involving from 1 to 15 observation requests.

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,26	0,40	0,28	0,32	0,06	0,06	0,06	0,06	0,40	0,28
2	0,18	1,67	1,41	0,41	0,43	0,41	0,44	113,71	1,67	1,41
3	15,36	2,07	2,01	15,19	15,36	15,36	15,36	15,36	2,07	2,01
4	15,55	4,61	4,97	30,22	187,10	15,05	50,22	15,55	10,80	11,29
5	10,31	6,42	6,22	90,28	60,71	99,83	90,30	150,27	128,08	117,68
6	10,89	12,79	11,00	169,16	143,23	118,88	10,89	218,69	239,97	457,78
7	10,81	23,85	23,54	260,27	116,02	104,11	30,22	230,31	159,69	1303,60
8	8,08	38,65	37,35	90,37	150,50	150,31	266,27	226,40	95,28	754,33
9	8,13	-	-	90,18	41,75	113,53	30,29	228,88	-	-
10	7,64	-	-	150,18	152,04	168,99	152,05	149,53	-	-
11	9,87	-	-	111,27	250,68	225,25	111,27	225,13	-	-
12	16,98	-	-	225,25	225,25	255,74	225,31	225,22	-	-
13	18,68	-	-	75,59	190,39	225,69	75,49	75,11	-	-
14	16,64	-	-	225,22	76,80	225,28	76,77	225,21	-	-
15	17,52	-	-	151,22	151,57	151,99	151,21	167,90	-	-

Table 5.10: Time results for the *first* random test for anti-collision mechanism *Path isolation* on *small* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	58	58	58	**58**	**58**	**58**	**58**	**58**	**58**	**58**
2	96	85	85	96	96	96	96	**85**	**85**	**85**
3	97	97	97	**97**	**97**	**97**	**97**	**97**	**97**	**97**
4	178	118	118	**111**	**111**	**111**	**111**	121	**111**	**111**
5	166	232	232	130	**127**	**127**	130	130	134	134
6	154	235	235	157	**149**	**149**	154	150	168	156
7	238	333	333	171	171	**168**	172	176	269	211
8	230	451	451	194	194	**190**	192	194	316	304
9	335	-	-	**222**	229	225	225	238	-	-
10	288	-	-	253	275	253	275	**252**	-	-
11	321	-	-	**267**	289	295	269	293	-	-
12	367	-	-	307	**305**	332	317	330	-	-
13	382	-	-	**304**	**304**	**304**	**304**	**304**	-	-
14	389	-	-	361	374	362	364	**359**	-	-
15	463	-	-	**393**	**393**	**393**	**393**	**393**	-	-

Table 5.11: Makespan results for the *first* random test for anti-collision mechanism *Path isolation* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	58	0,00%	0,00%	0,00%	0,00%	0,00%
2	85	12,94%	12,94%	12,94%	12,94%	0,00%
3	97	0,00%	0,00%	0,00%	0,00%	0,00%
4	111	0,00%	0,00%	0,00%	0,00%	9,01%
5	127	2,36%	0,00%	0,00%	2,36%	2,36%
6	149	5,37%	0,00%	0,00%	3,36%	0,67%
7	168	1,79%	1,79%	0,00%	2,38%	4,76%
8	190	2,11%	2,11%	0,00%	1,05%	2,11%
9	222	0,00%	3,15%	1,35%	1,35%	7,21%
10	252	0,40%	9,13%	0,40%	9,13%	0,00%
11	267	0,00%	8,24%	10,49%	0,75%	9,74%
12	305	0,66%	0,00%	8,85%	3,93%	8,20%
13	304	0,00%	0,00%	0,00%	0,00%	0,00%
14	359	0,56%	4,18%	0,84%	1,39%	0,00%
15	393	0,00%	0,00%	0,00%	0,00%	0,00%
Average Gap		**1,74%**	**2,77%**	**2,32%**	**2,58%**	**2,94%**

Table 5.12: Gap to the best known solution for the *first* random test for anti-collision mechanism *Path isolation* on *small* field problem instances.

5.4.2.4 Comparison

Again, as in Sections 5.2 and 5.3, for nearly all problem instances, the five proposed strategies for the two-layer approach achieve better makespan results than the global CP approach, in a significantly shorter computation time, and the makespan values obtained are very close to the makespan lower bounds. For the smallest instances, most of the strategies of the two-layer approach manage to find the optimal solution (when this value is known) but without proving its optimality. For the largest instances, the Full Model does not find any solution, even with a CPU time of 30 minutes.

These results confirm that the five iterative strategies that were presented are much more effective than the Full Model for solving large-size instances, allowing to quickly obtain good quality solutions no matter the problem size.

The experiments presented in this section also confirm the conclusion of the previous experimental results of Section 5.3 about the interest of a portfolio solver exploiting the different kinds of cuts, the goal being to outperform each of the five individual iterative strategies. Indeed, there is not a single winner among the five tested strategies for the two-layer approach over the set of benchmarks that were tested. This is, for some instances, it may be more advantageous to diversify the exploration of the search space by generating moderate cuts, coarse-grain cuts or updating the entire set of setup times,

while for other instances it may be more convenient to explore a search space not so far from the current problem by generating finer cuts.

5.5 Conclusions

In this chapter, *two novel flexible approaches* to deal with scheduling problems involving complex setup operations were presented. These two-layer processes are not used to obtain an optimal solution but to *get solutions of good quality within a short computation time*. The approaches *exploit the strengths of existing CP solvers* and gives *acceptable computation times*, even on problems for which the set of possible decompositions of setup operations is large.

The first surrogate-based approach corresponds to a flexible approach allowing at each given step to consider setup times in layer L1 which might correspond to lower or upper bounds on the real setup times. This fact allows to update the model of layer L1 instead of accumulating a conjunction of cuts as in LBBD-type approaches in which the cuts returned are usually valid cuts, i.e. cuts that only prune suboptimal solutions. Recent research has also come to approaches involving multiple surrogate models simultaneously [150] to deal with the lack of model adjustment. This track could be explored in future research.

Also, four different strategies to generate cuts in the two-layer approach for solving a type of HSP with complex setup operations were proposed. The generated cuts account for the interferences found in the low-level solutions, related to conflicts on resources of a shared network that have a negative impact on makespan minimization.

The results obtained demonstrate the efficiency and complementarity of these cuts. Even for large size problems, in which the global CP approach we developed has difficulties to produce a first solution, the cut generation strategies show a superior performance. The complementary of the cuts along with the surrogate-based strategy leads to the idea of merging them in a portfolio of cuts. This idea might be further refined in upcoming related research, for which restart strategies when solutions found by the two layers cannot be improved might also be considered.

Last, the proposed approaches can be extended to other scheduling problems involving complex setup operations between the main tasks. As already mentioned, an example of such problems is the placement of embedded functions on a many-core processor [112, 111], where the functions placed on the different cores interact through data exchanges over a shared network. Similarly, logistic in warehouses involves object transfers between locations and requires the utilization of shared resources whose activities must also be scheduled.

Chapter 6

Line balancing optimization use case

Contents

6.1 Introduction

This last chapter consists of a final exploratory part of this thesis and seeks to investigate other real-world applications which can be treated as HSPs. The experiments conducted in this chapter will serve as preliminary results for future related research.

More precisely, we consider in this chapter another case study regarding a real-world manufacturing application related to *aeronautical assembly lines*.

This chapter is organized as follows. Section 6.2 presents an overview of the assembly line balancing application including a formal description of the inputs of a general case study. Section 6.3 presents a first decomposition-based framework to deal with this new case study. Finally, Section 6.4 discusses the preliminary experiments and provides conclusions on the contributions of this last chapter. It also gives some insights for further research.

6.2 The line balancing application

As stated previously, the application of interest in this chapter is an aircraft assembly line application in which the production line is organized as a pulse line containing several work stations to process an aircraft section. The closeness of this kind of applications with the RCPSP involving time windows will be enlightened later in this chapter. Other case studies involving time windows and coming from different fields can be adapted to the approach presented below.

From a general point of view, the conceptual design of aircraft assembly lines is relevant both for industrial needs and for academic purposes [97]. In this context, one key activity is *line balancing*, which in aeronautics is used to designate a scheduling of the assembly tasks and is not related to the actual balancing of this scheduling [114].

Formally, the following inputs are considered:

- a pulse line containing a set $\mathcal{S}$ of successive **work stations** to process an aircraft section
  - $gap \in \mathbb{N}$ denotes the transfer duration of the section between stations;
  - $takt \in \mathbb{N}$ refers to the duration during which an aircraft section stays in each work station, including the transfer duration to the next station;
- a set $\mathcal{T}$ of tasks or **operations**, that can only be carried out when the aircraft remains at a station;
- a set $\mathcal{O}$ of **operators** to perform the tasks
  - $nMaxOperators \in \mathbb{N}$ denotes the maximum number of operators available. Each operator can be associated only to a single station;
- a set $\mathcal{A}$ of **manufacturing areas** or zones in the aircraft section
  - $\forall a \in \mathcal{A},\ cap_a \in \mathbb{N}$ denotes the maximum number of operators that can work simultaneously in area a;
- a set $\mathcal{P} \subseteq \mathcal{T} \times \mathcal{T}$ of **precedence constraints** between tasks, related to project precedences;
- for each task $t \in \mathcal{T}$, the following inputs are considered:
  - a set of neutralized areas $neutr_t \subseteq \mathcal{A}$ which can not be used by the operators during the execution of task t, for instance, for security reasons;
  - a task duration $dur_t \in [0, takt \cdot |\mathcal{S}|]$;

- the number of operators $nReqOperators_t \in [0, nMaxOperators]$ required to perform t;
- a set of aircraft areas consumed $areas_t \subseteq \mathcal{A}$, which represent physical sections of an aircraft that are occupied by operators performing t; for each area $a \in areas_t$, a consumption $cons_{t,a} \in [1..cap_a]$ for t is considered.

Here, the objective considered is to determine the minimum number of operators required in the pulse line. But one can also seek, for instance, to decrease the *takt time*, or to actually balance the task allocation to the available resources.

6.3 Decomposition-based model

In this section, a first decomposition-based framework for the assembly line application is proposed. As in Chapter 5, a two-layer approach is introduced, but here the high-level layer L1 is based upon a MILP model formulation, and layer L2 can either be based on a *detailed* CBS model, or use some inputs from layer L1 to quickly generate plans based on a simple scheduler.

Is important to note that this decomposition can be based on different formats of the line balancing problem. We consider here the following decomposition in layers L1 and L2. In layer L1, the idea is to approximate the occupation and neutralization of areas by tasks. To do so, we split stations into several time intervals and consider the overall occupation and neutralization of areas. In layer L2, we consider the real occupation and neutralization of areas. Moreover, we take as input from layer L1 the number of operators required for each station and the station in which each task is performed.

6.3.1 Layer L1: MILP based on partially occupied temporal intervals

This high-level layer L1 is based on a MILP model of the assembly line application. This model considers that the time horizon to perform all the aircraft assembly operations on each work station is split into several homogeneous temporal intervals. These intervals are *partially* occupied by the tasks to perform, whose duration must be below the size of these intervals. The approach is described more formally thereafter.

6.3.1.1 Additional inputs

In addition to the above presented inputs, the following ones are considered:

- a set $\mathcal{N}$ of **time intervals**, considered to perform all tasks in the line
  - $\forall n \in \mathcal{N}$, $dur_n \in \mathbb{N}$ denotes its duration;
  - $\forall n \in \mathcal{N}$, $station_n \in \mathcal{S}$ denotes its associated station;
  - $\forall t \in \mathcal{T}$, $\mathcal{N}_t \subseteq \mathcal{N}$ denotes the set of all possible intervals which task t can occupy by taking into account task precedences;
- $\forall t \in \mathcal{T}$, a set $\mathcal{M}_t$ of **execution modes**, related to the combinations of candidate temporal intervals to perform task t
  - $\forall t \in \mathcal{T}$, $\forall m \in \mathcal{M}_t$, $startItv_{t,m} \in \mathcal{N}$ denotes the start interval of task t in mode m;
  - $\forall t \in \mathcal{T}$, $\forall m \in \mathcal{M}_t$, $endItv_{t,m} \in \mathcal{N}$ denotes the end interval of task t in mode m;
  - $\forall t \in \mathcal{T}$, $\forall m \in \mathcal{M}_t$, $allItvs_{t,m} \subseteq \mathcal{N}$ denotes the set of intervals occupied by task t in mode m;
  - $\forall t \in \mathcal{T}$, $\forall m \in \mathcal{M}_t$, $midItvs_{t,m} \subset \mathcal{N}$ denotes the set of fully occupied intervals by task t in mode m.

  For instance, Figure 6.1 displays an example of two execution modes for task t of duration 4, in a station divided into 10 temporal intervals from n_0 to n_9, each of them of duration 1. For the execution mode m_1, the start interval $startItv_{t,m_1}$ corresponds to n_3, the end interval $endItv_{t,m_1}$ corresponds to n_6, and the set $allItvs_{t,m_1} = \{n_3, n_4, n_5, n_6\}$ corresponds exactly to the set $midItvs_{t,m_1}$ since in this mode all the temporal intervals are fully occupied by t. For the execution mode m_2, even if the start interval $startItv_{t,m_2}$ is also n_3, the end interval $endItv_{t,m_2}$ corresponds to n_7 and thus, the set $allItvs_{t,m_2} = \{n_3, n_4, n_5, n_6, n_7\}$ includes an additional temporal interval n_7. In this case, the set including the fully occupied temporal intervals by t corresponds to $midItvs_{t,m_2} = \{n_4, n_5, n_6\}$.

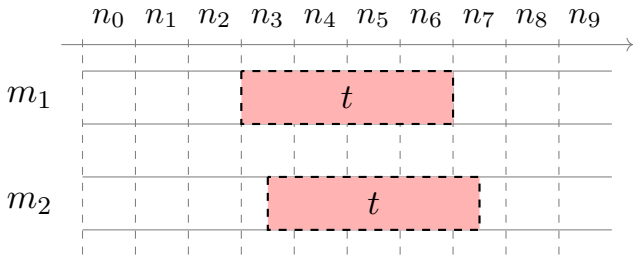

Figure 6.1: Example of two execution modes for a task t.

6.3.1.2 Decision variables

The following decision variables are considered:

- $\forall t \in \mathcal{T},\ \forall m \in \mathcal{M}_t,\ z_{t,m} = \begin{cases} 1, \text{ iff task } t \text{ uese execution mode } m \\ 0, \text{ otherwise} \end{cases}$

- $\forall t \in \mathcal{T},\ \forall n \in \mathcal{N}_t,\ d_{t,n} \in [0, dur_n]$, represents the duration of t on a temporal interval n;

- $\forall s \in \mathcal{S},\ nOperators_s$ represents the number of operators assigned to station s to execute tasks;

- $\forall a \in \mathcal{A},\ \forall n \in \mathcal{N},\ neutralized_{a,n} \in [0, dur_n]$ represents the time during which area a is neutralized over time interval n.

6.3.1.3 Constraints and objective function

The following constraints are considered. Constraint 6.1a states that each task t must implement one of its allowed filling modes. Constraint 6.1b limits the maximum number of operators in the line. The project precedences must be satisfied (Constraint 6.1c). Constraint 6.1d states the minimum number of operators required over each station based on the workload associated with each time interval of the station. Constraint 6.1e and Constraint 6.1f state the zone neutralization per area and per temporal interval whenever a *neutralizer* task t is being executed. More precisely, those constraints consider a neutralization duration on each area a and each time interval n, which corresponds at least to the duration of the longest task occupying time interval n on area a. Constraint 6.1g states the minimum duration of a task in each occupied temporal interval. Constraint 6.1h and Constraint 6.1i state that the sum of the durations on the occupied intervals must meet the task duration. Finally, Constraint 6.1j states that

the duration on each fully occupied interval must meet the interval duration.

$$\forall t \in \mathcal{T},\ \sum_{m \in \mathcal{M}_t} z_{t,m} = 1 \tag{6.1a}$$

$$\sum_{s \in \mathcal{S}} nOperators_s \leq nMaxOperators \tag{6.1b}$$

$$\forall (t, t') \in \mathcal{P},\ \sum_{m \in \mathcal{M}_t} z_{t,m} \cdot endItv_{t,m} \leq \sum_{m \in \mathcal{M}_{t'}} z_{t',m} \cdot startItv_{t',m} \tag{6.1c}$$

$$\forall n \in \mathcal{N},\ \sum_{t \in \mathcal{T}\ \mid\ n \in \mathcal{N}_t} d_{t,n} \cdot nReqOperators_t \leq nOperators_{station_n} \cdot dur_n \tag{6.1d}$$

$$\forall a \in \mathcal{A},\ \forall n \in \mathcal{N},\ \sum_{t \in \mathcal{T}\ \mid\ a \in areas_t,\ n \in \mathcal{N}_t} d_{t,n} \cdot cons_{t,a} \leq cap_a \cdot (dur_n - neutralized_{a,n}) \tag{6.1e}$$

$$\forall a \in \mathcal{A},\ \forall n \in \mathcal{N},\ \forall t \in \mathcal{T} \mid (a \in neutr_t,\ n \in \mathcal{N}_t),\ neutralized_{a,n} \geq d_{t,n} \tag{6.1f}$$

$$\forall t \in \mathcal{T} \mid dur_t \neq 0,\ \forall n \in \mathcal{N}_t,\ d_{t,n} \geq \sum_{m \in \mathcal{M}_t\ \mid\ n \in allItvs_{t,m}} z_{t,m} \tag{6.1g}$$

$$\forall t \in \mathcal{T},\ \sum_{n \in \mathcal{N}_t} d_{t,n} = dur_t \tag{6.1h}$$

$$\forall t \in \mathcal{T},\ \forall m \in \mathcal{M}_t,\ \sum_{n \in allItvs_{t,m}} d_{t,n} \geq z_{t,m} \cdot dur_t \tag{6.1i}$$

$$\forall t \in \mathcal{T},\ \forall n \in \mathcal{N}_t,\ d_{t,n} \geq \sum_{m \in \mathcal{M}_t\ \mid\ n \in midItvs_{t,m}} z_{t,m} \cdot\ dur_n \tag{6.1j}$$

The objective is to minimize the total number of operators in the line:

$$\text{minimize} \sum_{s \in \mathcal{S}} nOperators_s \tag{6.2}$$

6.3.2 Different alternatives for a Layer L2

As stated previously, layer L2 can simply use a basic scheduler to generate plans, using as an additional input the station s_o chosen for each task in Layer 1. This station is obtained from values $station_n \in \mathcal{S}$ associated with the temporal intervals n occupied by task t. This basic scheduler at layer L2, might also use a selection heuristic to choose the tasks to schedule at each time in the right station.

In another direction, L2 can also consider a *detailed* CBS model, as follows.

6.3.2.1 Decision variables

The following decision variables are considered:

- for each task $t \in \mathcal{T}$, one interval variable $task_t$ whose duration corresponds to dur_t; for each interval variable $task_t$ we consider
  - its associated time values $\mathbf{start}(task_t) \in [rd_t, dd_t]$ and $\mathbf{end}(task_t) \in [rd_t, dd_t]$;
- for each station $s \in \mathcal{S}$, $nOperators_s^{L2}$ represents the number of operators who will execute the tasks in s.

6.3.2.2 Constraints and objective function

The following constraints are considered.

Constraints 6.3a and 6.3b state that each task $o \in \mathcal{T}$ must be fully contained in its station s_o, fixed by Layer 1. The project precedences must be satisfied (Constraint 6.3c). Constraint 6.3d states the maximum capacity per zone using the *pulse* constraint which allows to manage cumulative resources. Constraint 6.3e limits the number of operators per station s to perform station tasks, according to the information obtained from layer L1. Constraint 6.3f states the maximum number of operators in the line. Finally, Constraint 6.3g states the neutralization constraints between tasks related to defined fixed areas.

$$\forall o \in T,\ \mathbf{start}(task_o) \geq (s_o - 1) \cdot takt \tag{6.3a}$$

$$\forall o \in T,\ \mathbf{start}(task_o) + dur_o \leq s_o \cdot takt - gap \tag{6.3b}$$

$$\forall (t, t') \in \mathcal{P},\ endBeforeStart(task_t, task_{t'}) \tag{6.3c}$$

$$\forall s \in \mathcal{S},\ \forall a \in \mathcal{A},\ \sum_{t \in \mathcal{T}, a \in areas_t, s_t = s} pulse(task_t, cons_{t,a}) \leq cap_a \tag{6.3d}$$

$$\forall s \in \mathcal{S},\ \sum_{t \in \mathcal{T}, s_t = s} pulse(task_t, nReqOperators_t) \leq nOperators_s^{L1} \tag{6.3e}$$

$$\sum_{s \in \mathcal{S}} nOperators_s \leq nMaxOperators \tag{6.3f}$$

$$\forall s \in \mathcal{S},\ \forall t \in \mathcal{T} | s_t = s,\ \forall a \in neutr_t,\ \forall t' \in \mathcal{T} | a \in areas_{t'},\ noOverlap(\{task_t\} \cup \{task_{t'}\}) \tag{6.3g}$$

The objective remains the minimization of the number of operators $nOperators$ in the pulse line

$$\text{minimize} \sum_{s \in \mathcal{S}} nOperators_s^{L2} \tag{6.4}$$

6.4 Conclusions

In this chapter, another case study regarding a real-world *line balancing* application was introduced. This application can be related to the RCPSP involving time windows (RCPSPTW), and similar case studies involving time windows can be adapted to the introduced representation.

The interaction between both layers and the information transferred in an eventual iterative process should be more extensively studied. With the considered real-world large scale instances with more than 600 tasks, 4 stations, and about 50 areas, for the experiments that were carried out it was not possible to generate good quality high-level solutions with the MILP model considered in layer L1.

In order for the size of the intervals to be not to long compared to the size of most tasks and therefore generate finer plans, the number of intervals had to be increased considerably, thus also increasing the decision variables considered by the model of layer L1 and preventing it from being resolved in reasonable computation times, as from 5 temporal intervals per station. This has to be done to efficiently consider (on finer temporal intervals) the neutralization constraints stated previously in Section 6.3.1.3.

The HSPs introduced in the previous chapters of this thesis, do not allow to capture some more complex features of the kind of problem described in this chapter, for instance, the **cumulative resources** and the **neutralization constraints** considered for the aircraft areas. Thus, more generic formalization of HSPs would be required to efficiently deal with problems such as the one considered in this chapter and cover more types of resources and constraints.

The results obtained are **preliminary results** in the context of the exploration of new decomposition strategies for different types of applications, and will serve as a basis for future related research.

Conclusion

Contributions

In this thesis, we introduce several modelling frameworks and approaches to tackle *hierarchical* decision making problems, formalized here as Hierarchical Scheduling Problems (HSPs). We manipulate hierarchical representations adapted to CBS problems based on "tasks-resources" models and not on "states-actions" models as in Hierarchical Task Network (HTN) planning.

An HSP based on several alternative representations of a Multi-Robot Deployment (MRD) case study is formalized, along with a first encoding in Constraint Programming. These alternative representations of the MRD problem involve three different anti-collision mechanisms in order to avoid collisions during traversals of a shared network, or at least to reduce the need to deal with collision situations online. The first anti-collision mechanism, named "Link isolation" corresponds to a simple approach which considers the observation areas and the waypoints as shareable resources in the network. In the second approach, named "Link and waypoint isolation and minimum handover", the only locations that are considered as shareable are those associated with observation areas, and there is a handover period to forbid inconsistent solutions where two robots would instantaneously cross each other over the network. The last anti-collision approach, named "Path isolation", and inspired by works dealing with inter-core interferences in multi-core processors, considers that all path resources are reserved during the whole path transition.

A first decision method to deal with HSPs based on task abstractions and iterative task decomposition is introduced. This approach attempts to use the task decomposition to guide the search, exploiting it at the algorithmic level and not only for modelling purposes.

More generic frameworks to model HSPs allowing to consider disjunctive resources with setup times and execution alternatives are also presented. In particular, a generic single-layer framework and a generic multi-layer framework are introduced to model HSPs. These frameworks are illustrated in a MRD case study.

Several efficient multi-layer strategies to deal with HSPs involving complex setup operations are proposed. The first one uses a flexible surrogate-based iterative approach in which the high-level decision layer encapsulates a surrogate model of one or several downstream layers. The second multi-layer approach uses a cut generation strategy in which a downstream layer contains an explanation module able to generate constraints holding on high-level decision variables. The generated cuts in this approach account for the interferences found in the low-level solutions, related to conflicts on resources

of a shared network. The results obtained demonstrate the efficiency and complementarity of the surrogate-based iterative approach and the four cut-generation strategies proposed.

The approaches presented exploit the strengths of existing CP solvers and give acceptable computation times, even on problems for which the set of possible decompositions of setup operations is large. These works show that the multi-layer decomposition framework allows to increase the efficiency of CBS solvers while getting high quality schedules. Last, they can be extended to other scheduling problems involving complex setup operations between the main tasks (the placement of embedded functions on a many-core processor or applications in logistics).

Finally, another case study regarding a real-world manufacturing application related to line balancing optimization is introduced, along with a first (exploratory) decomposition-based framework to deal with this new case study.

Future research directions

The mechanism based on task abstraction and iterative task decomposition introduced in Chapter 2 could be greatly improved by the use of more advanced or finer abstraction strategies, for instance, considering abstractions using cumulative resources that could represent a percentage of consumption of a resource by a subtask. The **information transferred between iterations** could be also reviewed, so that the search space doesn't turn out to be significantly reduced as is the case with the *start-to-start* precedence constraints that were considered.

The complementary of the cuts along with the surrogate-based strategy leads to the idea of merging them in a portfolio of cuts. This idea might be further refined in upcoming related research, for which restart strategies when solutions found by the two layers cannot be improved might be considered. Also, the proposed approaches can be extended to other scheduling problems involving complex setup operations between the main tasks. For all the introduced decomposition mechanisms, the settings concerning the allowed time per iteration and the learning parameters should also be revised.

The idea of an *incomplete learning* for both the surrogate-based strategy and the cut-generation approach could be also explored. The intention could be to adopt mechanisms already existing in the field of Machine Learning, for instance, adding an artificial neural network to represent in layer L1 the content of layer L2. This idea is already being used by some connected works in literature.

On the modelling side, more generic frameworks able to deal with HSPs with more general classes of resources could be considered. For instance, it could be useful to handle cumulative resources as mentioned in the last case study explored. A challenge

in this last case is to be able to manage a trade-off between genericity (as in HTN planning) and efficiency in tackling a given OR problem.

The final exploratory part of this thesis might lead to look further to tackle more real-world applications which can be adapted and treated as HSPs. Also, we can seek to explore more alternatives on decomposition approaches which involve other additional paradigms to CP.

Appendix A

Annexes

A.1 Comparison: Surrogate-Based Strategy vs. Cut Generation Approach

A.1.1 The simplest case: Link isolation

A.1.1.1 Small field problem instances

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	40	40	40	**40**	**40**	**40**	**40**	**40**	**40**	**40**
2	51	62	62	**51**	**51**	**51**	**51**	**51**	**51**	**51**
3	74	80	80	**74**	**74**	**74**	**74**	**74**	**74**	**74**
4	95	101	101	95	95	95	95	95	99	**93**
5	114	133	133	**114**	**114**	**114**	**114**	**114**	116	116
6	131	166	166	**131**	**131**	**131**	**131**	**131**	154	154
7	158	186	186	158	158	**155**	156	**155**	186	186
8	182	230	230	176	176	**174**	178	177	230	230
9	200	274	274	199	200	198	200	**196**	274	273
10	235	-	-	225	229	220	**217**	225	-	-
11	251	-	-	**245**	251	247	247	247	-	-
12	277	-	-	**270**	**270**	280	276	276	-	-
13	313	-	-	298	294	**293**	301	305	-	-
14	317	-	-	**317**	318	323	**317**	**317**	-	-
15	356	-	-	**338**	343	341	350	356	-	-

Table A.1: Makespan results for the *second* random test for anti-collision mechanism *Link isolation* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	40	0,00%	0,00%	0,00%	0,00%	0,00%
2	51	0,00%	0,00%	0,00%	0,00%	0,00%
3	74	0,00%	0,00%	0,00%	0,00%	0,00%
4	93	2,15%	2,15%	2,15%	2,15%	2,15%
5	114	0,00%	0,00%	0,00%	0,00%	0,00%
6	131	0,00%	0,00%	0,00%	0,00%	0,00%
7	155	1,94%	1,94%	0,00%	0,65%	0,00%
8	174	1,15%	1,15%	0,00%	2,30%	1,72%
9	196	1,53%	2,04%	1,02%	2,04%	0,00%
10	217	3,69%	5,53%	1,38%	0,00%	3,69%
11	245	0,00%	2,45%	0,82%	0,82%	0,82%
12	270	0,00%	0,00%	3,70%	2,22%	2,22%
13	293	1,71%	0,34%	0,00%	2,73%	4,10%
14	317	0,00%	0,32%	1,89%	0,00%	0,00%
15	338	0,00%	1,48%	0,89%	3,55%	5,33%
Average Gap		**0,81%**	**1,16%**	**0,79%**	**1,10%**	**1,33%**

Table A.2: Gap to the best known solution for the *second* random test for anti-collision mechanism *Link isolation* on *small* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	40	40	40	**40**	**40**	**40**	**40**	**40**	**40**	**40**
2	51	62	62	**51**	**51**	**51**	**51**	**51**	**51**	**51**
3	74	80	80	**74**	**74**	**74**	**74**	**74**	**74**	**74**
4	95	101	101	95	95	95	95	95	99	**93**
5	114	133	133	**114**	**114**	**114**	**114**	**114**	116	116
6	131	166	166	**131**	**131**	**131**	**131**	**131**	154	154
7	158	186	186	158	158	**155**	156	**155**	186	186
8	182	230	230	176	176	**174**	178	177	230	230
9	200	274	274	200	199	198	200	**196**	274	273
10	235	-	-	227	235	220	**219**	225	-	-
11	251	-	-	**245**	251	247	247	247	-	-
12	277	-	-	270	**269**	283	276	276	-	-
13	313	-	-	300	**289**	294	301	305	-	-
14	317	-	-	**317**	**317**	323	**317**	**317**	-	-
15	356	-	-	343	343	**341**	350	356	-	-

Table A.3: Makespan results for the *third* random test for anti-collision mechanism *Link isolation* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	40	0,00%	0,00%	0,00%	0,00%	0,00%
2	51	0,00%	0,00%	0,00%	0,00%	0,00%
3	74	0,00%	0,00%	0,00%	0,00%	0,00%
4	93	2,15%	2,15%	2,15%	2,15%	2,15%
5	114	0,00%	0,00%	0,00%	0,00%	0,00%
6	131	0,00%	0,00%	0,00%	0,00%	0,00%
7	155	1,94%	1,94%	0,00%	0,65%	0,00%
8	174	1,15%	1,15%	0,00%	2,30%	1,72%
9	196	2,04%	1,53%	1,02%	2,04%	0,00%
10	219	3,65%	7,31%	0,46%	0,00%	2,74%
11	245	0,00%	2,45%	0,82%	0,82%	0,82%
12	269	0,37%	0,00%	5,20%	2,60%	2,60%
13	289	3,81%	0,00%	1,73%	4,15%	5,54%
14	317	0,00%	0,00%	1,89%	0,00%	0,00%
15	341	0,59%	0,59%	0,00%	2,64%	4,40%
Average Gap		**1,05%**	**1,14%**	**0,88%**	**1,16%**	**1,33%**

Table A.4: Gap to the best known solution for the *third* random test for anti-collision mechanism *Link isolation* on *small* field problem instances.

A.1.1.2 Medium field problem instances

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,20	0,23	0,23	0,20	0,19	0,36	0,27	0,18	0,23	0,23
2	0,31	0,56	0,56	0,31	0,31	0,44	0,32	0,35	0,56	0,56
3	15,56	4,46	4,46	15,56	15,97	15,24	15,52	15,40	33,47	33,47
4	15,25	25,79	25,79	15,25	15,22	15,35	15,28	15,30	58,55	58,55
5	10,23	27,05	27,05	10,23	10,22	10,23	10,28	10,31	71,30	71,30
6	10,23	170,56	170,56	10,23	10,45	10,23	10,23	10,23	197,36	197,36
7	10,69	-	358,76	42,69	21,57	21,93	21,98	23,52	-	470,11
8	13,23	-	-	157,72	12,67	12,69	12,66	150,96	-	-
9	15,99	-	-	105,96	16,14	189,07	64,75	271,34	-	-
10	22,67	-	-	66,65	64,75	242,63	22,58	22,03	-	-
11	28,40	-	-	28,40	25,55	25,54	25,87	25,53	-	-
12	31,65	-	-	90,70	93,02	32,35	32,35	32,16	-	-
13	38,83	-	-	38,83	196,64	38,80	40,06	267,93	-	-
14	50,11	-	-	57,80	55,88	55,97	50,10	56,06	-	-
15	79,79	-	-	233,36	233,71	79,88	79,81	80,16	-	-

Table A.5: Time results for the *first* random test for anti-collision mechanism *Link isolation* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	46	46	46	**46**	**46**	**46**	**46**	**46**	**46**	**46**
2	57	57	57	**57**	**57**	**57**	**57**	**57**	**57**	**57**
3	80	92	92	**80**	**80**	**80**	**80**	**80**	**80**	**80**
4	99	127	127	**99**	**99**	**99**	**99**	**99**	111	111
5	124	160	160	**124**	**124**	**124**	**124**	**124**	143	143
6	179	226	226	**179**	**179**	**179**	**179**	**179**	225	225
7	232	-	330	**227**	**227**	**227**	**227**	**227**	-	304
8	275	-	-	**262**	264	264	264	**262**	-	-
9	312	-	-	**303**	309	**303**	**303**	**303**	-	-
10	442	-	-	397	397	**357**	412	433	-	-
11	455	-	-	401	**373**	**373**	375	**373**	-	-
12	464	-	-	457	**438**	448	448	459	-	-
13	582	-	-	582	**559**	582	560	579	-	-
14	727	-	-	727	**695**	**695**	**695**	**695**	-	-
15	767	-	-	**754**	**754**	767	767	767	-	-

Table A.6: Makespan results for the *first* random test for anti-collision mechanism *Link isolation* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	46	0,00%	0,00%	0,00%	0,00%	0,00%
2	57	0,00%	0,00%	0,00%	0,00%	0,00%
3	80	0,00%	0,00%	0,00%	0,00%	0,00%
4	99	0,00%	0,00%	0,00%	0,00%	0,00%
5	124	0,00%	0,00%	0,00%	0,00%	0,00%
6	179	0,00%	0,00%	0,00%	0,00%	0,00%
7	227	0,00%	0,00%	0,00%	0,00%	0,00%
8	262	0,00%	0,76%	0,76%	0,76%	0,00%
9	303	0,00%	1,98%	0,00%	0,00%	0,00%
10	357	11,20%	11,20%	0,00%	15,41%	21,29%
11	373	7,51%	0,00%	0,00%	0,54%	0,00%
12	438	4,34%	0,00%	2,28%	2,28%	4,79%
13	559	4,11%	0,00%	4,11%	0,18%	3,58%
14	695	4,60%	0,00%	0,00%	0,00%	0,00%
15	754	0,00%	0,00%	1,72%	1,72%	1,72%
Average Gap		**2,12%**	**0,93%**	**0,59%**	**1,39%**	**2,09%**

Table A.7: Gap to the best known solution for the *first* random test for anti-collision mechanism *Link isolation* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	46	46	46	**46**	**46**	**46**	**46**	**46**	**46**	**46**
2	57	57	57	**57**	**57**	**57**	**57**	**57**	**57**	**57**
3	80	92	92	**80**	**80**	**80**	**80**	**80**	**80**	**80**
4	99	127	127	**99**	**99**	**99**	**99**	**99**	111	111
5	124	160	160	**124**	**124**	**124**	**124**	**124**	143	143
6	179	226	226	**179**	**179**	**179**	**179**	**179**	225	225
7	232	-	330	**227**	**227**	**227**	**227**	**227**	-	304
8	275	-	-	**262**	263	264	264	263	-	-
9	312	-	-	**303**	309	**303**	**303**	**303**	-	-
10	442	-	-	402	397	**377**	406	411	-	-
11	455	-	-	401	382	**373**	375	**373**	-	-
12	464	-	-	457	**438**	448	448	459	-	-
13	582	-	-	576	**559**	582	560	579	-	-
14	727	-	-	727	**695**	702	**695**	**695**	-	-
15	767	-	-	**754**	**754**	767	767	767	-	-

Table A.8: Makespan results for the *second* random test for anti-collision mechanism *Link isolation* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	46	0,00%	0,00%	0,00%	0,00%	0,00%
2	57	0,00%	0,00%	0,00%	0,00%	0,00%
3	80	0,00%	0,00%	0,00%	0,00%	0,00%
4	99	0,00%	0,00%	0,00%	0,00%	0,00%
5	124	0,00%	0,00%	0,00%	0,00%	0,00%
6	179	0,00%	0,00%	0,00%	0,00%	0,00%
7	227	0,00%	0,00%	0,00%	0,00%	0,00%
8	262	0,00%	0,38%	0,76%	0,76%	0,38%
9	303	0,00%	1,98%	0,00%	0,00%	0,00%
10	377	6,63%	5,31%	0,00%	7,69%	9,02%
11	373	7,51%	2,41%	0,00%	0,54%	0,00%
12	438	4,34%	0,00%	2,28%	2,28%	4,79%
13	559	3,04%	0,00%	4,11%	0,18%	3,58%
14	695	4,60%	0,00%	1,01%	0,00%	0,00%
15	754	0,00%	0,00%	1,72%	1,72%	1,72%
Average Gap		**1,74%**	**0,67%**	**0,66%**	**0,88%**	**1,30%**

Table A.9: Gap to the best known solution for the *second* random test for anti-collision mechanism *Link isolation* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	46	46	46	**46**	**46**	**46**	**46**	**46**	**46**	**46**
2	57	57	57	**57**	**57**	**57**	**57**	**57**	**57**	**57**
3	80	92	92	**80**	**80**	**80**	**80**	**80**	**80**	**80**
4	99	127	127	**99**	**99**	**99**	**99**	**99**	111	111
5	124	160	160	**124**	**124**	**124**	**124**	**124**	143	143
6	179	226	226	**179**	**179**	**179**	**179**	**179**	225	225
7	232	-	330	**227**	**227**	**227**	**227**	**227**	-	304
8	275	-	-	**262**	264	264	264	**262**	-	-
9	312	-	-	**303**	305	**303**	**303**	**303**	-	-
10	442	-	-	**367**	397	**367**	412	433	-	-
11	455	-	-	398	**373**	**373**	375	**373**	-	-
12	464	-	-	457	**438**	448	448	459	-	-
13	582	-	-	573	564	582	**560**	579	-	-
14	727	-	-	727	**695**	**695**	701	**695**	-	-
15	767	-	-	**754**	**754**	767	767	767	-	-

Table A.10: Makespan results for the *third* random test for anti-collision mechanism *Link isolation* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	46	0,00%	0,00%	0,00%	0,00%	0,00%
2	57	0,00%	0,00%	0,00%	0,00%	0,00%
3	80	0,00%	0,00%	0,00%	0,00%	0,00%
4	99	0,00%	0,00%	0,00%	0,00%	0,00%
5	124	0,00%	0,00%	0,00%	0,00%	0,00%
6	179	0,00%	0,00%	0,00%	0,00%	0,00%
7	227	0,00%	0,00%	0,00%	0,00%	0,00%
8	262	0,00%	0,76%	0,76%	0,76%	0,00%
9	303	0,00%	0,66%	0,00%	0,00%	0,00%
10	367	0,00%	8,17%	0,00%	12,26%	17,98%
11	373	6,70%	0,00%	0,00%	0,54%	0,00%
12	438	4,34%	0,00%	2,28%	2,28%	4,79%
13	560	2,32%	0,71%	3,93%	0,00%	3,39%
14	695	4,60%	0,00%	0,00%	0,86%	0,00%
15	754	0,00%	0,00%	1,72%	1,72%	1,72%
Average Gap		**1,20%**	**0,69%**	**0,58%**	**1,23%**	**1,86%**

Table A.11: Gap to the best known solution for the *third* random test for anti-collision mechanism *Link isolation* on *medium* field problem instances.

A.1.1.3 Large field problem instances

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,17	0,16	0,16	0,17	0,20	0,24	0,19	0,16	0,16	0,16
2	0,22	0,30	0,30	0,22	0,22	0,24	0,29	0,32	0,67	0,67
3	15,20	1,63	1,63	15,20	15,44	15,39	15,24	15,19	3,11	3,11
4	15,17	23,47	23,47	15,17	15,17	15,26	15,46	15,17	32,59	32,59
5	10,19	51,08	51,08	10,19	10,19	20,58	20,64	20,61	238,02	238,02
6	10,24	41,91	41,91	135,09	55,18	31,78	83,99	21,17	113,94	100,09
7	10,22	-	470,44	61,32	30,45	128,55	41,65	41,63	-	470,44
8	12,25	-	325,73	78,62	42,31	172,84	269,11	40,10	-	325,73
9	15,51	-	-	188,57	105,17	45,58	45,66	45,45	-	-
10	21,68	-	-	105,99	105,97	88,11	105,78	148,51	-	-
11	25,23	-	-	75,96	125,36	25,07	25,33	75,51	-	-
12	30,60	-	-	270,52	150,81	90,38	90,40	210,97	-	-
13	37,64	-	-	111,81	37,03	37,12	37,48	37,12	-	-
14	50,14	-	-	50,14	152,23	251,13	250,66	50,99	-	-
15	75,96	-	-	225,98	225,45	225,82	225,74	225,83	-	-

Table A.12: Time results for the *first* random test for anti-collision mechanism *Link isolation* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	55	55	55	55	**55**	**55**	**55**	**55**	**55**	**55**
2	73	85	85	**73**	**73**	**73**	**73**	**73**	**73***	**73***
3	109	116	116	109	109	109	109	109	**106**	**106**
4	136	153	153	136	136	136	**133**	136	**133**	**133**
5	160	197	197	160	160	**157**	**157**	**157**	167	167
6	189	255	255	**186**	**186**	187	**186**	**186**	226	226
7	229	-	331	**214**	217	**214**	217	**214**	-	331
8	262	-	317	250	255	243	245	**242**	-	317
9	292	-	-	284	282	**280**	**280**	291	-	-
10	330	-	-	**302**	305	309	309	319	-	-
11	333	-	-	330	339	333	333	**329**	-	-
12	395	-	-	371	**365**	368	374	366	-	-
13	410	-	-	**405**	410	410	410	410	-	-
14	440	-	-	440	**437**	**437**	**437**	440	-	-
15	491	-	-	486	486	**476**	484	481	-	-

Table A.13: Makespan results for the *first* random test for anti-collision mechanism *Link isolation* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	55	0,00%	0,00%	0,00%	0,00%	0,00%
2	73	0,00%	0,00%	0,00%	0,00%	0,00%
3	106	2,83%	2,83%	2,83%	2,83%	2,83%
4	133	2,26%	2,26%	2,26%	0,00%	2,26%
5	157	1,91%	1,91%	0,00%	0,00%	0,00%
6	186	0,00%	0,00%	0,54%	0,00%	0,00%
7	214	0,00%	1,40%	0,00%	1,40%	0,00%
8	242	3,31%	5,37%	0,41%	1,24%	0,00%
9	280	1,43%	0,71%	0,00%	0,00%	3,93%
10	302	0,00%	0,99%	2,32%	2,32%	5,63%
11	329	0,30%	3,04%	1,22%	1,22%	0,00%
12	365	1,64%	0,00%	0,82%	2,47%	0,27%
13	405	0,00%	1,23%	1,23%	1,23%	1,23%
14	437	0,69%	0,00%	0,00%	0,00%	0,69%
15	476	2,10%	2,10%	0,00%	1,68%	1,05%
Average Gap		**1,10%**	**1,46%**	**0,78%**	**0,96%**	**1,19%**

Table A.14: Gap to the best known solution for the *first* random test for anti-collision mechanism *Link isolation* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	55	55	55	**55**	**55**	**55**	**55**	**55**	**55**	**55**
2	73	85	85	**73**	**73**	**73**	**73**	**73**	**73***	**73***
3	109	116	116	109	109	109	109	109	**106**	**106**
4	136	153	153	136	136	136	**133**	136	**133**	**133**
5	160	197	197	160	160	**157**	**157**	**157**	167	167
6	189	255	255	**186**	**186**	187	**186**	**186**	226	226
7	229	-	331	**214**	217	**214**	217	**214**	-	331
8	262	-	317	250	255	247	245	**242**	-	317
9	292	-	-	284	282	**280**	**280**	291	-	-
10	330	-	-	**302**	305	309	309	313	-	-
11	333	-	-	330	339	333	333	**329**	-	-
12	395	-	-	371	**365**	368	374	370	-	-
13	410	-	-	**407**	410	410	410	410	-	-
14	440	-	-	440	**437**	**437**	**437**	440	-	-
15	491	-	-	486	486	**476**	484	481	-	-

Table A.15: Makespan results for the *second* random test for anti-collision mechanism *Link isolation* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	55	0,00%	0,00%	0,00%	0,00%	0,00%
2	73	0,00%	0,00%	0,00%	0,00%	0,00%
3	106	2,83%	2,83%	2,83%	2,83%	2,83%
4	133	2,26%	2,26%	2,26%	0,00%	2,26%
5	157	1,91%	1,91%	0,00%	0,00%	0,00%
6	186	0,00%	0,00%	0,54%	0,00%	0,00%
7	214	0,00%	1,40%	0,00%	1,40%	0,00%
8	242	3,31%	5,37%	2,07%	1,24%	0,00%
9	280	1,43%	0,71%	0,00%	0,00%	3,93%
10	302	0,00%	0,99%	2,32%	2,32%	3,64%
11	329	0,30%	3,04%	1,22%	1,22%	0,00%
12	365	1,64%	0,00%	0,82%	2,47%	1,37%
13	407	0,00%	0,74%	0,74%	0,74%	0,74%
14	437	0,69%	0,00%	0,00%	0,00%	0,69%
15	476	2,10%	2,10%	0,00%	1,68%	1,05%
Average Gap		**1,10%**	**1,42%**	**0,85%**	**0,93%**	**1,10%**

Table A.16: Gap to the best known solution for the *second* random test for anti-collision mechanism *Link isolation* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	55	55	55	**55**	**55**	**55**	**55**	**55**	**55**	**55**
2	73	85	85	**73**	**73**	**73**	**73**	**73**	**73***	**73***
3	109	116	116	109	109	109	109	109	**106**	**106**
4	136	153	153	136	136	136	**133**	136	**133**	**133**
5	160	197	197	160	160	**157**	**157**	**157**	167	167
6	189	255	255	**186**	**186**	187	**186**	**186**	226	226
7	229	-	331	**214**	217	**214**	217	**214**	-	331
8	262	-	317	250	251	243	245	**242**	-	317
9	292	-	-	284	282	**280**	**280**	291	-	-
10	330	-	-	**302**	305	309	309	315	-	-
11	333	-	-	330	339	333	333	**329**	-	-
12	395	-	-	371	**365**	368	374	369	-	-
13	410	-	-	**405**	410	410	410	410	-	-
14	440	-	-	440	**437**	**437**	**437**	440	-	-
15	491	-	-	486	486	**476**	484	481	-	-

Table A.17: Makespan results for the *third* random test for anti-collision mechanism *Link isolation* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	55	0,00%	0,00%	0,00%	0,00%	0,00%
2	73	0,00%	0,00%	0,00%	0,00%	0,00%
3	106	2,83%	2,83%	2,83%	2,83%	2,83%
4	133	2,26%	2,26%	2,26%	0,00%	2,26%
5	157	1,91%	1,91%	0,00%	0,00%	0,00%
6	186	0,00%	0,00%	0,54%	0,00%	0,00%
7	214	0,00%	1,40%	0,00%	1,40%	0,00%
8	242	3,31%	3,72%	0,41%	1,24%	0,00%
9	280	1,43%	0,71%	0,00%	0,00%	3,93%
10	302	0,00%	0,99%	2,32%	2,32%	4,30%
11	329	0,30%	3,04%	1,22%	1,22%	0,00%
12	365	1,64%	0,00%	0,82%	2,47%	1,10%
13	405	0,00%	1,23%	1,23%	1,23%	1,23%
14	437	0,69%	0,00%	0,00%	0,00%	0,69%
15	476	2,10%	2,10%	0,00%	1,68%	1,05%
Average Gap		**1,10%**	**1,35%**	**0,78%**	**0,96%**	**1,16%**

Table A.18: Gap to the best known solution for the *third* random test for anti-collision mechanism *Link isolation* on *large* field problem instances.

A.1.2 Link and waypoint isolation and Minimum handover

A.1.2.1 Small field problem instances

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	42	42	42	**42**	**42**	**42**	**42**	**42**	**42**	**42**
2	52	52	52	**52**	**52**	**52**	**52**	**52**	**52***	**52***
3	77	112	112	**76**	**76**	77	**76**	**76**	**76**	**76**
4	93	102	102	96	**93**	**93**	**93**	**93**	96	**93**
5	148	129	129	**115**	**115**	**115**	**115**	**115**	121	118
6	139	166	166	**133**	**133**	136	**133**	**133**	154	154
7	162	202	202	**156**	**156**	**156**	**156**	**156**	195	195
8	195	246	246	182	182	179	**176**	**176**	240	240
9	213	-	-	201	203	199	198	**195**	-	-
10	277	-	-	**224**	229	225	225	226	-	-
11	292	-	-	255	249	252	**246**	250	-	-
12	296	-	-	275	269	276	279	**265**	-	-
13	336	-	-	291	300	299	**285**	293	-	-
14	333	-	-	328	322	316	320	**313**	-	-
15	381	-	-	347	**329**	339	**329**	**329**	-	-

Table A.19: Makespan results for the *second* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	42	0,00%	0,00%	0,00%	0,00%	0,00%
2	52	0,00%	0,00%	0,00%	0,00%	0,00%
3	76	0,00%	0,00%	1,32%	0,00%	0,00%
4	93	3,23%	0,00%	0,00%	0,00%	0,00%
5	115	0,00%	0,00%	0,00%	0,00%	0,00%
6	133	0,00%	0,00%	2,26%	0,00%	0,00%
7	156	0,00%	0,00%	0,00%	0,00%	0,00%
8	176	3,41%	3,41%	1,70%	0,00%	0,00%
9	195	3,08%	4,10%	2,05%	1,54%	0,00%
10	224	0,00%	2,23%	0,45%	0,45%	0,89%
11	246	3,66%	1,22%	2,44%	0,00%	1,63%
12	265	3,77%	1,51%	4,15%	5,28%	0,00%
13	285	2,11%	5,26%	4,91%	0,00%	2,81%
14	313	4,79%	2,88%	0,96%	2,24%	0,00%
15	329	5,47%	0,00%	3,04%	0,00%	0,00%
Average Gap		**1,97%**	**1,37%**	**1,55%**	**0,63%**	**0,36%**

Table A.20: Gap to the best known solution for the *second* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *small* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	42	42	42	**42**	**42**	**42**	**42**	**42**	**42**	**42**
2	52	52	52	**52**	**52**	**52**	**52**	**52**	**52***	**52***
3	77	112	112	**76**	**76**	77	**76**	**76**	**76**	**76**
4	93	102	102	96	**93**	**93**	**93**	**93**	96	**93**
5	148	129	129	**115**	**115**	**115**	**115**	**115**	121	118
6	139	166	166	**133**	**133**	136	**133**	**133**	154	154
7	162	202	202	**156**	**156**	**156**	**156**	**156**	195	195
8	195	246	246	182	180	179	**176**	**176**	240	240
9	213	-	-	201	201	199	**196**	198	-	-
10	277	-	-	224	229	225	**223**	226	-	-
11	292	-	-	255	249	253	**246**	250	-	-
12	296	-	-	277	**265**	276	279	**265**	-	-
13	336	-	-	291	300	305	**285**	297	-	-
14	333	-	-	328	326	316	320	**313**	-	-
15	381	-	-	347	**329**	342	**329**	**329**	-	-

Table A.21: Makespan results for the *third* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	42	0,00%	0,00%	0,00%	0,00%	0,00%
2	52	0,00%	0,00%	0,00%	0,00%	0,00%
3	76	0,00%	0,00%	1,32%	0,00%	0,00%
4	93	3,23%	0,00%	0,00%	0,00%	0,00%
5	115	0,00%	0,00%	0,00%	0,00%	0,00%
6	133	0,00%	0,00%	2,26%	0,00%	0,00%
7	156	0,00%	0,00%	0,00%	0,00%	0,00%
8	176	3,41%	2,27%	1,70%	0,00%	0,00%
9	196	2,55%	2,55%	1,53%	0,00%	1,02%
10	223	0,45%	2,69%	0,90%	0,00%	1,35%
11	246	3,66%	1,22%	2,85%	0,00%	1,63%
12	265	4,53%	0,00%	4,15%	5,28%	0,00%
13	285	2,11%	5,26%	7,02%	0,00%	4,21%
14	313	4,79%	4,15%	0,96%	2,24%	0,00%
15	329	5,47%	0,00%	3,95%	0,00%	0,00%
Average Gap		**2,01%**	**1,21%**	**1,78%**	**0,50%**	**0,55%**

Table A.22: Gap to the best known solution for the *third* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *small* field problem instances.

A.1.2.2 Medium field problem instances

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,75	2,80	2,80	0,54	0,55	0,18	0,19	0,21	3,15	3,15
2	0,32	1,06	1,06	0,32	1,05	0,33	0,33	0,45	1,06	1,06
3	30,24	5,20	5,20	30,24	15,25	15,24	15,24	15,26	18,26	18,26
4	30,22	51,25	51,16	30,22	15,18	15,22	15,22	15,80	195,52	437,91
5	30,24	31,87	31,04	30,24	10,21	10,60	15,25	15,47	206,80	218,80
6	30,21	-	1697,73	30,21	15,26	10,29	15,25	15,27	-	1697,73
7	16,24	-	-	75,63	75,96	54,77	75,62	107,31	-	-
8	19,39	-	-	153,29	93,36	91,44	91,68	151,74	-	-
9	30,76	-	-	244,85	55,43	31,04	30,81	30,99	-	-
10	31,59	-	-	259,92	281,91	264,32	115,00	38,59	-	-
11	37,51	-	-	37,51	25,09	113,90	50,55	50,60	-	-
12	30,02	-	-	30,02	39,79	112,02	226,11	82,93	-	-
13	83,78	-	-	83,78	83,91	53,22	84,05	151,61	-	-
14	50,12	-	-	68,17	228,38	155,89	75,11	228,92	-	-
15	131,64	-	-	231,25	230,50	150,03	157,06	157,15	-	-

Table A.23: Time results for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	48	70	70	**48**	**48**	**48**	**48**	**48**	**48**	**48**
2	58	58	58	**58**	**58**	**58**	**58**	**58**	**58**	**58**
3	82	92	92	**82**	**82**	**82**	**82**	**82**	**82**	**82**
4	99	129	129	**99**	**99**	**99**	**99**	**99**	108	106
5	124	202	202	**124**	**124**	**124**	**124**	**124**	151	151
6	182	-	252	**182**	**182**	**182**	**182**	**182**	-	252
7	246	-	-	232	232	**229**	232	**229**	-	-
8	268	-	-	265	**264**	**264**	**264**	**264**	-	-
9	330	-	-	307	311	**305**	**305**	**305**	-	-
10	435	-	-	**356**	386	375	382	384	-	-
11	430	-	-	382	382	**376**	382	382	-	-
12	466	-	-	466	452	**437**	440	448	-	-
13	514	-	-	514	514	534	514	**510**	-	-
14	722	-	-	698	**621**	654	638	668	-	-
15	787	-	-	716	**692**	720	720	720	-	-

Table A.24: Makespan results for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	48	0,00%	0,00%	0,00%	0,00%	0,00%
2	58	0,00%	0,00%	0,00%	0,00%	0,00%
3	82	0,00%	0,00%	0,00%	0,00%	0,00%
4	99	0,00%	0,00%	0,00%	0,00%	0,00%
5	124	0,00%	0,00%	0,00%	0,00%	0,00%
6	182	0,00%	0,00%	0,00%	0,00%	0,00%
7	229	1,31%	1,31%	0,00%	1,31%	0,00%
8	264	0,38%	0,00%	0,00%	0,00%	0,00%
9	305	0,66%	1,97%	0,00%	0,00%	0,00%
10	356	0,00%	8,43%	5,34%	7,30%	7,87%
11	376	1,60%	1,60%	0,00%	1,60%	1,60%
12	437	6,64%	3,43%	0,00%	0,69%	2,52%
13	510	0,78%	0,78%	4,71%	0,78%	0,00%
14	621	12,40%	0,00%	5,31%	2,74%	7,57%
15	692	3,47%	0,00%	4,05%	4,05%	4,05%
Average Gap		**1,82%**	**1,17%**	**1,29%**	**1,23%**	**1,57%**

Table A.25: Gap to the best known solution for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	48	70	70	**48**	**48**	**48**	**48**	**48**	**48**	**48**
2	58	58	58	**58**	**58**	**58**	**58**	**58**	**58**	**58**
3	82	92	92	**82**	**82**	**82**	**82**	**82**	**82**	**82**
4	99	129	129	**99**	**99**	**99**	**99**	**99**	108	106
5	124	202	202	**124**	**124**	**124**	**124**	**124**	151	151
6	182	-	252	**182**	**182**	**182**	**182**	**182**	-	252
7	246	-	-	232	232	**229**	232	**229**	-	-
8	268	-	-	265	**264**	**264**	**264**	**264**	-	-
9	330	-	-	307	311	310	**305**	**305**	-	-
10	435	-	-	380	386	**375**	382	384	-	-
11	430	-	-	382	382	**376**	382	382	-	-
12	466	-	-	466	456	**437**	442	452	-	-
13	514	-	-	514	514	534	514	**510**	-	-
14	722	-	-	637	**621**	654	638	668	-	-
15	787	-	-	716	**692**	720	720	720	-	-

Table A.26: Makespan results for the *second* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	48	0,00%	0,00%	0,00%	0,00%	0,00%
2	58	0,00%	0,00%	0,00%	0,00%	0,00%
3	82	0,00%	0,00%	0,00%	0,00%	0,00%
4	99	0,00%	0,00%	0,00%	0,00%	0,00%
5	124	0,00%	0,00%	0,00%	0,00%	0,00%
6	182	0,00%	0,00%	0,00%	0,00%	0,00%
7	229	1,31%	1,31%	0,00%	1,31%	0,00%
8	264	0,38%	0,00%	0,00%	0,00%	0,00%
9	305	0,66%	1,97%	1,64%	0,00%	0,00%
10	375	1,33%	2,93%	0,00%	1,87%	2,40%
11	376	1,60%	1,60%	0,00%	1,60%	1,60%
12	437	6,64%	4,35%	0,00%	1,14%	3,43%
13	510	0,78%	0,78%	4,71%	0,78%	0,00%
14	621	2,58%	0,00%	5,31%	2,74%	7,57%
15	692	3,47%	0,00%	4,05%	4,05%	4,05%
Average Gap		**1,25%**	**0,86%**	**1,05%**	**0,90%**	**1,27%**

Table A.27: Gap to the best known solution for the *second* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	48	70	70	**48**	**48**	**48**	**48**	**48**	**48**	**48**
2	58	58	58	**58**	**58**	**58**	**58**	**58**	**58**	**58**
3	82	92	92	**82**	**82**	**82**	**82**	**82**	**82**	**82**
4	99	129	129	**99**	**99**	**99**	**99**	**99**	108	106
5	124	202	202	**124**	**124**	**124**	**124**	**124**	151	151
6	182	-	252	**182**	**182**	**182**	**182**	**182**	-	252
7	246	-	-	232	232	**229**	232	**229**	-	-
8	268	-	-	265	**264**	**264**	**264**	**264**	-	-
9	330	-	-	307	311	307	**305**	**305**	-	-
10	435	-	-	379	386	**375**	382	384	-	-
11	430	-	-	382	398	**376**	382	382	-	-
12	466	-	-	466	452	**437**	443	448	-	-
13	514	-	-	514	514	534	514	**510**	-	-
14	722	-	-	668	**631**	654	638	668	-	-
15	787	-	-	716	**712**	720	720	720	-	-

Table A.28: Makespan results for the *third* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	48	0,00%	0,00%	0,00%	0,00%	0,00%
2	58	0,00%	0,00%	0,00%	0,00%	0,00%
3	82	0,00%	0,00%	0,00%	0,00%	0,00%
4	99	0,00%	0,00%	0,00%	0,00%	0,00%
5	124	0,00%	0,00%	0,00%	0,00%	0,00%
6	182	0,00%	0,00%	0,00%	0,00%	0,00%
7	229	1,31%	1,31%	0,00%	1,31%	0,00%
8	264	0,38%	0,00%	0,00%	0,00%	0,00%
9	305	0,66%	1,97%	0,66%	0,00%	0,00%
10	375	1,07%	2,93%	0,00%	1,87%	2,40%
11	376	1,60%	5,85%	0,00%	1,60%	1,60%
12	437	6,64%	3,43%	0,00%	1,37%	2,52%
13	510	0,78%	0,78%	4,71%	0,78%	0,00%
14	631	5,86%	0,00%	3,65%	1,11%	5,86%
15	712	0,56%	0,00%	1,12%	1,12%	1,12%
Average Gap		**1,26%**	**1,09%**	**0,68%**	**0,61%**	**0,90%**

Table A.29: Gap to the best known solution for the *third* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *medium* field problem instances.

A.1.2.3 Large field problem instances

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,16	0,18	0,18	0,16	0,24	0,17	0,65	0,19	0,18	0,18
2	0,23	0,28	0,28	0,23	0,37	0,66	0,68	0,49	0,75	0,75
3	50,16	1,73	1,73	250,43	75,59	261,46	68,29	167,14	3,33	3,33
4	21,17	38,30	38,30	44,83	44,75	44,85	45,14	47,14	50,26	49,31
5	30,19	45,00	45,00	30,19	63,67	130,87	173,20	63,72	265,33	832,95
6	30,69	41,64	41,64	32,56	30,55	31,18	30,99	30,67	41,64	41,64
7	30,25	-	-	198,20	30,24	266,78	266,97	30,21	-	-
8	30,47	-	-	237,86	173,17	113,83	237,62	237,54	-	-
9	50,27	-	-	50,27	50,24	50,54	50,24	75,21	-	-
10	23,49	-	-	175,11	250,70	250,10	250,92	250,79	-	-
11	30,52	-	-	30,52	30,44	150,82	30,51	150,52	-	-
12	32,17	-	-	210,84	151,54	150,45	251,03	210,52	-	-
13	66,85	-	-	252,65	263,47	188,32	186,61	188,01	-	-
14	51,46	-	-	150,73	152,33	150,81	225,14	51,25	-	-
15	77,75	-	-	225,83	226,04	225,74	75,06	225,90	-	-

Table A.30: Time results for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	57	57	57	**57**	**57**	**57**	**57**	**57**	**57**	**57**
2	73	96	96	**73**	**73**	**73**	**73**	**73**	**73***	**73***
3	111	116	116	**106**	**106**	**106**	**106**	**106**	**106**	**106**
4	137	143	143	**133**	**133**	**133**	**133**	**133**	**133**	**133**
5	161	191	191	161	**157**	159	**157**	159	171	167
6	197	269	269	**193**	**193**	**193**	**193**	**193**	269	269
7	215	-	-	**214**	215	**214**	**214**	215	-	-
8	262	-	-	**242**	245	245	243	**242**	-	-
9	270	-	-	**270**	**270**	**270**	**270**	**270**	-	-
10	334	-	-	320	315	314	**303**	314	-	-
11	337	-	-	337	337	**333**	337	**333**	-	-
12	403	-	-	375	389	379	371	**369**	-	-
13	424	-	-	422	423	**407**	421	413	-	-
14	448	-	-	**431**	447	447	447	448	-	-
15	493	-	-	**465**	467	467	493	485	-	-

Table A.31: Makespan results for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	57	0,00%	0,00%	0,00%	0,00%	0,00%
2	73	0,00%	0,00%	0,00%	0,00%	0,00%
3	106	0,00%	0,00%	0,00%	0,00%	0,00%
4	133	0,00%	0,00%	0,00%	0,00%	0,00%
5	157	2,55%	0,00%	1,27%	0,00%	1,27%
6	193	0,00%	0,00%	0,00%	0,00%	0,00%
7	214	0,00%	0,47%	0,00%	0,00%	0,47%
8	242	0,00%	1,24%	1,24%	0,41%	0,00%
9	270	0,00%	0,00%	0,00%	0,00%	0,00%
10	303	5,61%	3,96%	3,63%	0,00%	3,63%
11	333	1,20%	1,20%	0,00%	1,20%	0,00%
12	369	1,63%	5,42%	2,71%	0,54%	0,00%
13	407	3,69%	3,93%	0,00%	3,44%	1,47%
14	431	0,00%	3,71%	3,71%	3,71%	3,94%
15	465	0,00%	0,43%	0,43%	6,02%	4,30%
Average Gap		**0,98%**	**1,36%**	**0,87%**	**1,02%**	**1,01%**

Table A.32: Gap to the best known solution for the *first* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	57	57	57	**57**	**57**	**57**	**57**	**57**	**57**	**57**
2	73	96	96	**73**	**73**	**73**	**73**	**73**	**73***	**73***
3	111	116	116	**106**	**106**	**106**	**106**	**106**	**106**	**106**
4	137	143	143	**133**	**133**	**133**	**133**	**133**	**133**	**133**
5	161	191	191	161	**157**	158	**157**	159	171	167
6	197	269	269	**193**	**193**	**193**	**193**	**193**	269	269
7	215	-	-	216	215	**214**	**214**	215	-	-
8	262	-	-	**242**	245	245	243	**242**	-	-
9	270	-	-	**270**	272	**270**	273	**270**	-	-
10	334	-	-	318	315	314	**307**	314	-	-
11	337	-	-	337	337	334	337	**333**	-	-
12	403	-	-	375	379	379	371	**369**	-	-
13	424	-	-	423	423	**407**	421	413	-	-
14	448	-	-	**431**	447	447	447	448	-	-
15	493	-	-	**465**	481	467	493	485	-	-

Table A.33: Makespan results for the *second* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	57	0,00%	0,00%	0,00%	0,00%	0,00%
2	73	0,00%	0,00%	0,00%	0,00%	0,00%
3	106	0,00%	0,00%	0,00%	0,00%	0,00%
4	133	0,00%	0,00%	0,00%	0,00%	0,00%
5	157	2,55%	0,00%	0,64%	0,00%	1,27%
6	193	0,00%	0,00%	0,00%	0,00%	0,00%
7	214	0,93%	0,47%	0,00%	0,00%	0,47%
8	242	0,00%	1,24%	1,24%	0,41%	0,00%
9	270	0,00%	0,74%	0,00%	1,11%	0,00%
10	307	3,58%	2,61%	2,28%	0,00%	2,28%
11	333	1,20%	1,20%	0,30%	1,20%	0,00%
12	369	1,63%	2,71%	2,71%	0,54%	0,00%
13	407	3,93%	3,93%	0,00%	3,44%	1,47%
14	431	0,00%	3,71%	3,71%	3,71%	3,94%
15	465	0,00%	3,44%	0,43%	6,02%	4,30%
Average Gap		**0,92%**	**1,34%**	**0,75%**	**1,10%**	**0,92%**

Table A.34: Gap to the best known solution for the *second* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	57	57	57	**57**	**57**	**57**	**57**	**57**	**57**	**57**
2	73	96	96	**73**	**73**	**73**	**73**	**73**	**73***	**73***
3	111	116	116	**106**	**106**	**106**	**106**	**106**	**106**	**106**
4	137	143	143	**133**	**133**	**133**	**133**	**133**	**133**	**133**
5	161	191	191	161	**133**	159	157	159	171	167
6	197	269	269	**193**	**193**	**193**	**193**	**193**	269	269
7	215	-	-	215	215	**214**	**214**	215	-	-
8	262	-	-	**242**	245	245	243	**242**	-	-
9	270	-	-	**270**	271	**270**	**270**	271	-	-
10	334	-	-	320	315	314	**305**	314	-	-
11	337	-	-	337	337	334	337	**333**	-	-
12	403	-	-	381	389	379	371	**369**	-	-
13	424	-	-	422	423	**411**	421	413	-	-
14	448	-	-	**431**	447	447	447	448	-	-
15	493	-	-	**465**	479	467	493	485	-	-

Table A.35: Makespan results for the *third* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	57	0,00%	0,00%	0,00%	0,00%	0,00%
2	73	0,00%	0,00%	0,00%	0,00%	0,00%
3	106	0,00%	0,00%	0,00%	0,00%	0,00%
4	133	0,00%	0,00%	0,00%	0,00%	0,00%
5	157	2,55%	0,00%	1,27%	0,00%	1,27%
6	193	0,00%	0,00%	0,00%	0,00%	0,00%
7	214	0,47%	0,47%	0,00%	0,00%	0,47%
8	242	0,00%	1,24%	1,24%	0,41%	0,00%
9	270	0,00%	0,37%	0,00%	0,00%	0,37%
10	305	4,92%	3,28%	2,95%	0,00%	2,95%
11	333	1,20%	1,20%	0,30%	1,20%	0,00%
12	369	3,25%	5,42%	2,71%	0,54%	0,00%
13	411	2,68%	2,92%	0,00%	2,43%	0,49%
14	431	0,00%	3,71%	3,71%	3,71%	3,94%
15	465	0,00%	3,01%	0,43%	6,02%	4,30%
Average Gap		**1,00%**	**1,44%**	**0,84%**	**0,95%**	**0,92%**

Table A.36: Gap to the best known solution for the *third* random test for anti-collision mechanism *Link and waypoint isolation and Minimum handover* on *large* field problem instances.

A.1.3 Path isolation

A.1.3.1 Small field problem instances

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	58	58	58	**58**	**58**	**58**	**58**	**58**	**58**	**58**
2	96	85	85	96	96	96	96	**85**	**85**	**85**
3	97	97	97	**97**	**97**	**97**	**97**	**97**	**97**	**97**
4	178	118	118	**111**	**111**	**111**	**111**	121	**111**	**111**
5	166	232	232	130	**127**	**127**	130	130	134	134
6	154	235	235	152	**149**	**149**	154	150	168	156
7	238	333	333	171	171	**168**	172	173	269	211
8	230	451	451	194	196	**190**	192	194	316	304
9	335	-	-	**222**	229	229	225	238	-	-
10	288	-	-	**252**	259	253	275	**252**	-	-
11	321	-	-	271	283	295	**269**	293	-	-
12	367	-	-	**305**	**305**	316	317	330	-	-
13	382	-	-	**304**	305	**304**	**304**	**304**	-	-
14	389	-	-	361	371	362	364	**359**	-	-
15	463	-	-	**393**	**393**	**393**	**393**	**393**	-	-

Table A.37: Makespan results for the *second* random test for anti-collision mechanism *Path isolation* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	58	0,00%	0,00%	0,00%	0,00%	0,00%
2	85	12,94%	12,94%	12,94%	12,94%	0,00%
3	97	0,00%	0,00%	0,00%	0,00%	0,00%
4	111	0,00%	0,00%	0,00%	0,00%	9,01%
5	127	2,36%	0,00%	0,00%	2,36%	2,36%
6	149	2,01%	0,00%	0,00%	3,36%	0,67%
7	168	1,79%	1,79%	0,00%	2,38%	2,98%
8	190	2,11%	3,16%	0,00%	1,05%	2,11%
9	222	0,00%	3,15%	3,15%	1,35%	7,21%
10	252	0,00%	2,78%	0,40%	9,13%	0,00%
11	269	0,74%	5,20%	9,67%	0,00%	8,92%
12	305	0,00%	0,00%	3,61%	3,93%	8,20%
13	304	0,00%	0,33%	0,00%	0,00%	0,00%
14	359	0,56%	3,34%	0,84%	1,39%	0,00%
15	393	0,00%	0,00%	0,00%	0,00%	0,00%
Average Gap		**1,50%**	**2,18%**	**2,04%**	**2,53%**	**2,76%**

Table A.38: Gap to the best known solution for the *second* random test for anti-collision mechanism *Path isolation* on *small* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	58	58	58	**58**	**58**	**58**	**58**	**58**	**58**	**58**
2	96	85	85	96	96	96	96	**85**	**85**	**85**
3	97	97	97	**97**	**97**	**97**	**97**	**97**	**97**	**97**
4	178	118	118	**111**	**111**	**111**	**111**	121	**111**	**111**
5	166	232	232	130	**127**	**127**	130	130	134	134
6	154	235	235	154	**149**	**149**	154	150	168	156
7	238	333	333	171	171	**168**	172	173	269	211
8	230	451	451	194	195	**190**	192	194	316	304
9	335	-	-	**222**	229	225	225	238	-	-
10	288	-	-	253	269	254	271	**252**	-	-
11	321	-	-	270	281	292	**269**	293	-	-
12	367	-	-	**305**	**305**	321	317	330	-	-
13	382	-	-	**304**	**304**	**304**	**304**	**304**	-	-
14	389	-	-	365	374	362	364	**359**	-	-
15	463	-	-	**393**	**393**	**393**	**393**	**393**	-	-

Table A.39: Makespan results for the *third* random test for anti-collision mechanism *Path isolation* on *small* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	58	0,00%	0,00%	0,00%	0,00%	0,00%
2	85	12,94%	12,94%	12,94%	12,94%	0,00%
3	97	0,00%	0,00%	0,00%	0,00%	0,00%
4	111	0,00%	0,00%	0,00%	0,00%	9,01%
5	127	2,36%	0,00%	0,00%	2,36%	2,36%
6	149	3,36%	0,00%	0,00%	3,36%	0,67%
7	168	1,79%	1,79%	0,00%	2,38%	2,98%
8	190	2,11%	2,63%	0,00%	1,05%	2,11%
9	222	0,00%	3,15%	1,35%	1,35%	7,21%
10	252	0,40%	6,75%	0,79%	7,54%	0,00%
11	269	0,37%	4,46%	8,55%	0,00%	8,92%
12	305	0,00%	0,00%	5,25%	3,93%	8,20%
13	304	0,00%	0,00%	0,00%	0,00%	0,00%
14	359	1,67%	4,18%	0,84%	1,39%	0,00%
15	393	0,00%	0,00%	0,00%	0,00%	0,00%
Average Gap		**1,67%**	**2,39%**	**1,98%**	**2,42%**	**2,76%**

Table A.40: Gap to the best known solution for the *third* random test for anti-collision mechanism *Path isolation* on *small* field problem instances.

A.1.3.2 Medium field problem instances

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,04	0,23	0,23	0,68	0,66	0,77	0,70	0,69	0,23	0,23
2	0,62	2,00	2,00	0,62	0,65	0,68	0,69	0,71	2,00	2,00
3	15,20	4,50	4,50	15,20	15,22	15,90	16,51	15,78	7,65	7,65
4	102,95	280,42	280,42	102,95	225,11	107,86	97,47	28,49	280,42	280,42
5	18,24	21,46	21,46	18,24	37,20	18,29	20,63	18,76	33,01	33,01
6	15,26	-	-	255,06	15,25	15,86	45,22	165,00	-	-
7	20,66	-	-	20,66	20,79	27,58	20,84	20,67	-	-
8	23,05	-	-	105,34	210,71	105,41	190,85	103,35	-	-
9	30,69	-	-	210,64	270,85	210,67	30,66	21,05	-	-
10	31,55	-	-	270,44	270,78	91,39	156,79	288,50	-	-
11	39,56	-	-	94,57	39,72	52,54	39,43	39,85	-	-
12	80,32	-	-	225,39	150,38	56,56	150,39	56,49	-	-
13	94,53	-	-	225,38	251,46	95,80	225,40	57,66	-	-
14	114,48	-	-	225,85	225,32	225,63	225,73	156,89	-	-
15	150,46	-	-	159,87	150,11	160,17	159,88	88,95	-	-

Table A.41: Time results for the *first* random test for anti-collision mechanism *Path isolation* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	70	70	70	**70**	**70**	**70**	**70**	**70**	**70**	**70**
2	108	97	97	108	108	108	108	108	**97**	**97**
3	109	174	174	**109**	**109**	**109**	**109**	**109**	**109**	**109**
4	190	151	151	**133**	**133**	**133**	**133**	**133**	151	151
5	139	235	235	**139**	**139**	**139**	**139**	**139**	218	218
6	235	-	-	**232**	235	235	**232**	**232**	-	-
7	327	-	-	**327**	**327**	**327**	**327**	**327**	-	-
8	387	-	-	**374**	**374**	375	386	375	-	-
9	431	-	-	428	430	430	431	**426**	-	-
10	651	-	-	**502**	504	511	551	551	-	-
11	672	-	-	**539**	**539**	**539**	**539**	**539**	-	-
12	669	-	-	622	**607**	632	**607**	632	-	-
13	804	-	-	786	**745**	804	780	769	-	-
14	1219	-	-	**830**	845	867	902	869	-	-
15	1154	-	-	**1154**	**1154**	**1154**	**1154**	1166	-	-

Table A.42: Makespan results for the *first* random test for anti-collision mechanism *Path isolation* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	70	0,00%	0,00%	0,00%	0,00%	0,00%
2	97	11,34%	11,34%	11,34%	11,34%	11,34%
3	109	0,00%	0,00%	0,00%	0,00%	0,00%
4	133	0,00%	0,00%	0,00%	0,00%	0,00%
5	139	0,00%	0,00%	0,00%	0,00%	0,00%
6	232	0,00%	1,29%	1,29%	0,00%	0,00%
7	327	0,00%	0,00%	0,00%	0,00%	0,00%
8	374	0,00%	0,00%	0,27%	3,21%	0,27%
9	426	0,47%	0,94%	0,94%	1,17%	0,00%
10	502	0,00%	0,40%	1,79%	9,76%	9,76%
11	539	0,00%	0,00%	0,00%	0,00%	0,00%
12	607	2,47%	0,00%	4,12%	0,00%	4,12%
13	745	5,50%	0,00%	7,92%	4,70%	3,22%
14	830	0,00%	1,81%	4,46%	8,67%	4,70%
15	1154	0,00%	0,00%	0,00%	0,00%	1,04%
Average Gap		**1,32%**	**1,05%**	**2,14%**	**2,59%**	**2,30%**

Table A.43: Gap to the best known solution for the *first* random test for anti-collision mechanism *Path isolation* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	70	70	70	**70**	**70**	**70**	**70**	**70**	**70**	**70**
2	108	97	97	108	108	108	108	108	**97**	**97**
3	109	174	174	**109**	**109**	**109**	**109**	**109**	**109**	**109**
4	190	151	151	**133**	**133**	**133**	**133**	**133**	151	151
5	139	235	235	**139**	**139**	**139**	**139**	**139**	218	218
6	235	-	-	**232**	235	235	**232**	**232**	-	-
7	327	-	-	**327**	**327**	**327**	**327**	**327**	-	-
8	387	-	-	**374**	**374**	375	386	375	-	-
9	431	-	-	428	430	430	431	**426**	-	-
10	651	-	-	**504**	506	511	538	529	-	-
11	672	-	-	**539**	**539**	**539**	**539**	**539**	-	-
12	669	-	-	622	612	632	**607**	632	-	-
13	804	-	-	786	**762**	789	780	769	-	-
14	1219	-	-	849	**845**	867	902	869	-	-
15	1154	-	-	**1154**	**1154**	**1154**	**1154**	1166	-	-

Table A.44: Makespan results for the *second* random test for anti-collision mechanism *Path isolation* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	70	0,00%	0,00%	0,00%	0,00%	0,00%
2	97	11,34%	11,34%	11,34%	11,34%	11,34%
3	109	0,00%	0,00%	0,00%	0,00%	0,00%
4	133	0,00%	0,00%	0,00%	0,00%	0,00%
5	139	0,00%	0,00%	0,00%	0,00%	0,00%
6	232	0,00%	1,29%	1,29%	0,00%	0,00%
7	327	0,00%	0,00%	0,00%	0,00%	0,00%
8	374	0,00%	0,00%	0,27%	3,21%	0,27%
9	426	0,47%	0,94%	0,94%	1,17%	0,00%
10	504	0,00%	0,40%	1,39%	6,75%	4,96%
11	539	0,00%	0,00%	0,00%	0,00%	0,00%
12	607	2,47%	0,82%	4,12%	0,00%	4,12%
13	762	3,15%	0,00%	3,54%	2,36%	0,92%
14	845	0,47%	0,00%	2,60%	6,75%	2,84%
15	1154	0,00%	0,00%	0,00%	0,00%	1,04%
Average Gap		**1,19%**	**0,99%**	**1,70%**	**2,11%**	**1,70%**

Table A.45: Gap to the best known solution for the *second* random test for anti-collision mechanism *Path isolation* on *medium* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	70	70	70	**70**	**70**	**70**	**70**	**70**	**70**	**70**
2	108	97	97	108	108	108	108	108	**97**	**97**
3	109	174	174	**109**	**109**	**109**	**109**	**109**	**109**	**109**
4	190	151	151	**133**	**133**	**133**	**133**	**133**	151	151
5	139	235	235	**139**	**139**	**139**	**139**	**139**	218	218
6	235	-	-	**232**	235	235	**232**	**232**	-	-
7	327	-	-	**327**	**327**	**327**	**327**	**327**	-	-
8	387	-	-	**374**	**374**	375	386	375	-	-
9	431	-	-	431	430	430	431	**426**	-	-
10	651	-	-	**502**	527	511	519	532	-	-
11	672	-	-	**539**	**539**	**539**	**539**	**539**	-	-
12	669	-	-	622	619	632	**607**	632	-	-
13	804	-	-	786	**745**	804	769	769	-	-
14	1219	-	-	**837**	845	867	858	869	-	-
15	1154	-	-	**1154**	**1154**	**1154**	**1154**	1166	-	-

Table A.46: Makespan results for the *third* random test for anti-collision mechanism *Path isolation* on *medium* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	70	0,00%	0,00%	0,00%	0,00%	0,00%
2	97	11,34%	11,34%	11,34%	11,34%	11,34%
3	109	0,00%	0,00%	0,00%	0,00%	0,00%
4	133	0,00%	0,00%	0,00%	0,00%	0,00%
5	139	0,00%	0,00%	0,00%	0,00%	0,00%
6	232	0,00%	1,29%	1,29%	0,00%	0,00%
7	327	0,00%	0,00%	0,00%	0,00%	0,00%
8	374	0,00%	0,00%	0,27%	3,21%	0,27%
9	426	1,17%	0,94%	0,94%	1,17%	0,00%
10	502	0,00%	4,98%	1,79%	3,39%	5,98%
11	539	0,00%	0,00%	0,00%	0,00%	0,00%
12	607	2,47%	1,98%	4,12%	0,00%	4,12%
13	745	5,50%	0,00%	7,92%	3,22%	3,22%
14	837	0,00%	0,96%	3,58%	2,51%	3,82%
15	1154	0,00%	0,00%	0,00%	0,00%	1,04%
Average Gap		**1,37%**	**1,43%**	**2,08%**	**1,66%**	**1,99%**

Table A.47: Gap to the best known solution for the *third* random test for anti-collision mechanism *Path isolation* on *medium* field problem instances.

A.1.3.3 Large field problem instances

	First Feasible Sol [s]			Best Solution [s]						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	0,39	0,38	0,38	0,49	0,27	0,39	0,48	0,39	0,38	0,38
2	0,38	0,74	0,74	0,91	1,40	1,89	1,67	0,38	1,69	1,69
3	30,38	3,25	3,25	31,66	30,58	30,39	30,38	30,38	6,15	6,15
4	50,07	54,87	54,87	50,66	52,11	53,12	50,43	50,07	126,74	126,74
5	25,22	54,18	54,18	120,56	101,32	104,18	104,71	75,86	138,84	782,95
6	15,65	114,28	114,28	186,31	150,08	75,41	15,70	225,23	176,46	176,46
7	15,52	-	-	277,65	270,65	111,43	207,64	75,92	-	-
8	18,44	-	-	270,87	189,62	262,32	167,55	91,30	-	-
9	75,20	-	-	37,20	75,29	37,23	37,19	75,20	-	-
10	50,51	-	-	250,31	79,14	75,03	51,84	250,29	-	-
11	38,07	-	-	262,45	75,06	186,30	250,30	186,47	-	-
12	22,81	-	-	210,50	225,24	190,97	150,42	190,75	-	-
13	31,96	-	-	150,22	283,95	281,28	270,78	90,43	-	-
14	31,85	-	-	111,35	164,49	150,44	270,78	111,21	-	-
15	51,55	-	-	250,45	225,39	76,69	250,49	250,56	-	-

Table A.48: Time results for the *first* random test for anti-collision mechanism *Path isolation* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	82	82	82	**82**	**82**	**82**	**82**	**82**	**82**	**82**
2	132	132	132	**121**	**121**	**121**	**121**	**121**	**121**	**121**
3	166	196	196	147	147	147	147	147	**139**	**139**
4	182	230	230	**182**	**182**	**182**	**182**	**182**	186	186
5	187	239	239	**181**	**181**	**181**	**181**	**181**	236	203
6	253	326	326	219	243	**218**	243	219	302	302
7	274	-	-	244	**238**	261	241	246	-	-
8	319	-	-	282	301	281	275	**267**	-	-
9	294	-	-	**294**	**294**	**294**	**294**	**294**	-	-
10	394	-	-	**337**	364	364	353	355	-	-
11	415	-	-	**376**	415	382	382	380	-	-
12	488	-	-	454	431	465	484	**429**	-	-
13	503	-	-	463	522	518	**452**	489	-	-
14	537	-	-	487	522	517	**454**	526	-	-
15	598	-	-	569	**567**	598	589	581	-	-

Table A.49: Makespan results for the *first* random test for anti-collision mechanism *Path isolation* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	82	0,00%	0,00%	0,00%	0,00%	0,00%
2	121	0,00%	0,00%	0,00%	0,00%	0,00%
3	139	5,76%	5,76%	5,76%	5,76%	5,76%
4	182	0,00%	0,00%	0,00%	0,00%	0,00%
5	181	0,00%	0,00%	0,00%	0,00%	0,00%
6	218	0,46%	11,47%	0,00%	11,47%	0,46%
7	238	2,52%	0,00%	9,66%	1,26%	3,36%
8	267	5,62%	12,73%	5,24%	3,00%	0,00%
9	294	0,00%	0,00%	0,00%	0,00%	0,00%
10	337	0,00%	8,01%	8,01%	4,75%	5,34%
11	376	0,00%	10,37%	1,60%	1,60%	1,06%
12	429	5,83%	0,47%	8,39%	12,82%	0,00%
13	452	2,43%	15,49%	14,60%	0,00%	8,19%
14	454	7,27%	14,98%	13,88%	0,00%	15,86%
15	567	0,35%	0,00%	5,47%	3,88%	2,47%
Average Gap		**2,02%**	**5,28%**	**4,84%**	**2,97%**	**2,83%**

Table A.50: Gap to the best known solution for the *first* random test for anti-collision mechanism *Path isolation* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	82	82	82	**82**	**82**	**82**	**82**	**82**	**82**	**82**
2	132	132	132	**121**	**121**	**121**	**121**	**121**	**121**	**121**
3	166	196	196	147	147	147	147	147	**139**	**139**
4	182	230	230	**182**	**182**	**182**	**182**	**182**	186	186
5	187	239	239	**181**	**181**	**181**	**181**	**181**	236	203
6	253	326	326	219	237	**218**	253	219	302	302
7	274	-	-	244	**238**	261	241	246	-	-
8	319	-	-	282	293	281	275	**267**	-	-
9	294	-	-	**294**	**294**	**294**	**294**	**294**	-	-
10	394	-	-	**337**	364	364	353	355	-	-
11	415	-	-	**376**	415	382	382	380	-	-
12	488	-	-	454	431	465	484	**429**	-	-
13	503	-	-	463	512	518	**452**	489	-	-
14	537	-	-	**465**	508	517	467	536	-	-
15	598	-	-	569	**567**	598	589	581	-	-

Table A.51: Makespan results for the *second* random test for anti-collision mechanism *Path isolation* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	82	0,00%	0,00%	0,00%	0,00%	0,00%
2	121	0,00%	0,00%	0,00%	0,00%	0,00%
3	139	5,76%	5,76%	5,76%	5,76%	5,76%
4	182	0,00%	0,00%	0,00%	0,00%	0,00%
5	181	0,00%	0,00%	0,00%	0,00%	0,00%
6	218	0,46%	8,72%	0,00%	16,06%	0,46%
7	238	2,52%	0,00%	9,66%	1,26%	3,36%
8	267	5,62%	9,74%	5,24%	3,00%	0,00%
9	294	0,00%	0,00%	0,00%	0,00%	0,00%
10	337	0,00%	8,01%	8,01%	4,75%	5,34%
11	376	0,00%	10,37%	1,60%	1,60%	1,06%
12	429	5,83%	0,47%	8,39%	12,82%	0,00%
13	452	2,43%	13,27%	14,60%	0,00%	8,19%
14	465	0,00%	9,25%	11,18%	0,43%	15,27%
15	567	0,35%	0,00%	5,47%	3,88%	2,47%
Average Gap		**1,53%**	**4,37%**	**4,66%**	**3,30%**	**2,79%**

Table A.52: Gap to the best known solution for the *second* random test for anti-collision mechanism *Path isolation* on *large* field problem instances.

	First Feasible Sol			Best Solution						
$\|\mathcal{R}eq\|$	**L2**	**Full 5m**	**Full 30m**	**S**	**C1**	**C2**	**C3**	**C4**	**Full 5m**	**Full 30m**
1	82	82	82	**82**	**82**	**82**	**82**	**82**	**82**	**82**
2	132	132	132	**121**	**121**	**121**	**121**	**121**	**121**	**121**
3	166	196	196	147	147	147	147	147	**139**	**139**
4	182	230	230	**182**	**182**	**182**	**182**	**182**	186	186
5	187	239	239	**181**	**181**	**181**	**181**	**181**	236	203
6	253	326	326	219	237	**218**	253	219	302	302
7	274	-	-	244	**238**	261	241	246	-	-
8	319	-	-	282	301	281	275	**267**	-	-
9	294	-	-	297	**294**	**294**	**294**	**294**	-	-
10	394	-	-	**337**	364	364	353	355	-	-
11	415	-	-	**376**	415	382	382	380	-	-
12	488	-	-	454	431	465	484	**429**	-	-
13	503	-	-	463	512	501	**452**	489	-	-
14	537	-	-	**465**	506	507	471	536	-	-
15	598	-	-	569	**567**	598	589	581	-	-

Table A.53: Makespan results for the *third* random test for anti-collision mechanism *Path isolation* on *large* field problem instances.

		Gap to the Best Known Solution				
$\|\mathcal{R}eq\|$	**Best Solution**	**S**	**C1**	**C2**	**C3**	**C4**
1	82	0,00%	0,00%	0,00%	0,00%	0,00%
2	121	0,00%	0,00%	0,00%	0,00%	0,00%
3	139	5,76%	5,76%	5,76%	5,76%	5,76%
4	182	0,00%	0,00%	0,00%	0,00%	0,00%
5	181	0,00%	0,00%	0,00%	0,00%	0,00%
6	218	0,46%	8,72%	0,00%	16,06%	0,46%
7	238	2,52%	0,00%	9,66%	1,26%	3,36%
8	267	5,62%	12,73%	5,24%	3,00%	0,00%
9	294	1,02%	0,00%	0,00%	0,00%	0,00%
10	337	0,00%	8,01%	8,01%	4,75%	5,34%
11	376	0,00%	10,37%	1,60%	1,60%	1,06%
12	429	5,83%	0,47%	8,39%	12,82%	0,00%
13	452	2,43%	13,27%	10,84%	0,00%	8,19%
14	465	0,00%	8,82%	9,03%	1,29%	15,27%
15	567	0,35%	0,00%	5,47%	3,88%	2,47%
Average Gap		**1,60%**	**4,54%**	**4,27%**	**3,36%**	**2,79%**

Table A.54: Gap to the best known solution for the *third* random test for anti-collision mechanism *Path isolation* on *large* field problem instances.

Bibliography

[1] Alford, R., Behnke, G., Höller, D., Bercher, P., Biundo-Stephan, S., Aha, D.W.: Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive htn problems. In: International Conference on Automated Planning and Scheduling (ICAPS) (2016)

[2] Alford, R., Shivashankar, V., Roberts, M., Frank, J., Aha, D.W.: Hierarchical planning: Relating task and goal decomposition with task sharing. In: International Joint Conference on Artificial Intelligence, (IJCAI). pp. 3022–3029 (2016)

[3] Alford, R., Kuter, U., Nau, D.: Translating htns to pddl: A small amount of domain knowledge can go a long way. In: International Joint Conference on Artificial Intelligence, (IJCAI). pp. 1629–1634 (2009)

[4] Amadini, R., Gabbrielli, M., Mauro, J.: An extensive evaluation of portfolio approaches for constraint satisfaction problems. International Journal of Interactive Multimedia and Artificial Intelligence **3**(7), 81–86 (2016)

[5] Amadini, R., Gabbrielli, M., Mauro, J.: SUNNY-CP and the Minizinc challenge. Theory and Practice of Logic Programming (2017)

[6] Amaran, S., Sahinidis, N., Sharda, B., Bury, S.: Simulation optimization: A review of algorithms and applications (2017)

[7] Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence **116**(1), 123 – 191 (2000)

[8] Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems, International Series in Operations Research and Management Science, vol. 39. Kluwer Academic Publishers (2001)

[9] Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., Smith, D.: EUROPA: A platform for AI planning, scheduling, constraint programming, and optimization. In: Proceedings of the 28th International Conference on Automated Planning and Scheduling (2012)

[10] Barrett, A., Weld, D.S.: Task-decomposition via plan parsing. In: Proceedings of the Twelfth National Conference on Artificial Intelligence. p. 1117–1122. AAAI'94, American Association for Artificial Intelligence, USA (1994)

[11] Barzanji, R., Naderi, B., Begen, M.A.: Decomposition algorithms for the integrated process planning and scheduling problem. Omega - The International Journal of Management Science **93**, 102025 (2020)

[12] Bassett, M.H., Pekny, J.F., Reklaitis, G.V.: Decomposition techniques for the solution of large-scale scheduling problems. AIChE Journal pp. 3373–3387 (1996)

[13] Bechon, P., Barbier, M., Infantes, G., Lesire, C., Vidal, V.: HiPOP: Hierarchical partial-order planning. In: Proceedings of the 7th European Starting AI Researcher Symposium (2014)

[14] Behnke, G., Höller, D., Bercher, P., Biundo, S., Pellier, D., Fiorino, H., Alford, R.: Hierarchical planning in the IPC (2019)

[15] Behnke, G., Höller, D., Biundo-Stephan, S.: totSAT - totally-ordered hierarchical planning through SAT. In: AAAI Conference on Artificial Intelligence (AAAI). p. 6110–6118. AAAI Press (2018)

[16] Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik **4**(1), 238–252 (1962)

[17] Benini, L., Bertozzi, D., Guerri, A., Milano, M.: Allocation and scheduling for MPSoCs via decomposition and no-good generation. pp. 107–121 (2005)

[18] Bercher, P., Alford, R., Höller, D.: A survey on hierarchical planning – one abstract idea, many concrete realizations. pp. 6267–6275 (2019)

[19] Bercher, P., Behnke, G., Höller, D., Biundo, S.: An admissible HTN planning heuristic. In: International Joint Conference on Artificial Intelligence, (IJCAI). pp. 480–488 (2017)

[20] Bercher, P., Höller, D., Behnke, G., Biundo-Stephan, S.: More than a name? on implications of preconditions and effects of compound HTN planning tasks. In: European Conference on Artificial Intelligence (ECAI). pp. 225–233 (2016)

[21] Bit-Monnot, A.: Temporal and Hierarchical Models for Planning and Acting in Robotics. Thesis (2016), thèse de doctorat dirigée par Malik Ghallab et Félix Ingrand. Robotique et Informatique. Toulouse, INPT.

[22] Bockmayr, A., Kasper, T.: Branch-and-Infer: a Framework for Combining CP and IP, pp. 59–87 (2003)

[23] Botea, A., Enzenberger, M., Müller, M., Schaeffer, J.: Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. Journal of Artificial Intelligence Research (JAIR) **24**, 581–621 (2005)

[24] Bowman, E.H.: The schedule-sequencing problem. Operations Research **7**(5), 621–624 (1959)

[25] Brucker, P.: Scheduling Algorithms. SpringerVerlag, 5th edn. (2004)

[26] Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research **112**(1), 3 – 41 (1999)

[27] Cambazard, H., Hladik, P.E., Déplanche, A.M., Jussien, N., Trinquet, Y.: Decomposition and learning for a hard real time task allocation problem. pp. 153–167. Springer Berlin Heidelberg (2004)

[28] Cambazard, H., Jussien, N.: Benders decomposition in constraint programming. Joint Annual Workshop of ERCIM/CoLogNet on Constraint Solving and Constraint Logic Programming (CSCLP) (2005)

[29] Castellanos-Paez, S., Pellier, D., Fiorino, H., Pesty, S.: Mining useful macro-actions in planning. In: Third International Conference on Artificial Intelligence and Pattern Recognition (AIPR). pp. 1–6 (2016)

[30] Chaimowicz, L., Cowley, A., Gomez-Ibanez, D., Grocholsky, B., A. Hsieh, M., Hsu, H., F. Keller, J., Kumar, V., Swaminathan, R., J. Taylor, C.: Deploying air-ground multi-robot teams in urban environments. In: Multi-Robot Systems. From Swarms to Intelligent Automata Volume III. pp. 223–234 (2005)

[31] Chen, Z., Powell, W.B.: Solving parallel machine scheduling problems by column generation. Informs Journal on Computing **11**, 78–94 (1999)

[32] Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., Tran, D.: ASPEN - automated planning and scheduling for space mission operations. International Conference on Space Operations (SpaceOps) (2000)

[33] Ciré, A.A., Çoban, E., Hooker, J.N.: Logic-based Benders decomposition for planning and scheduling: a computational analysis. The Knowledge Engineering Review **31**(5), 440–451 (2016)

[34] Dantzig, G.B.: Linear Programming and Extensions. Princeton University Press (1991)

[35] Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. Operations Research **8**(1), 101–111 (1960)

[36] Dean, T., Wellman, M.: Planning and Control. Morgan Kaufmann series in Representation and Reasoning, Morgan Kaufmann Publishers (1991)

[37] Dechter, R., Dechter, A.: Belief maintenance in dynamic constraint networks. pp. 37–42 (1988)

[38] Dechter, R., Pearl, J.: Tree clustering for constraint networks. Artificial Intelligence **38**(3), 353 – 366 (1989)

[39] Dix, J., Kuter, U., Nau, D.: Planning in answer set programming using ordered task decomposition. In: Annual German Conference on AI. pp. 490–504. Springer (2003)

[40] Dvořák, F., Barták, R., Bit-Monnot, A., Ingrand, F., Ghallab, M.: Planning and acting with temporal and hierarchical decomposition models. In: 26th IEEE International Conference on Tools with Artificial Intelligence. pp. 115–121 (2014)

[41] Eremin, A., Wallace, M.: Hybrid Benders decomposition algorithms in constraint logic programming. In: International Conference on Principles and Practice of Constraint Programming (CP). vol. 2239, pp. 1–15 (2001)

[42] Erol, K., Hendler, J., Nau, D.S.: HTN planning: Complexity and expressivity. In: Proceedings of the 12th National Conference on Artificial Intelligence. pp. 1123–1128. AAAI Press (1994)

[43] Erol, K., Hendler, J.A., Nau, D.S.: Semantics for Hierarchical Task Network planning (1994)

[44] Fikes, R.E., Nilsson, N.J.: Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2**(3), 189 – 208 (1971)

[45] Fleming, Q., Koppelman, J.: Earned Value Project Management. Project Management Institute, 4th edn. (2016)

[46] Ford, L.R., Fulkerson, D.R.: A suggested computation for maximal multi-commodity network flows (1964)

[47] Fox, M., Long, D.: Pddl2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research **20**, 61–124 (2003)

[48] Fukunaga, A., Rabideau, G., Chien, S., Yan, D.: ASPEN: A framework for automated planning and scheduling of spacecraft control and operations. Int. Symposium on AI, Robotics and Automation in Space (ISAIRAS) (1997)

[49] Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1st edn. (1978)

[50] Geier, T., Bercher, P.: On the decidability of HTN planning with task insertion. pp. 1955–1961 (2011)

[51] Geoffrion, A.: Generalized Benders decomposition. Journal of Optimization Theory and Applications **10**, 237–260 (1972)

[52] Ghallab, M., et al.: PDDL - The Planning Domain Definition Language (1998)

[53] Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., 1st edn. (2004)

[54] Ghallab, M., Nau, D., Traverso, P.: Automated Planning and Acting. Cambridge University Press, 1st edn. (2016)

[55] Girbal, S., Jean, X., Rhun, J.L., Pérez, D.G., Gatti, M.: Deterministic platform software for hard real-time systems using multi-core cots. In: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC) (2015)

[56] González-Ferrer, A., Fernández-Olivares, J., Castillo, L.: JABBAH: A java application framework for the translation between business process models and HTN. pp. 28–37 (2009)

[57] Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence **124**(2), 243 – 282 (2000)

[58] Graham, R., Lawler, E., Lenstra, J., Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a Survey. In: Discrete Optimization II, Annals of Discrete Mathematics, vol. 5, pp. 287 – 326. Elsevier (1979)

[59] Hao, J.H., Liu, M., hua Lin, J., Wu, C.: A hybrid differential evolution approach based on surrogate modelling for scheduling bottleneck stages. Computers & Operations Research **66**, 215 – 224 (2016)

[60] Harjunkoski, I., Grossmann, I.: Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. Computers and Chemical Engineering **26**, 1533–1552 (2002)

[61] Haugan, G.T.: Effective Work Breakdown Structures. Management Concepts (2001)

[62] Heinz, S., Beck, J.: Reconsidering mixed integer programming and MIP-based hybrids for scheduling. pp. 211–227 (05 2012)

[63] Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., Alford, R.: HDDL – a language to describe Hierarchical Planning Problems. In: 2nd ICAPS Workshop on Hierarchical Planning. pp. 6–14 (2019)

[64] Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., Alford, R.: HDDL: An extension to PDDL for expressing Hierarchical Planning Problems. In: 34th AAAI Conference on Artificial Intelligence (AAAI) (2020)

[65] Höller, D., Bercher, P., Behnke, G., Biundo, S.: On guiding search in HTN planning with classical planning heuristics. In: International Joint Conference on Artificial Intelligence, (IJCAI). pp. 6171–6175 (2019)

[66] Höller, D., Bercher, P., Behnke, G., Biundo-Stephan, S.: A generic method to guide HTN progression search with classical heuristics. In: International Conference on Automated Planning and Scheduling (ICAPS). pp. 114–122 (2018)

[67] Hooker, J.N.: A search-infer-and-relax framework for integrating solution methods. In: Barták, R., Milano, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 243–257. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

[68] Hooker, J., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming **96**(1), 33–60 (2003)

[69] Hooker, J.: A hybrid method for planning and scheduling. pp. 305–316 (2004)

[70] Hooker, J.: Planning and scheduling by logic-based Benders decomposition. Operations Research **55**, 588–602 (2007)

[71] Hooker, J.N.: Logic-based Benders decomposition for large-scale optimization. Large Scale Optimization in Supply Chains and Smart Manufacturing p. 1–26 (2019)

[72] Jain, V., Grossmann, I.E.: Algorithms for hybrid MILP/CP models for a class of optimization problems. INFORMS Journal on Computing **13**, 258–276 (2001)

[73] Jegou, P., Ndiaye, S., Terrioux, C.: Combined strategies for decomposition-based methods for solving csps. pp. 184–192 (2009)

[74] Ji, Y., Kim, S., Xu, W.L.: A new framework for combining global and local methods in black box optimization (2013)

[75] Junker, U., Karisch, S., Kohl, N., Vaaben, B., Fahle, T., Sellmann, M.: A framework for constraint programming based column generation. In: International Conference on Principles and Practice of Constraint Programming. vol. 1713, pp. 261–274 (1999)

[76] Khac Vu, K., D'Ambrosio, C., Hamadi, Y., Liberti, L.: Surrogate-based methods for black-box optimization. International Transactions in Operational Research **24**, 393–424 (2016)

[77] Knoblock, C.A.: An analysis of abstrips. In: Hendler, J. (ed.) Artificial Intelligence Planning Systems, pp. 126 – 135. Morgan Kaufmann, San Francisco (CA) (1992)

[78] Knoblock, C.A.: Automatically generating abstractions for planning. Artificial Intelligence **68**(2), 243 – 302 (1994)

[79] Koes, M., R. Nourbakhsh, I., Sycara, K.: Heterogeneous multirobot coordination with spatial and temporal constraints. In: International Conference on Artificial Intelligence (AAAI). pp. 1292–1297 (2005)

[80] Kolisch, R., Hartmann, S.: Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, pp. 147–178. Springer US, Boston, MA (1999)

[81] Kondili, E., Pantelides, C., Sargent, R.: A general algorithm for short-term scheduling of batch operations — i. milp formulation. Computers & Chemical Engineering **17**(2), 211 – 227 (1993), an International Journal of Computer Applications in Chemical Enginnering

[82] Ku, W.Y., Beck, J.C.: Mixed integer programming models for job shop scheduling: A computational analysis. Computers & Operations Research **73**, 165 – 173 (2016)

[83] Laborie, P., Rogerie, J.: Reasoning with conditional time-intervals. In: FLAIRS Conference (2008)

[84] Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: Reasoning with conditional time-intervals. Part II: An algebraical model for resources. In: FLAIRS Conference (2009)

[85] Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP Optimizer for scheduling. Constraints **23**(2), 210–250 (2018)

[86] Laborie, P., Rogerie, J., Shaw, P., Vilím, P., Katai, F.: Interval-Based Language for Modeling Scheduling Problems: An Extension to Constraint Programming, pp. 111–143. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[87] Lallement, R., de Silva, L., Alami, R.: Hatp: Hierarchical agent-based task planner. In: International Conference on Autonomous Agents and MultiAgent Systems. p. 1823–1825. International Foundation for Autonomous Agents and Multiagent Systems (AAMAS), Richland, SC (2018)

[88] Lawler, E., Lenstra, J., Rinnooy Kan, A., Shmoys, D.: Sequencing and scheduling: algorithms and complexity, pp. 445–522. Handbooks in Operations Research and Management Science, North-Holland Publishing Company (1993)

[89] Lenstra, J., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. In: Hammer, P., Johnson, E., Korte, B., Nemhauser, G. (eds.) Studies in Integer Programming, Annals of Discrete Mathematics, vol. 1, pp. 343 – 362. Elsevier (1977)

[90] Li, Z., Ierapetritou, M.: Integrated production planning and scheduling using a decomposition framework. Chemical Engineering Science **64**, 3585–3597 (2009)

[91] Lourenço, H., Martin, O., Stützle, T.: Iterated Local Search: Framework and Applications, vol. 146, pp. 363–397 (2010)

[92] Maillard, A., Verfaillie, G., Pralet, C., Jaubert, J., Desmousceaux, T.: Building Flexible Download Plans for Agile Earth-Observing Satellites. In: 12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS) (2014)

[93] Maillard, A.: Flexible Scheduling for Agile Earth Observing Satellites. Thesis (2015), thèse de doctorat dirigée par Gérard Verfaillie et Cédric Pralet. Systèmes embarqués et Robotique Toulouse, ISAE.

[94] Manne, A.S.: On the job-shop scheduling problem. Operations Research **8**(2), 219–223 (1960)

[95] Mansouri, M., Pecora, F.: More knowledge on the table: Planning with space, time and resources for robots. In: IEEE International Conference on Robotics and Automation (ICRA). pp. 647–654 (2014)

[96] Mansouri, M., Pecora, F.: A robot sets a table: a case for hybrid reasoning with different types of knowledge. Journal of Experimental & Theoretical Artificial Intelligence **28**(5), 801–821 (2016)

[97] Mas, F., Ríos, J., Menéndez, J., Gomez, A.: A process oriented approach to modelling the conceptual design of aircraft assembly line. International Journal of Advanced Manufacturing Technology **67**, 771–784 (2013)

[98] Milano, M., Van Hentenryck, P. (eds.): Hybrid Optimization. The ten years of CPAIOR, Springer-Verlag New York (2011)

[99] Mittal, S., Falkenhainer, B.: Dynamic Constraint Satisfaction Problems. In: Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 1. p. 25–32. AAAI'90, AAAI Press (1990)

[100] Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An htn planning system. Journal of Artificial Intelligence Research **20**, 379–404 (2003)

[101] Nau, D.S., Muñoz-Avila, H., Cao, Y., Lotem, A., Mitchell, S.: Total-order planning with partially ordered subtasks. In: International Joint Conferences on Artificial Intelligence (IJCAI). p. 425–430 (2001)

[102] Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Wu, D., Yaman, F., Muñoz-Ávila, H., Murdock, J.W.: Applications of SHOP and SHOP2. IEEE Intelligent Systems **20**(2), 34–41 (2005)

[103] Nesterov, Y., Nemirovskii, A.: Interior-point Polynomial Algorithms in Convex Programming. Studies in Applied Mathematics, Society for Industrial and Applied Mathematics (1994)

[104] Oddi, A., Rasconi, R., Cesta, A., Smith, S.F.: Applying iterative flattening search to the job shop scheduling problem with alternative resources and sequence dependent setup times. In: Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems. pp. 15–22 (2011)

[105] O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: Irish Conference on Artificial Intelligence and Cognitive Science (2008)

[106] Pacheco, A., Pralet, C., Roussel, S.: Techniques de décisions hiérarchiques pour l'ordonnancement de tâches. In: Journées Francophones de Programmation par Contraintes (JFPC) (2018)

[107] Pacheco, A., Pralet, C., Roussel, S.: Constraint-Based Scheduling with Complex Setup Operations: An Iterative Two-Layer Approach. In: International Joint Conference on Artificial Intelligence (IJCAI) (2019)

[108] Pacheco, A., Pralet, C., Roussel, S.: Decomposition and Cut Generation Strategies for Solving Multi-Robot Deployment Problems. In: International Conference on Principles and Practice of Constraint Programming (CP) (2019)

[109] Pacheco, A., Pralet, C., Roussel, S.: Hierarchical scheduling for multi-robot missions. In: Congrès Annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF) (2019)

[110] Pacheco, A., Pralet, C., Roussel, S.: Ordonnancement avec opérations de préparation complexes : programmation par contraintes et optimisation hiérarchique. In: Congrès Annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF) (2020)

[111] Perret, Q., Maurère, P., Noulard, E., Pagetti, C., Sainrat, P., Triquet, B.: Mapping hard real-time applications on many-core processors. In: Proceedings of the 24th International Conference on Real-Time Networks and Systems. pp. 235–244. ACM (2016)

[112] Perret, Q., Maurère, P., Noulard, E., Pagetti, C., Sainrat, P., Triquet, B.: Temporal isolation of hard real-time applications on many-core processors. In: RTAS : Real-Time Embedded Technology & Applications Symposium (2016)

[113] Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer Publishing Company, 5th edn. (2016)

[114] Pralet, C., Roussel, S., Polacsek, T., Bouissière, F., Cuiller, C., Dereux, P.E., Kersuzan, S., Lelay, M.: A scheduling tool for bridging the gap between aircraft design and aircraft manufacturing. In: International Conference on Automated Planning and Scheduling (ICAPS) (2018)

[115] Pralet, C., Lesire, C.: Deployment of mobile wireless sensor networks for crisis management: A constraint-based local search approach. pp. 870–885 (2014)

[116] Pralet, C., Solange, L.C., Jaubert, J.: Scheduling running modes of satellite instruments using constraint-based local search. pp. 704–719 (2015)

[117] Puchinger, J., Stuckey, P., Wallace, M., Brand, S.: Dantzig-Wolfe decomposition and branch-and-price solving in g12. Constraints **16**, 77–99 (2011)

[118] Qi, C., Wang, D., Muñoz-Ávila, H., Zhao, P., Wang, H.: Hierarchical Task Network Planning with resources and temporal constraints. Knowledge-Based Systems **133**, 17–32 (2017)

[119] Quilliot, A., Toussaint, H.: Flow models for project scheduling with transfer delays. 2012 Federated Conference on Computer Science and Information Systems (FedCSIS) pp. 439–446 (2012)

[120] Ralphs, T., Galati, M.: Decomposition and dynamic cut generation in integer linear programming. Mathematical Programming **106**, 261–285 (2006)

[121] Ramoul, A., Pellier, D., Fiorino, H., Pesty, S.: Grounding of HTN planning domain. International Journal on Artificial Intelligence Tools **26** (2017)

[122] Ren, Q., Man, K.L., Lim, E.G., Lee, J., Kim, K.K.: Cooperation of multi robots for disaster rescue. In: 2017 International SoC Design Conference (ISOCC). pp. 133–134 (2017)

[123] Robertson, N., Seymour, P.: Graph minors. ii. algorithmic aspects of tree-width. Journal of Algorithms **7**(3), 309 – 322 (1986)

[124] Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2. Elsevier (2006)

[125] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, 3rd edn. (2010)

[126] Sacerdoti, E.D.: Planning in a hierarchy of abstraction spaces. In: Artificial Intelligence. pp. 115–135 (1974)

[127] Sadykov, R., Wolsey, L.: Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. Informs Journal on Computing **18**, 209–217 (2006)

[128] Schattenberg, B., Biundo, S.: On the identification and use of hierarchical resources in planning and scheduling. In: Proceedings of the International Conference on Artificial Intelligence Planning Systems (AIPS). pp. 263–273 (2002)

[129] Schiex, T., Verfaillie, G.: Nogood recording for static and dynamic Constraint Satisfaction Problems. pp. 48–55 (1993)

[130] Schreiber, D., Pellier, D., Fiorino, H., Balyo, T.: Tree-rex: SAT-based tree exploration for efficient and high-quality htn planning. In: International Conference on Automated Planning and Scheduling (ICAPS) (2019)

[131] Sette, F., Vaquero, T., Park, S.W., Silva, J.: Are automated planners up to solve real problems. Proceedings of the 17th World Congress The International Federation of Automatic Control (IFAC) **17** (2008)

[132] Shivashankar, V., Kuter, U., Nau, D., Alford, R.: A hierarchical goal-based formalism and algorithm for single-agent planning. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). p. 981–988. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2012)

[133] d. Silva, L., Lallement, R., Alami, R.: The HATP hierarchical planner: Formalisation and an initial study of its usability and practicality. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 6465–6472 (2015)

[134] Smith, D., Frank, J., Jónsson, A.: Bridging the gap between planning and scheduling. The Knowledge Engineering Review **15** (2000)

[135] Smith, D.E., Frank, J., Cushing, W.: The ANML Language. In: The ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) (2008)

[136] Sohrabi, S., Baier, J., Mcilraith, S.: HTN planning with preferences. pp. 1790–1797 (2009)

[137] Stankovic, J.A., Ramamritham, K., Spuri, M.: Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms. Kluwer Academic Publishers, USA (1998)

[138] Stock, S., Mansouri, M., Pecora, F., Hertzberg, J.: Online task merging with a hierarchical hybrid task planner for mobile service robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 6459–6464 (2015)

[139] Sugiyama, H., Tsujioka, T., Murata, M.: Real-time exploration of a multi-robot rescue system in disaster areas. Advanced Robotics **27** (2013)

[140] Surynek, P., Barták, R.: Encoding htn planning as a dynamic CSP. vol. 3709, p. 868 (10 2005)

[141] Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operations Research **64**, 278–285 (1993)

[142] Thorsteinsson, E.: Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. vol. 2239, p. 16–30 (2001)

[143] Tran, T.T., Vaquero, T.S., Nejat, G., Beck, J.C.: Robots in retirement homes: Applying off-the-shelf planning and scheduling to a team of assistive robots. Journal of Artificial Intelligence Research **58**, 523–590 (2017)

[144] Umbrico, A., Cesta, A., Mayer, M.C., Orlandini, A.: PLATINUm: A New Framework for Planning and Acting. In: 2017 Advances in Artificial Intelligence (AI*IA). "Springer International Publishing (2017)

[145] Umbrico, A., Cesta, A., Mayer, M.C., Orlandini, A.: Integrating resource management and timeline-based planning. In: Proceedings of the 28th International Conference on Automated Planning and Scheduling. pp. 264–272 (2018)

[146] Vanderbeck, F.: On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. Operations Research **48**, 111–128 (2000)

[147] Vanderbeck, F., Wolsey, L.: Reformulation and Decomposition of Integer Programs, pp. 431–502 (2010)

[148] Verfaillie, G., Jussien, N.: Constraint solving in uncertain and dynamic environments: A survey. Constraints **10** (2005)

[149] Verfaillie, G., Schiex, T.: Solution reuse in Dynamic Constraint Satisfaction Problems. vol. 1, pp. 307–312 (1994)

[150] Viana, F., Haftka, R., Watson, L.: Efficient global optimization algorithm assisted by multiple surrogate techniques. Journal of Global Optimization **56**, 669–689 (2013)

[151] Vinot, M.: Résolution conjointe de problèmes d'ordonnancement et de routage. Thesis (2017), thèse de doctorat dirigée par Philippe Lacomme, Aziz Moukirm et Alain Quilliot. Informatique. Université Clermont Auvergne.

[152] Wagner, H.M.: An Integer Linear-Programming model for machine scheduling. Naval Research Logistics Quarterly **6**(2), 131–140 (1959)

[153] Wallace, R., Grimes, D., Freuder, E.: Dynamic Constraint Satisfaction Problems: Relations among search strategies, solution sets and algorithm performance. vol. 6384, pp. 105–121 (06 2009)

[154] Wilhelm, R., Reineke, J.: Embedded systems: Many cores - many problems. In: 7th IEEE International Symposium on Industrial Embedded Systems. pp. 176–180 (2012)

[155] Wilkins, D.E.: Practical Planning: Extending the Classical AI Planning Paradigm. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1988)

[156] Williamson, D.P., Shmoys, D.B.: The Design of Approximation Algorithms. Cambridge University Press (2011)

Résumé — Dans de nombreuses applications qui présentent un problème de décision ou d'optimisation combinatoire, il est utile de raisonner à différents niveaux d'abstraction. C'est tout d'abord le cas pour des missions d'exploration multi-robots, où l'on peut s'intéresser premièrement à la répartition de tâches d'exploration entre différents robots, puis à la manière dont chaque robot enchaîne les tâches qui lui sont allouées, et enfin à la décomposition de ces enchaînements de tâches sous la forme de déplacements à coordonner pour éviter des collisions ou pour maintenir des liens de communication. C'est aussi le cas pour la gestion d'une constellation de satellites d'observation de la Terre, pour lesquels on peut décider dans un premier temps de la répartition des tâches d'acquisition candidates entre les différents satellites, puis de l'enchaînement de ces acquisitions pour chaque satellite de la constellation, et enfin des commandes élémentaires à envoyer aux instruments pour réaliser effectivement cet enchaînement. C'est encore le cas pour l'implémentation de fonctions sur une architecture avionique, avec en premier lieu une décision concernant l'allocation de fonctions sur des unités de calcul temps réel, puis une décision concernant l'ordonnancement des fonctions sur chaque unité de calcul, et enfin une décision sur la stratégie de routage des données échangées entre fonctions sur un réseau disponible. D'un point de vue général, il est ainsi nécessaire dans ce type d'applications de considérer différents niveaux de décision couvrant allocation des tâches sur des ressources et ordonnancement des tâches sur ces mêmes ressources. Chaque tâche à considérer peut de plus se décomposer en plusieurs sous-tâches, dans le sens par exemple où une tâche de calcul d'une fonction correspond à l'enchaînement d'une tâche de lecture des données utilisées par la fonction, d'une tâche de calcul proprement dite, et d'une tâche d'écriture des sorties de la fonction dans une zone mémoire donnée. En plus de cela, les contraintes des problèmes de décision à résoudre peuvent être représentées avec différents niveaux d'abstraction. Par exemple, en exploration multi-robots, il existe des contraintes portant sur l'énergie disponible pour les robots. Au moment de la répartition des tâches d'exploration entre les robots, il n'est pas forcément possible pour des considérations combinatoires de considérer un modèle dynamique complexe reliant l'énergie disponible à la puissance consommée à chaque instant. On considère alors une consommation d'énergie forfaitaire pour chaque activité et une capacité maximale pour chaque robot. Le modèle d'énergie complexe peut être pris en compte dès lors que les tâches ont été réparties et que l'on synthétise les déplacements des robots. De manière analogue, pour l'agencement des observations d'un satellite, on peut considérer en première approximation qu'il existe une durée forfaitaire requise pour passer d'une observation à la suivante, avant de considérer des modèles cinématiques plus complexes prenant en compte les capacités des actionneurs gyroscopiques et les caractéristiques des zones à imager. Cette thèse s'intéresse à la définition de modèles et d'algorithmes de décision utilisables pour gérer ces problématiques de décision hiérarchique.

Mots-clés : ordonnancement hiérarchique, optimisation combinatoire, décomposition du problème, programmation par contraintes.

ONERA / DTIS, Université de Toulouse
F-31055 Toulouse, France

Abstract — In many decision making problems encountered in practice, there is a need to reason at different abstraction levels. This is the case for multi-robot exploration missions, which involve decisions on how exploration tasks are shared between robots, on how these tasks are successively realized by each robot, and on how robots moves are coordinated so as to avoid collisions or to maintain communication links. This is also the case for the management of a constellation of Earth observation satellites, for which decisions must be made concerning how candidate observation tasks are shared between satellites, concerning how each satellite realizes the set of observation tasks allocated to it, and concerning the basic commands that must be sent to instruments for achieving the acquisition plan. This is again the case for the mapping of functions onto an embedded architecture, with first a decision concerning the allocation of functions onto real-time computation units, then a decision concerning the scheduling of functions on each unit, and last a decision on the routing of data exchanged between functions inside an available inter-units network. From a general point of view, such applications require being able to deal with different decision levels covering task allocation and task scheduling over ressources. Additionnally, each task involved must also sometimes be decomposed into subtasks, as in the case of a function computation task which is decomposed into one subtask for reading the set of data used by the function, one subtask for actually performing the computation, and one subtask for writing the result of the computation in a dedicated buffer. Additionnally, the constraints of the decision making problem to solve can be modeled using different abstraction levels. For instance, in multi-robot exploration, there exist constraints on the energy available for each robot all along the mission. At the time of task sharing between robots, it is not possible for combinatorial reasons to consider a complex dynamic model linking the level of energy available to the instantaneous power consumption. Instead, a simpler model might be used, in which a fixed amount of energy is consumed by each task. The complex energy model can be considered again once tasks have been allocated to robots and once robot moves have been synthesized. Similarly, for Earth observing satellites, it is possible to consider a fixed duration between the realization of two successive activities before considering finer a model taking into account the kinematics of satellites and the coordinates of images to be realized. The goal of this thesis is to define new models and algorithms for handling such hierarchical decision making problems.

Keywords: hierarchical scheduling, combinatorial optimization, problem decomposition constraint programming.

ONERA / DTIS, Université de Toulouse
F-31055 Toulouse, France