Parameterized complexity and new efficient enumerative schemes for RCPSP

Maher Mallem



SORBONNE UNIVERSITÉ

EDITE DE PARIS (ED130)
INFORMATIQUE, TELECOMUNICATIONS ET ELECTRONIQUE

Complexité Paramétrée et Nouveaux Schémas Enumeratifs Efficaces pour le RCPSP

Parameterized Complexity and New Efficient Enumerative Schemes for RCPSP

Thèse de doctorat par

Maher MALLEM

Spécialité: INFORMATIQUE

Thèse dirigée par Claire HANEN

Préparée au LIP6 (UMR 7606 Sorbonne Université - CNRS)

Présentée et soutenue publiquement le 11 Octobre 2024

Directrice:	Claire HANEN	-	Professeure (LIP6 - Université Paris Nanterre)
Rapporteurs:	Nadia BRAUNER	-	Professeure (Université Grenoble Alpes)
	Imed KACEM	-	Professeur (Université de Lorraine, Metz)
Examinateurs:	Hans L. BODLAENDER	-	Professeur (Utrecht University)
	Bruno ESCOFFIER	-	Professeur (Sorbonne Université, Paris)
	Michael LAMPIS	-	Maître de conférences (Université Paris Dauphine)
Invitée:	Alix MUNIER KORDON	-	Professeure (Sorbonne Université, Paris)

Acknowledgments - Remerciements

First, I would like to thank my reviewers Nadia Brauner and Imed Kacem for their careful reading and their insightful comments on my manuscript. I am very grateful for the time and care you took in reviewing my thesis. I would also like to thank my examiners Bruno Escoffier, Hans L. Bodlaender and Michael Lampis for accepting to be part of the committee. I feel honored to have spent a privileged moment with all of you on the day of my defense.

Je souhaite maintenant remercier ma directrice de thèse Claire et sa collègue Alix pour leur soutien continu et inconditionnel. Malgré vos emplois du temps chargés, nous avons pu se retrouver à trois très régulièrement pour discuter de ma thèse et la faire avancer. Vous m'avez fait confiance tout au long de ces trois années, discutant de chacune de mes idées avec écoute, sans filtre et sur un pied d'égalité. Surtout, vous m'avez donné ma chance durant une période délicate de ma vie, où j'étais en année de césure non planifiée. Je suis plus qu'heureux d'avoir porté ce sujet de recherche, et j'aurai à cœur de poursuivre ma recherche dans le domaine de la recherche opérationnelle et de l'ordonnancement, notamment à vos côtés.

Un grand merci à mes collègues du LIP6, avec qui j'ai pu partager des moments privilégiés. Je pense particulièrement à Bruno, Carola, Christoph, Emmanuel, Evripidis, Fanny et Valentin pour leur accueil chaleureux dans l'équipe RO et les moments dédiés à l'équipe, notamment les deux journées à Trouville en juin 2024. Merci également aux doctorants de l'équipe Martin, François et Georgii pour leur organisation de séminaires réguliers. Une mention spéciale à mon collègue de bureau Mohamed, doctorant de l'équipe DECISION, pour les nombreuses discussions enrichissantes et les partages de galères personnelles !

Pour conclure, je souhaite remercier mes amis de longue date et ma famille proche, qui m'ont accompagné tout au long de ma thèse. À Mickaël, avec qui nos parcours se suivent depuis l'ENS Cachan. Il me tarde de voir où nos carrières respectives vont nous mener, et de collaborer pour nos éventuels futurs projets personnels ! À Thierry, ami depuis le collège, prenant toujours de mes nouvelles malgré l'éloignement. Ton amitié m'est précieuse, et je te souhaite bonheur et réussite pour la suite ! À mes deux petits frères Sami et Nader, à qui j'ai adoré consacrer du temps suite à un long séjour à l'étranger. À mon père, toujours de bon conseil, et qui m'a soutenu tout au long de mes études. Enfin à ma mère, qui n'a eu de cesse de s'assurer de mon bien-être - notamment grâce à sa cuisine, forcément la meilleure au monde ! Papa, maman, j'espère vous avoir rendu fier, et vous le rendre encore à l'avenir.

Parameterized Complexity and New Efficient Enumerative Schemes for RCPSP

Abstract: Most scheduling problems are strongly NP-hard and have required intricate heuristics to be solved in practice. While yielding results short term, such heuristics rarely help to identify what fundamentally makes these problems difficult. Such analysis would uncover the similarities between seemingly different problems and facilitate strategy transfers. This would also prove useful whenever one would like to enrich a problem - for example by adding precedence relations or going from equal-length tasks to tasks of arbitrary time length.

One way to gather such knowledge is to go beyond classical complexity theory and consider parameterized complexity theory. Given a problem $\mathcal{P}$ we choose parameter k as some property of the input like the number of machines or the width of the precedence graph (if there is one). Depending on problem $\mathcal{P}$ and parameter k either we design algorithms which are *fixed-parameter tractable* with respect to k (i.e. which operate in time $f(k) \times poly(n)$ with f an arbitrary computable function), or we show that $\mathcal{P}$ remains hard even when k is small (typically via a reduction from a well-known difficult parameterized problem in the same manner as an NP-hardness proof). Then the set of parameters for which problem $\mathcal{P}$ is *fixed-parameter tractable* can be interpreted as a footprint. By comparing it to the set of working parameters from another NP-hard problem $\mathcal{P}'$, one could infer whether $\mathcal{P}$ is 'equivalent' to $\mathcal{P}'$, 'strictly harder' than $\mathcal{P}'$, or if both problems are difficult for different reasons.

In this thesis we study the parameterized complexity of the Resource-Constrained Project Scheduling Problem (RCPSP) and its subproblems, possibly enhanced with job time windows and precedence delays. With precedence delays we consider parameter ℓ_{max} - i.e. the maximum delay value appearing in the input - in the case of minimum delays, exact delays and the lesser studied maximum delays. For all delay types we show that scheduling unit-time jobs on a single machine with a single available delay value is hard even with small ℓ_{max}. This suggests that another property of the problem has to be bounded in order to deal with precedence delays.

This motivates the integration of job time windows to problems featuring precedence delays in order to broaden available parameter choices. We consider two parameters which have shown recent success in the literature. Namely *slack* σ - i.e. the maximum difference between the time window length of a job and its processing time - and *pathwidth* μ - i.e. the maximum number of job time windows which can include the same time unit. We also introduce a new parameter, the *proper level* q, which is defined as the maximum number of job time windows which can strictly include on both ends another job time window. We obtain several *fixed-parameter tractable* algorithms and set multiple hardness results with to respect to these parameters, both with or without precedence delays. We also consider various other scheduling parameters in order to give a more complete view of the parameterized landscape of RCPSP and its subproblems.

Keywords: complexity, parameterized, RCPSP, scheduling, algorithm design

Complexité Paramétrée et Nouveaux Schémas Enumeratifs Efficaces pour le RCPSP

Résumé : La plupart des problèmes d'ordonnancement sont fortement NP-difficiles et nécessitent des heuristiques élaborées pour être résolus en pratique. Bien que fournissant des résultats au court terme, ces approches expliquent rarement ce qui rend ces problèmes fondamentalement difficiles. Avoir cet éclairage pourrait mettre à jour des similitudes entre des problèmes apparemment distincts et faciliter les transferts de stratégie. Cela serait également utile dès lors que l'on souhaite enrichir un problème - par exemple en y ajoutant des relations de précédence, ou en passant de tâches de même durée à des tâches de durées arbitraires.

Pour acquérir une telle expertise, il est possible d'aller au-delà de la théorie classique de la complexité, et de se tourner vers la complexité paramétrée. Etant donné un problème $\mathcal{P}$ on choisit un paramètre k reflétant une propriété de l'instance en entrée, comme son nombre de machines ou, s'il y a des relations de précédence, la largeur du graphe sous-jacent. En fonction du problème $\mathcal{P}$ et du paramètre k, soit on développe des algorithmes dits *fixed-parameter tractable* (i.e. qui opèrent en temps $f(k) \times poly(n)$ avec f une fonction arbitraire), soit on montre que le problème reste difficile lorsque le paramètre est petit (typiquement via une réduction depuis un problème paramétré reconnu comme difficile, à la manière d'une preuve de NP-difficulté). L'ensemble des paramètres pour lesquels le problème $\mathcal{P}$ est *fixed-parameter tractable* peut alors être interprétée comme un "phénotype": en le comparant à celui d'un autre problème $\mathcal{P}'$, on peut déduire si les deux problèmes ont l'air équivalents, si l'un est une généralisation de l'autre, ou s'ils sont difficiles pour des raisons distinctes.

Dans cette thèse, nous étudions la complexité paramétrée du problème d'ordonnancement de projet sous contrainte de ressources (RCPSP) et de ses sous-problèmes, possiblement augmentés de délais de précédence et de fenêtres temporelles pour les tâches. Pour les délais de précédence nous étudions le paramètre ℓ_{max} - i.e. le valeur maximum de délai pouvant apparaître dans l'entrée - dans le cas de délais minimum, exact et maximum. Pour ces trois types de délais nous montrons que, même à petit ℓ_{max}, ordonnancer des tâches de durée unitaire sur une unique reste difficile, et ce même pour des durées de délai toutes identiques. Cela suggère de limiter une propriété supplémentaire dans le cas de problèmes avec délais de précédence.

A ce titre, nous proposons l'ajout de fenêtres temporelles pour les tâches afin d'élargir le panel de paramètres disponibles. Cela nous permet de considérer deux paramètres ayant récemment rencontré du succès dans la littérature : d'une part la *marge* σ - i.e. la différence maximum entre la taille de fenêtre d'une tâche et la durée de celle-ci -, d'autre part la *largeur de chemin* μ - i.e. le nombre maximum de fenêtres pouvant se chevaucher le long d'une même unité de temps. Nous introduisons également un nouveau paramètre, que nous appelons le *niveau propre* q, et qui indique le nombre maximum de fenêtres pouvant inclure strictement une même autre fenêtre le long de ses deux bornes. Vis-à-vis de ces paramètres nous proposons plusieurs algorithmes *fixed-parameter tractable*, et les complétons avec des preuves de difficulté paramétrée. Nous considérons également d'autres suggestions de paramètres pour proposer une vue d'ensemble plus complète de la complexité paramétrée du RCPSP et de ses sous-problèmes.

Mots clés : complexité, paramétré, RCPSP, ordonnancement, algorithmique

Résumé Long

Introduction

Cette thèse a pour but d'étudier le problème d'ordonnancement de projet sous contrainte de ressources (RCPSP), ses sous-problèmes et ses variantes dans le contexte de la complexité paramétrée. Étant donné un ensemble de tâches soumises à des contraintes de précédence et de ressources, l'objectif du RCPSP est de minimiser la date d'achèvement de l'ensemble du projet. Chaque tâche peut nécessiter une certaine quantité de différentes ressources, correspondant à des moyens de production renouvelables tels que la main-d'œuvre ou les machines dans une usine. Par exemple, si l'une des tâches consiste à peindre une voiture, elle peut nécessiter une machine spécifique et un employé capable de la faire fonctionner.

Bien que ce RCPSP soit un problème d'ordonnancement fondamental avec un large éventail d'applications, en pratique il est souvent difficile de trouver des ordonnancements optimaux pour plus d'une centaine de tâches dans un délai raisonnable. Cette difficulté peut être expliquée dans le contexte de la théorie de la complexité. En effet, de nombreux sous problèmes du RCPSP sont NP-difficiles au sens fort. Par exemple le cas où il n'y a qu'un seul type de ressource, des machines parallèles identiques et pas de contraintes de précédence. Ou encore celui des problèmes d'atelier (job-shops, flow-shops) où les tâches sont pré-affectées aux machines et où les précédences sont réduites à des chaînes. Ces sous-problèmes sont incomparables et sont donc très probablement difficiles pour des raisons différentes. Cependant, la théorie de la complexité classique n'offre pas les outils nécessaires pour distinguer plus finement les problèmes fortement NP-difficiles entre eux.

La théorie de la complexité paramétrée offre, en revanche, des moyens plus précis pour caractériser la difficulté des problèmes NP-difficiles. Étant donné un problème $\mathcal{P}$, on fixe un paramètre k choisi parmi les propriétés de la donnée du problème, comme le nombre de machines ou la largeur du graphe de précédence (s'il y en a un). En fonction du problème $\mathcal{P}$ et du paramètre k, on peut alors soit concevoir des algorithmes qui sont dits *fixed-parameter tractable (FPT)* par rapport à k (c'est-à-dire qui fonctionnent en temps $f(k)\cdot poly(n)$ avec f une fonction calculable arbitraire), soit montrer que $\mathcal{P}$ reste difficile même lorsque k est petit (typiquement via une réduction à partir d'un problème paramétré difficile bien

connu comme k-CLIQUE ou k-COLORING). L'ensemble des paramètres pour lesquels le problème $\mathcal{P}$ est FPT peut alors être interprété comme son portrait. En le comparant à l'ensemble des paramètres "fonctionnels" d'un autre problème NP-difficile $\mathcal{P}'$, on peut déduire si $\mathcal{P}$ est "équivalent" à $\mathcal{P}'$, "strictement plus difficile" que $\mathcal{P}'$, ou si les deux problèmes sont difficiles pour des raisons différentes.

Alors que la complexité paramétrée est largement utilisée dans la théorie des graphes depuis plus de quarante ans, ce n'est que récemment qu'elle a fait l'objet d'un intérêt croissant dans le domaine de l'ordonnancement. Au cours des dix dernières années, la plupart des approches permettant d'obtenir des algorithmes FPT nécessitaient de limiter le nombre de types de tâches d'une manière ou d'une autre, tandis que les paramètres structurels tels que la largeur w du graphe de précédence ont souvent conduit à des résultats négatifs [vBBB$^+$16, BGNS22]. Dans [MvB18] Mnich et van Bevern ont dressé une liste de quinze problèmes ouverts de complexité paramétrée dans le domaine de l'ordonnancement.

Juste avant cette thèse, plusieurs algorithmes FPT [MK21, BdWH21] ont été conçus avec un nouveau paramètre sur des sous-problèmes de RCPSP augmentés de fenêtres temporelles - avec pour chaque tâche j une date de disponibilité r_j et une date d'échéance $\bar{d}_j$. Ce paramètre, appelé pathwidth μ, est défini comme le nombre maximum de fenêtres de temps qui se chevauchent à tout moment - moins un, et peut être interprété comme la largeur de chemin du graphe d'intervalles sous-jacent. Les résultats obtenus pour une seule machine ne nécessitent notamment pas de limiter les durées des tâches.

A noter que Bodlaender et van der Wegen avaient introduit un an auparavant [BvdW20] un paramètre similaire, mais avec des fenêtres temporelles définies pour les chaînes de tâches et non les tâches individuelles. De plus ils considéraient des délais de précédence entre les tâches consécutives dans une chaîne.

Nous nous sommes donc attachés à réaliser une étude systématique de la complexité paramétrée des problèmes d'ordonnancements sous-problèmes du RCPSP, en y ajoutant au besoin fenêtres de temps et délais de précédence, pour différents paramètres. Notre approche est double. Tout d'abord, étant donné un problème $\mathcal{P}$, nous déterminons pour quels paramètres $\mathcal{P}$ est FPT et pour lesquels il ne l'est probablement pas. Cela définit une cartographie des paramètres qui pourrait être considérée comme le "portrait" de $\mathcal{P}$ et être comparée à la carte de paramètres d'autres problèmes d'ordonnancement. Deuxièmement, étant donné un paramètre k, nous déterminons quels problèmes d'ordonnancement sont FPT par rapport à k et lesquels ne le sont pas. Cela définit une frontière de complexité qui aide à identifier la "force" de ce paramètre. Le document est organisé selon cette dernière approche, un chapitre étant consacré à chacun des paramètres principalement étudiés : la valeur maximale de délai de précédence ℓ_{max}, la largeur de chemin μ, la marge de manœuvre σ et le niveau propre du graphe d'intervalles q.

Contexte

Dans le chapitre 2, nous donnons un aperçu du contexte dans lequel s'inscrit cette thèse. Tout d'abord, nous introduisons formellement le RCPSP, nous présentons les résultats existants sur ce problème dans la théorie classique de la complexité, et nous discutons de leurs limites.

Ensuite, nous prenons le temps de définir les notions de complexité paramétrée qui sont utilisées dans ce travail. Nous donnons les définitions des classes paramétrées et montrons comment relier les résultats de différents paramètres par des réductions et des relations de paramètres.

Enfin, nous donnons un aperçu général de la complexité paramétrée dans le domaine de l'ordonnancement. Nous montrons que de nombreux problèmes d'ordonnancement restent difficiles lorsque le nombre de type de tâches n'est pas limité. Nous notons que la grande majorité des algorithmes FPT récents limitent le nombre de types de tâches et/ou de types de machines, en particulier dans le cadre des problèmes à haute multiplicité.

Le Paramètre Délai Maximum

Dans le chapitre 3, nous considérons les problèmes à délais de précédence et prenons pour paramètre le plus grand délai ℓ_{max} indiqué dans l'entrée. Nous prouvons de nombreux résultats négatifs.

Tout d'abord, nous étudions les délais exacts ou maximaux sur une seule machine avec des chaînes de tâches unitaires. En présence de fenêtres temporelles, nous avons montré que le problème était para-NP-difficile avec l'un ou l'autre type de délais, même avec une seule valeur de délai. En l'absence de fenêtre temporelles, si le problème est toujours $W[1]$ difficile avec des délais exacts, nous montrons qu'il peut être résolu en temps linéaire avec des délais maximaux arbitraires.

Ensuite, avec des délais minimaux, nous démontrons plusieurs autres résultats négatifs. Pour l'ordonnancement sur une seule machine avec des tâches unitaires et une précédence générale, nous montrons que le problème est $XNLP-$difficile par rapport à ℓ_{max} combiné à la largeur du graphe de précédence w. Nous montrons ensuite que le problème est $W[2]-$difficile avec des chaînes de précédence si des délais supplémentaires de longueur zéro sont autorisés. Pour le paramètre ℓ_{max} seul, ce résultat a été renforcé en montrant que le problème à fenêtre de temps et machines parallèles est para-$NP-$difficile. Ce dernier résultat fait partie de notre publication IPEC 2022 [MHMK22a].

Nos résultats montrent que la limitation de la valeur maximale du délai ℓ_{max} n'est pas suffisante pour obtenir des algorithmes FPT pour la plupart des sous-problèmes d'ordonnancement de RCPSP avec des délais de précédence.

Le Paramètre Largeur de Chemin

Dans le chapitre 4, nous examinons la largeur de chemin μ pour lequel des résultats positifs avaient été déjà trouvés dans la littérature récente. Nous l'avons confirmé en trouvant un nouvel algorithme FPT pour l'ordonnancement de tâches avec des fenêtres temporelles et des contraintes de précédence sur une seule machine. Ce résultat a été présenté à MAPSP 2022 [HMMK22].

Cependant, lorsque l'on ajoute des délais de précédence, nous montrons que le problème devient para-NP-difficile quel que soit le type de délai, même avec des travaux unitaires et un graphe de précédence réduit à des chaînes. Ce travail a été présenté pour la première fois à ROADEF 2022. Il permet d'établir que la largeur de chemin μ seule en tant que paramètre n'était pas adaptée pour traiter les problèmes avec délais.

Cependant, lorsqu'on le combine avec la valeur maximale de délais ℓ_{max}, nous avons proposé un algorithme FPT avec un graphe de précédence quelconque et des tâches unitaires sur machines parallèles. Le résultat négatif et l'algorithme FPT ont été inclus dans notre publication IPEC 2022 [MHMK22a]. Cela a motivé l'intégration des fenêtres de temps aux problèmes comportant des délais de précédence afin d'élargir les choix de paramètres disponibles.

Le Paramètre Marge

Le chapitre 5 est consacré à la marge maximale d'une tâche, notée σ, un autre paramètre basé sur les fenêtres temporelles. Ce paramètre représente le nombre maximal de positions que peut prendre une tâche dans sa fenêtre de temps - moins un. Sur une seule machine ou un nombre fixe de processeurs parallèles identiques, nous avons montré que σ était plus fort que la largeur de chemin μ. Cela a permis de déduire plusieurs résultats FPT concernant σ à partir des progrès récents réalisés avec la largeur de chemin μ.

Dans le reste du chapitre, nous nous sommes concentrés sur l'ordonnancement sur une seule machine avec des fenêtres temporelles et des délais de précédence. Nous tirons des conclusions similaires à celles de la largeur de chemin μ. Avec la marge σ seule, nous avons montré que l'ordonnancement sur une seule machine était para-NP difficile pour les trois types de délais de précédence, même en se limitant aux tâches couplées (i.e. un graphe de précédence réduit à des arcs disjoints) de durées unitaires et une seule valeur de délai. Ensuite, lorsqu'il est combiné avec ℓ_{max}, le problème sur une seule machine devient FPT même avec un graphe de précedence général et des tâches de durée quelconque - alors qu'il reste ouvert en ce qui concerne le paramètre $\mu + \ell_{max}$. Le contenu de ce chapitre fait l'objet d'un article en préparation, destiné à une soumission à la revue Discrete Applied Mathematics dans un proche avenir.

Le Paramètre Niveau Propre

Au chapitre 6, nous proposons un nouveau paramètre basé sur les fenêtres temporelles des tâches. Nous l'appelons le niveau propre q et nous montrons qu'il est plus faible que la largeur de chemin μ. Cela implique que la plupart des problèmes que nous avons étudiés sont para-NP-difficiles pour ce paramètre q.

Néanmoins, nous avons réussi à obtenir un algorithme FPT pour l'ordonnancement de tâches avec contraintes de précédence et fenêtres de temps sur une seule machine. L'algorithme repose sur la dominance d'ordonnancements avec une structure très particulière, et la programmation dynamique. Ceci a conduit à notre publication dans ISCO 2024 [MHMK24a].

Par la suite nous montrons quelques résultats négatifs supplémentaires concernant ce paramètre q en présence de machines parallèles et de tâches unitaires - un problème pourtant connu pour être FPT par rapport à μ [MK21].

Autres Résultats Paramétrés

Enfin, dans le chapitre 7, nous présentons plusieurs autres résultats paramétrés obtenus au cours de cette thèse.

Nous discutons en premier lieu de la possibilité d'algorithmes de kernelisation pour les problèmes d'ordonnancement avec fenêtres de temps. Nous montrons qu'un noyau polynomial par rapport à la largeur de chemin μ a peu de chance d'exister, même dans le cas d'une seule machine.

Afin d'explorer plus avant les conditions d'existence d'un noyau polynomial, nous proposons un nouveau paramètre, la couverture par sommets vc. Ce paramètre, plus fort que μ, nous permet de définir un noyau polynomial pour le problème à une machine - combiné avec le temps de traitement maximum p_{max} dans un premier temps, puis seul.

Malheureusement, nous montrons un résultat négatif dans le cas de tâches préaffectées à des sous-ensembles de machines parallèles identiques. Nous montrons que le problème devient $W[1]$-difficile paramétré par vc. Il est donc peu probable de trouver un algorithme de kernelisation pour ce problème avec vc comme paramètre. Ces résultats sur la kernelisation font l'objet d'une soumission à la conférence IPEC 2024.

Ensuite, nous avons considéré d'autres paramètres structurels très utilisés dans la théorie des graphes, comme la largeur arborescente ou la twin-width, et nous étudions leur utilisation potentielle dans les problèmes d'ordonnancement avec des fenêtres de temps. Nous montrons que la plupart des paramètres définis sur le graphe d'intervalles induit par les fenêtres de temps ne parvient pas à distinguer les cas faciles des difficiles. Ils sont soit trop "optimistes" comme la twin-width, soit trop "pessimistes" comme le pathwidth μ.

Enfin, nous proposons une notion de paramètre moyen qui pourrait mieux rendre compte de la complexité effective en temps et en espace de nombreux algorithmes paramétrés existants pour les problèmes d'ordonnancement.

Conclusion et Perspectives

Le travail effectué dans cette thèse a permis d'obtenir une cartographie précise de la complexité paramétrée des sous-problèmes d'ordonnancement du problème RCPSP. Nous résumons les résultats obtenus dans la Figure 1.

Bien entendu de nombreuses pistes restent à explorer suite à ce travail. Tout d'abord, on peut s'inspirer du cadre MIMO développé pour les porblèmes à haute multiplicité, et définir davantage de métathéorèmes pour les problèmes d'ordonnancement. Pour chaque paramètre, l'objectif serait de caractériser les propriétés des tâches qui sont essentielles pour obtenir un résultat *FPT*.

En ce qui concerne la largeur de chemin μ, nous avons un algorithme *FPT* soit sur une seule machine avec des tâches de durée quelconques, soit sur plusieurs machines avec des tâches de durées unitaires. Cela poserait donc le nombre de machines et la durée des tâches comme des caractéristiques essentielles vis-à-vis de ce paramètre. En revanche, nous estimons que l'ajout de contraintes de précédence et de ressources renouvelables avec des demandes non unitaires n'a aucun impact sur un résultat *FPT* impliquant le paramètre μ. Ainsi, nous conjecturons que le RCPSP avec des tâches unitaires est *FPT* paramétré par μ.

À plus long terme, le but ultime serait de trouver un analogue au théorème de Courcelle pour les problèmes d'ordonnancement, par exemple en ce qui concerne la largeur de chemin μ pour les problèmes comportant des fenêtres temporelles pour les tâches. Cela pourrait s'avérer délicat, sachant que les problèmes d'ordonnancement sont notoirement hétérogènes dans les machines utilisées et dans les propriétés des tâches. Il s'agit d'un frein notoire dans l'optique de réutiliser de manière directe une grande partie du travail effectué en théorie paramétrée des graphes.

Une autre question théorique concerne les algorithmes de kernelisation pour les problèmes d'ordonnancement avec des paramètres structurels, comme nous l'avons fait pour la couverture par sommets *vc* dans le chapitre 7. De nombreux algorithmes *FPT* ont été trouvés avec la largeur de chemin μ, la marge σ et le niveau propre q au cours des cinq dernières années, alors qu'à notre connaissance aucun algorithme de kernelisation n'a été conçu par rapport à l'un de ces paramètres. Le principal défi reste de trouver des règles de réduction appropriées qui nous permettraient de supprimer un nombre important de tâches de l'instance d'origine. Même si nos récents progrès avec la couverture par sommets *vc* sont encourageants, nous nous attendons à ce que les règles de réduction souhaitées soient plus complexes et conduisent au mieux à des noyaux de taille exponentielle.

Par ailleurs, nous observons que les résultats paramétrés dans le domaine de l'ordonnancement ont été essentiellement théoriques. Alors que le premier objectif est d'identifier les problèmes *FPT* sans accorder beaucoup d'attention à la complexité temporelle, certains des algorithmes proposés ont déjà une chance de rivaliser avec l'état de l'art. En particulier, nos algorithmes *FPT* sur une seule machine concernant la largeur de chemin μ ou le niveau propre q pourraient être considérés d'emblée dans certains contextes, au vu de leur dépendance temporelle assez faible à la fois sur la taille de l'entrée et le paramètre - respec-

tivement quadratique et mono-exponentielle. Il est cependant à noter que les valeurs de ces paramètres ont tendance à être élevées dans les cas généraux. Par exemple, avec des fenêtres de temps tirées aléatoirement de manière uniforme, les deux paramètres ont une valeur moyenne au moins proportionnelle à la taille de l'entrée, ce qui est loin d'être idéal dans un cadre paramétré. Néanmoins, il est possible d'identifier des instances pratiques spécifiques dans lesquels nos paramètres pourraient être performants.

On peut également noter que dans la littérature, de nombreux algorithmes approchés ou schémas d'approximation ont été proposés pour les problèmes d'ordonnancement. Il est donc naturel d'envisager une étude d'algorithmes paramétrés approchés, qui pourraient permettre, pour certains paramètres, d'avoir un compromis entre complexité et optimalité. Cette direction est un champ encore très ouvert avec très peu de résultats jusqu'à présent.

Enfin, avec le développement rapide de la complexité paramétrée dans le domaine de l'ordonnancement, il est devenu de plus en plus difficile de rester à jour et de suivre tous les progrès réalisés dans ce domaine. Ainsi, les nouveaux arrivants bénéficieraient grandement d'une archive publique rassemblant tous les résultats de complexité paramétrée en ordonnancement. C'est dans cette optique que nous prévoyons de contribuer au Scheduling Zoo [BKD10] dans un avenir proche.

Chap.	Paramètre	Problème	Résultat
3	ℓ_{max}	$1\|chains(\ell), p_j = 1, r_j, \bar{d}_j\|\star$	Para-NP-difficile avec délais exacts ou maximaux.
		$1\|chains(\ell), p_j = 1\|C_{max} < D$	$W[1]$-difficile avec délais exacts.
		$1\|chains(\ell_{i,j}), p_j = 1\|C_{max} < D$	$\mathcal{O}(n)$ avec délais maximaux.
		$^{1\|chains(0,\ell),\, p_j = 1,\, r_j\|C_{max} < D}$ $1\|chains(0,\ell), p_j = 1, \bar{d}_j\|\star$	NP-difficile avec délais maximaux.
		$1\|prec(\ell), p_j = 1\|C_{max} < D$	$XNLP$-difficile($\ell_{max} + w$) avec délais minimaux.
		$1\|chains(0,\ell), p_j = 1\|C_{max} < D$	$W[2]$-difficile(ℓ_{max} + #chaînes) avec délais minimaux.
		$P\|chains(0,\ell), p_j = 1, r_j, d_j\|\star$	Para-NP-difficile($\ell = 1$) avec délais minimaux.
4	μ	$1\|prec, r_j, \bar{d}_j\|C_{max}$	FPT en temps $\mathcal{O}(\mu^2 \cdot 4^{\mu} \cdot n + n^2)$.
		$1\|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j\|\star$	Para-NP-difficile pour les trois types de délais.
	$\mu + \ell_{max}$	$P\|prec(\ell_{i,j}), p_j = 1, r_j, \bar{d}_j\|\star$	FPT pour délais minimaux.
5	σ	Single machine	$\mu \leq 2\sigma$
		Pm	$\mu \leq 2(\sigma + 1) \cdot m - 1$
		-	Plusieurs FPT inférés.
		$1\|(1,\ell,1), r_j, \bar{d}_j\|\star$	Para-NP-difficile pour les trois types de délais.
	$\sigma + \ell_{max}$	$1\|prec(\ell_{i,j}), r_j, \bar{d}_j\|\star$	FPT pour les trois types de délais combinés.
6	q	-	$q \leq \mu$
		$P2\|r_j, \bar{d}_j\|\star$ $1\|(1,\ell,1), r_j, \bar{d}_j\|\star$	Para-NP-difficulté inférée.
		$1\|prec, r_j, \bar{d}_j\|C_{max}$	FPT en temps $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N + n^2)$.
		$P\|prec, p_j = 1, r_j, \bar{d}_j\|\star$	Para-NP-difficile($q + D$).
			$W[2]$-difficile($q + w$).
7	μ	$1\|r_j, \bar{d}_j\|\star$	Pas de noyau polynomial (sauf si $NP \subseteq coNP/poly$).
	vc	$1\|prec, r_j, \bar{d}_j\|\star$	Noyau polynomial avec $(vc + p_{max})$ et vc.
		$P\|\mathcal{M}_j, r_j, \bar{d}_j\|\star$	$W[1]$-difficile.
	tww, $pthin, cw$	$1\|r_j, \bar{d}_j\|\star$ $P\|tree, p_j = 1, r_j, \bar{d}_j\|\star$	Para-NP-difficile.
	μ_{avg}, σ_{avg}, q_{avg}	-	plusieurs FPT inférés.

Figure 1: Résumé des résultats de la thèse.

Contents

Chapter 1

Introduction

1.1 Motivation

This thesis aims at studying the Resource-Constrained Project Scheduling Problem (RCPSP), its subproblems and its variants in the context of parameterized complexity. Given a set of jobs subject to precedence constraints, the goal of RCPSP is to minimize the completion of the whole project. Each job can require some amount of several resources, which are renewable means of production like workforce or machines in a factory. For example if one of the jobs is to paint a car then it may involve a specific machine and an employee who can operate it.

While this RCPSP is a cornerstone scheduling problem with a wide of range of applications, in practice it is often tough to find optimal schedules for more than a hundred jobs within a reasonable time. Such difficulty can be explained in the context of computational complexity theory. In the identical parallel machine setting, i.e. when restricted to a single resource type, several subproblems of RCPSP are known to be strongly-NP-hard. This is the case for all subproblems given in Figure 1.1, which are most likely difficult for different reasons. However the classical branch of complexity theory does not offer the tools to distinguish between strongly-NP-hard problems in this manner.

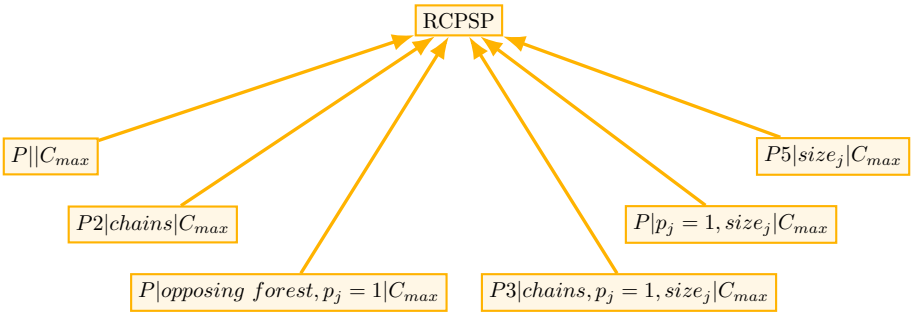


Figure 1.1: Some strongly NP-hard subproblems of RCPSP.

In contrast parameterized complexity theory gives more refined ways to characterize the difficulty of NP-hard problems. Given a problem $\mathcal{P}$ we choose parameter k as some property of the input like the number of machines or the width of the precedence graph (if there is one). Depending on problem $\mathcal{P}$ and parameter k either we design algorithms which are *fixed-parameter tractable* with respect to k (i.e. which operate in time $f(k) \cdot poly(n)$ with f an arbitrary computable function) and n the input size, or we show that $\mathcal{P}$ remains hard even when k is small (typically via a reduction from a well-known difficult parameterized problem like k-CLIQUE or k-COLORING). Then the set of parameters for which problem $\mathcal{P}$ is *fixed-parameter tractable* can be interpreted as a footprint. By comparing it to the set of 'working' parameters from another NP-hard problem $\mathcal{P}'$, one could infer whether $\mathcal{P}$ is 'equivalent' to $\mathcal{P}'$, 'strictly harder' than $\mathcal{P}'$, or if both problems are difficult for different reasons.

While parameterized complexity has been extensively used in graph theory for forty years, it has seen increasing interest in scheduling only recently. In the past ten years most of the successful approaches have required to bound the number of job types in some way, while structural parameters like the width w of the precedence graph have often led to negative results [vBBB$^+$16, BGNS22]. In [MvB18] Mnich and van Bevern listed fifteen open parameterized complexity problems in scheduling.

Right before this thesis several fixed-parameter tractable algorithms [MK21, BdWH21] were designed with a new parameter on subproblems of RCPSP augmented with job time windows - with a release date r_j and a deadline $\bar{d}_j$. This parameter, called pathwidth μ, is defined as the maximum number of overlapping job time windows at any time, and can be interpreted as the pathwidth of the underlying interval graph. The obtained single machine results notably do not require to bound the processing time values. The year before in [BvdW20] Bodlaender and van der Wegen introduced a parameter with a similar definition but with time windows defined along job chains, and possibly a delay between consecutive jobs in a chain. This suggested further consideration of other potential parameters based on time windows and/or precedence delays.

1.2 Summary

In this work we study the parameterized complexity of RCPSP and its subproblems, possibly enhanced with job time windows and/or precedence delays. Our approach is two-fold. First given a problem $\mathcal{P}$ we find for which parameters $\mathcal{P}$ is fixed-parameter tractable and for which it is probably not. This defines a parameter map which could be viewed as the 'footprint' of $\mathcal{P}$ and be compared to the parameter map of other scheduling problems. Second given a parameter k we determine which problems are fixed-parameter tractable with respect to k and which are not. This defines a complexity frontier which helps to identify the 'strength' of this parameter. More details are given in Subsection 2.2.3.

This document is organized based on the latter approach with a chapter dedicated to each of our mainly studied parameters: maximum delay value

ℓ_{max}, pathwidth μ, slack σ and proper level q. For these parameters we give their associated state of the art in the introduction of their respective chapter and we provide a problem map at the end of it.

In Chapter 2 we give an overview of the background behind this thesis. First we introduce RCPSP formally, present the existing results on this problem in classical complexity theory and discuss their limitations. Then we take some time to define the parameterized complexity notions which are used in this work. We give the definitions of the parameterized classes and show how to connect results from different parameters via reductions and parameter relations. Finally we give a general survey of parameterized complexity in scheduling. We show that a lot of scheduling problems remain difficult when the diversity of jobs properties is not limited in some way. We note that a vast majority of the recent fixed-parameter tractable algorithms bound the number of job types and/or machine types in some way, especially in the high-multiplicity setting.

Chapter 3 is dedicated to scheduling problems with precedence delays. We explore what happens when such delays are added to several subproblems of RCPSP. We do so via parameter ℓ_{max}, which is the maximum delay value that can appear in the precedence graph given in the input. We show that even when the values of the precedence delays are bounded these subproblems become much harder. We study three types of precedence delays: exact, maximum and minimum. We start by setting several results with exact and maximum delays, some of them being closely related to bin packing and bandwidth results in the literature. Then we focus on minimum precedence delays with unit-time jobs. We prove that single machine scheduling with general precedence and equal-length minimum precedence delays is hard even with bounded ℓ_{max} and bounded precedence graph width w. Next we adapt this reduction to the special case of precedence chains when extra minimum precedence delays of length zero are allowed. We end this chapter by strengthening the latter result in the identical parallel machine setting.

In Chapter 4 we introduce job time windows and we set pathwidth μ as our parameter, which is the maximum number of overlapping job time windows at any given time. This parameter has often been used successfully in the literature [BdWH21, MK21, HMK23, TCH$^+$23]. We give a fixed-parameter tractable algorithm on single machine scheduling with job time windows and precedence constraints. However adding precedence delays makes the problem NP-hard with fixed pathwidth, even with unit-time jobs and precedence constraints restricted to chains. We prove that this is the case for all three precedence delay types. Then we show that this problem becomes fixed-parameter tractable again when combining μ and ℓ_{max} in the identical parallel machine setting with unit-time jobs.

Chapter 5 is dedicated to slack σ which is another parameter previously considered in the literature when dealing with job time windows - albeit to a lesser extent [BdWH21, HMK23]. We show that σ is a strictly stronger parameter than pathwidth μ in the single machine setting and the identical parallel machine setting with a fixed number of machines. Despite this, we show that single machine scheduling with equal-length precedence delays is NP-hard with

fixed slack, even with unit-time jobs grouped into coupled tasks. Like with pathwidth we prove that this is the case for all three precedence delay types. Then with parameter $\sigma + \ell_{max}$ we give a fixed-parameter tractable algorithm on single machine scheduling with job time windows and general precedence constraints and precedence delays of any type.

In Chapter 6 we introduce a new parameter, which we call proper level q. We show that q is a smaller parameter than pathwidth μ and yet can still yield fixed-parameter tractable algorithms on scheduling problems with job time windows. We describe such an algorithm on single machine scheduling with job time windows and precedence constraints. This leads to a natural extension of the well-known *Earliest Deadline* rule [War59]. Next we argue that proper level q is a strictly smaller parameter than pathwidth μ. We do so by setting hardness results on identical parallel machine scheduling with unit-time jobs and general precedence constraints, which is known to be fixed-parameter tractable with respect to μ [MK21].

In Chapter 7 we present several other parameterized results obtained during this thesis and give input on their potential uses in scheduling problems with job time windows. First we consider kernelization algorithms. After showing that a polynomial kernel is unlikely to exist on single machine scheduling with respect to pathwidth μ, we introduce the vertex cover vc of the integral graph defined by the job time window overlaps as a parameter. We propose a polynomial kernel with respect to vc on single machine scheduling with job time windows, which is uncommon on scheduling problems with a parameter that does not bound the number of job types in some way. Then we examine other parameters based on this time window interval graph like twin-width. We show how they compare to parameters μ and q. We argue that most of these parameters are unlikely to yield any new fixed-parameter tractable results on scheduling problems and share some intuition behind our reasoning. At last we propose a notion of average parameters which may reflect better the effective time and space complexity of multiple existing fixed-parameter algorithms. We illustrate this concept on parameters μ, σ and q.

In Chapter 8 we give closing thoughts and several perspectives induced by this work. We claim that we successfully clarified the parameterized landscape of several subproblems of RCPSP when enhanced with job time windows and/or precedence delays. We conclude by suggesting several ways to pursue the work done in this thesis.

Finally Annex A gathers additional information to help navigate the work done in this thesis. First we list the publications produced as part of this thesis. Then we give a non-exhaustive list of the notations used throughout this work, as well as concise definitions of the complexity classes and the parameters mentioned in this thesis. Eventually we offer a figure index for problem maps and parameter maps.

Chapter 2

Background

In this chapter we depict the overall scientific background behind this thesis. Section 2.1 introduces RCPSP and its known classical complexity results, both with and without job time windows and/or precedence delays. Then in Section 2.2 we present the parameterized complexity notions which will be used throughout this work. Next in Section 2.3 we summarize the current state of parameterized complexity in scheduling. We end this chapter with some concluding remarks in Section 2.4.

2.1 RCPSP in Classical Complexity Theory

In this section we give an overview of the literature available on RCPSP.

2.1.1 Base RCPSP

We begin by defining the base problem formally:

RCPSP

Input: A set $\mathcal{J}$ of jobs, a deadline D, a partial order $prec$ on $\mathcal{J}$, a set $\mathcal{R}$ of renewable resources, for each resource $\rho \in \mathcal{R}$ the available amount $\mathcal{R}_\rho$, and for each $j \in \mathcal{J}$ a processing time $p_j \in \mathbb{N}$ and the amount $res_{j,\rho} \leq \mathcal{R}_\rho$ of resource $\rho \in \mathcal{R}$ that it uses during its execution.

Question: Is there a feasible schedule $\tau : \mathcal{J} \to \mathbb{N}_0$ with makespan lower than or equal to D? I.e. can you set a starting time $\tau(j) \in \mathbb{N}_0$ for each job $j \in \mathcal{J}$ such that:

1. for every relation $i \to j$ in $prec$ job i finishes before job j starts, i.e. $\tau(i) + p_i \leq \tau(j)$,
2. at any time t at most $\mathcal{R}_\rho$ units of each resource ρ are used, i.e. $\sum_{\{j \in \mathcal{J}, \tau(j) \leq t < \tau(j)+p_j\}} res_{j,\rho} \leq \mathcal{R}_\rho$,
3. makespan $C_{max} = max_{j \in \mathcal{J}}(\tau(j) + p_j)$ is lower than or equal to D?

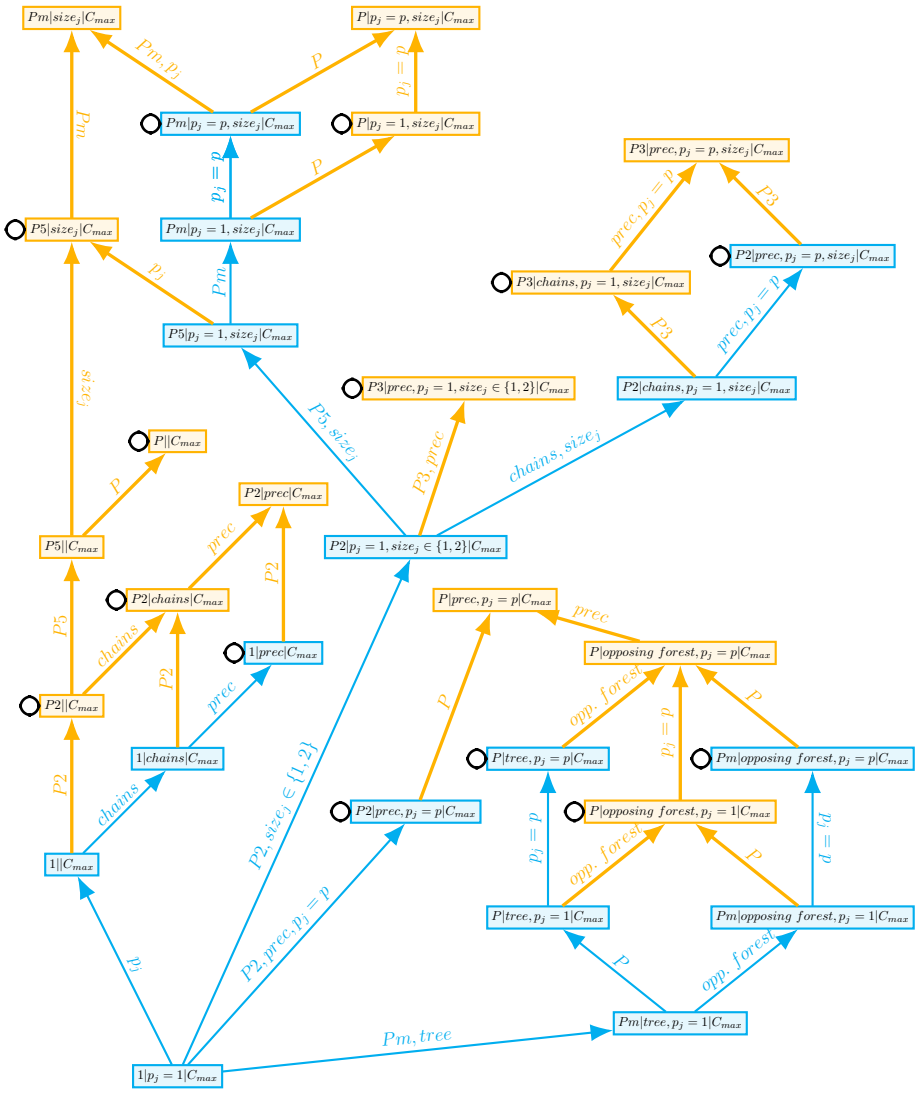


Figure 2.1: Subproblems of RCPSP.

As we mentioned in the introduction RCPSP has numerous incomparable strongly NP-hard subproblems, which illustrates the complex nature of this problem. In fact in [Uet02] Uetz showed that it cannot even be approximated in polynomial time with a factor of $n^{1-\epsilon}$ for any constant $\epsilon > 0$. In [HBS18] Habibi et al. counted more than two hundred articles in the field of RCPSP between 1980 and 2017. The main families of algorithmic techniques were compiled by Artigues et al. in [ADN13]. They range from formulations to resource flow networks and mixed-integer linear programs, lower bound computations from critical paths, resolution via constrained programming and/or branch-and-

bound, to heuristics typically using list scheduling, local search neighborhoods, column generation or metaheuristics like tabu search.

While numerous extensions and variants of RCPSP have been considered by the literature (see [HBS18]), in this work we focus on the base definition which minimizes makespan C_{max}. We also study the associated decision problems, noted either $(C_{max} < D)$ or $\star$ as the objective, from which the minimum makespan can be determined by a binary search on the objective threshold.

In the light of the difficulty behind RCPSP, it is decisive to determine the frontiers of polynomial-time solvability among its subproblems. We give a problem mapping in Figure 2.1 which discriminates between the polynomial-time solvable subproblems of RCPSP in blue and the NP-hard ones in orange. We denote $(\mathcal{P} \rightarrow \mathcal{P}')$ if $\mathcal{P}$ is a subproblem of $\mathcal{P}'$.

Since the problem is already strongly NP-hard with a single resource [GJ78], attempts at finding a polynomial-time algorithm often to bound the resource amounts and the job processing times. The identical parallel machine environment is a common way to simulate a single resource. With arbitrary processing times and unit resource amounts only the single machine case has a polynomial-time algorithm $\mathcal{O}(n^2)$ [Law73] whereas the two machine case is NP-hard [LRKB77] and even strongly NP-hard with chains of precedence relations [DLY91].

As a consequence most approaches considered unit-time or equal processing times. In the absence of precedence relations the problem with arbitrary resource amounts can be solved in linear time on a fixed number of machines [BDW86]. Such resource amounts on a single resource are denoted by $size_j$ and specify the number of machines required to process job j. Note that having a fixed number of machines is crucial, as the problem becomes strongly NP-hard with an unbounded number of machines [Llo81].

Still considering unit-time or equal processing times but with the addition of precedence relations the problem is already strongly NP-hard with three machines, either with general precedence and resource amounts in $\{1, 2\}$ [Llo81] or with chains and arbitrary resource amounts [BL96]. Nevertheless with unit resource amounts the problem can be solved in polynomial time on multiple machines, either on an unbounded number of machines with tree-like precedence [Hu61] or on a fixed number of machines with an opposing forest (i.e. composed of at least one intree and one outtree) [GJTY83]. With both an opposing forest and an unbounded number of machines the problem becomes strongly NP-hard [GJTY83], so the complexity frontier is tight.

2.1.2 Extensions to Job Time Windows and Precedence Delays

In the light of the previous subsection, the range of polynomial-time solvable subproblems of RCPSP is rather limited. This worsens when generalizing the base problem in hopes of expanding the scope of applicable problems. In this work we study two such extensions: job time windows and precedence delays.

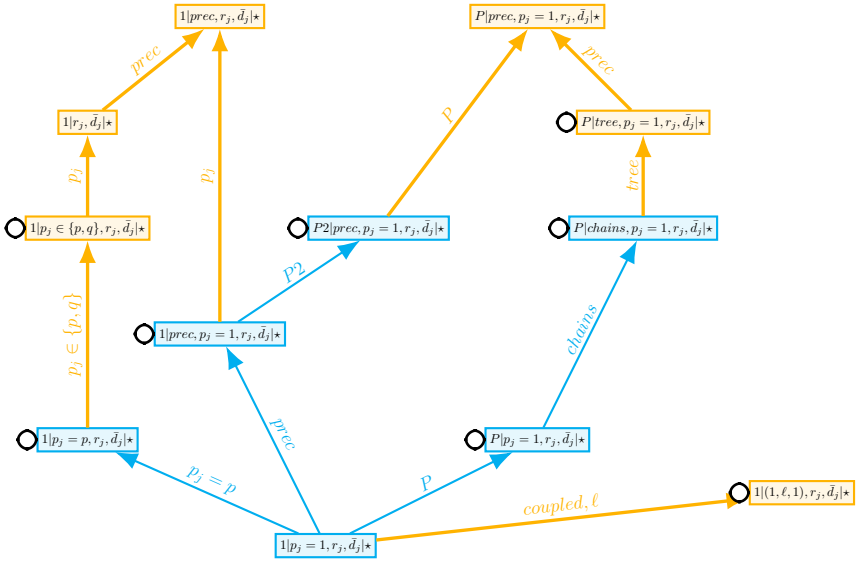


Figure 2.2: Subproblems of RCPSP enhanced with job time windows.

Time windows are a combination of a release r_j and a deadline $\bar{d}_j$. A job j can only be started at its release date or later and must be completed by its deadline. This defines a time window within which the job must be processed. Figure 2.2 gives the updated problem map. Then the single machine case becomes strongly NP-hard even with two distinct processing times greater than one (e.g 2 and 3) [EdW14]. With unit processing times the two machine case with general precedence can still be solved in quadratic time [GJ77]. However with an unbounded number of machines the problem with tree-like precedence becomes strongly NP-hard [GJ77]. Baptiste et al. showed that precedence constraints can be restricted to chains to retrieve polynomial-time solvability [BBKT04].

Now let us consider subproblems of RCPSP enhanced with precedence delays. Recall that given a precedence relation from job i to job j, the latter cannot start before the former is completed. A precedence delay builds upon such a relation and add an extra time constraint to it. We distinguish three types: exact, minimum and maximum. Then job j must start exactly, at least and at most $\ell_{i,j}$ time units after job i is completed respectively. Figures 2.3, 2.4 and 2.5 display the respective updated problem map. Note that the type of delay greatly impacts which problems remain polynomial-time solvable.

Among precedence delays the exact ones are definitely the most restrictive. Indeed scheduling chains of unit jobs on a single machine is NP-hard even with chains given in unary and with the same exact delay value on every relation (see Section 3.2). As such most results with exact precedence delays were obtained in the context of coupled task scheduling - see the recent survey by Khatami

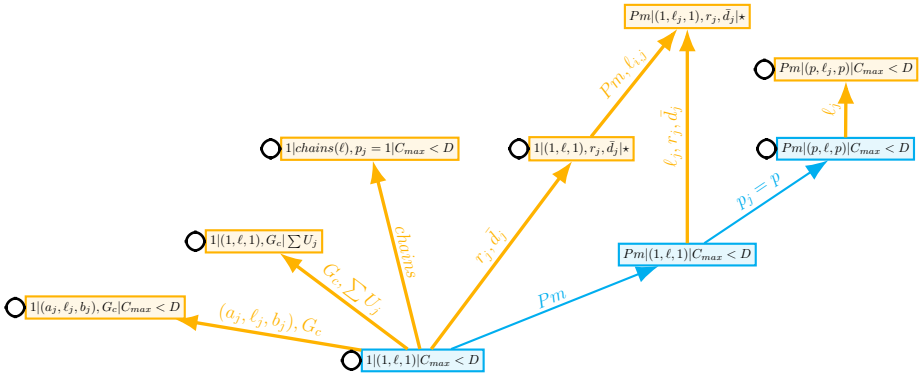


Figure 2.3: Subproblems of RCPSP enhanced with exact precedence delays.

et al. [KSC20]. The problem becomes notoriously difficult with either job time windows or a precedence graph G_c which specifies whenever two coupled tasks are allowed to interleave each other [BG19]. On a fixed number of machines Khatami et al. found a rare case where the problem can be solved in polynomial time, in the form of identical coupled tasks with jobs of equal length [KOS23].

With minimum precedence delays the single machine problem with identical coupled (unit) tasks remains strongly NP-hard in the presence of job time windows (see Section 5.3). However with either release dates or deadlines alone Bruno et al. showed that the problem can be solved efficiently even when the precedence graph is an outforest or an inforest respectively [BJS80]. Still Engels proved that the problem becomes difficult again with chains with either two distinct delay values or processing times one and two [Eng00].

Finally maximum precedence delays have been the least studied type. The single machine case with a single delay value is actually equivalent to the DIRECTED BANDWIDTH problem (see Section 3.2). Such a connection infers that unlike the minimum delay type the problem is NP-hard with tree-like precedence [GGJK78]. With coupled tasks or chains problems become quickly difficult in the presence of release dates and/or deadlines. However in their absence we have a straightforward linear time method even with arbitrary maximum precedence delays (see Section 3.2). This confirms that each precedence delay type sets its own set of restrictions and therefore must be studied individually.

In conclusion whether we consider base RCPSP or enhancements with job time windows and/or precedence delays, the vast majority of the subproblems are strongly NP-hard and thus unlikely to admit a polynomial time algorithm. In the face of such limitations, we view the parameterized complexity framework as an opportunity to expand the scope of tractable problems.

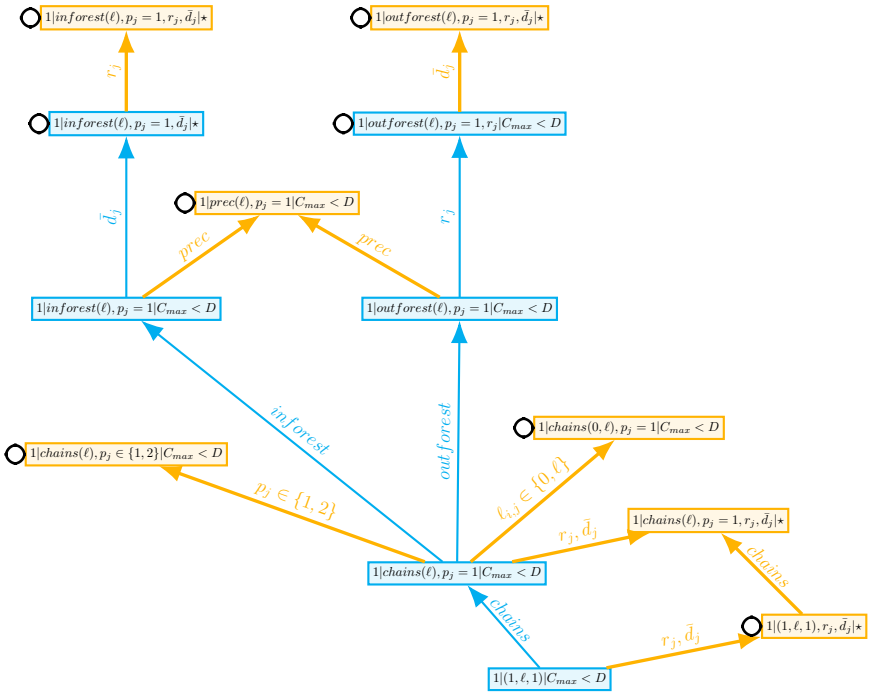


Figure 2.4: Subproblems of RCPSP enhanced with minimum precedence delays.

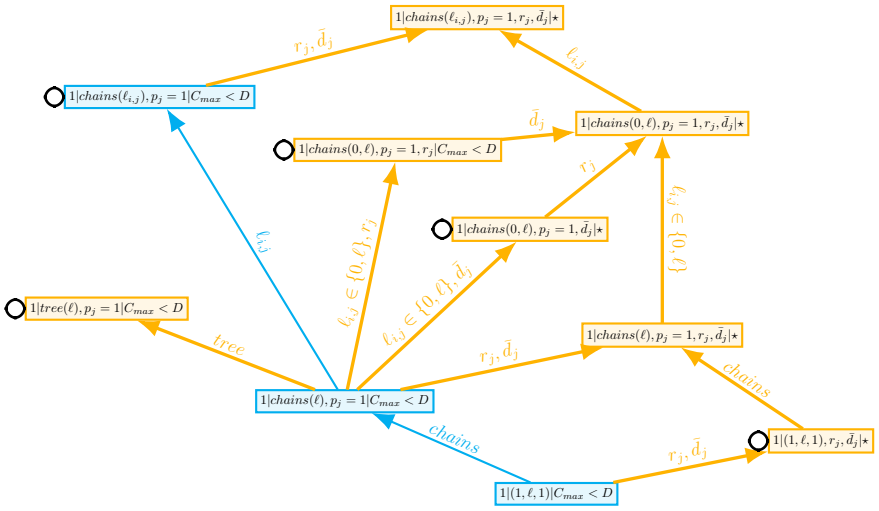


Figure 2.5: Subproblems of RCPSP enhanced with maximum precedence delays.

2.2 Basics of Parameterized Complexity Theory

In this section we formally introduce several definitions and notions from parameterized complexity theory. The given definitions are based on book [FG98] by Flum and Grohe, book [CFK+15] by Cygan et al. and paper [BGNS22] by Bodlaender et al..

Definition 1. *A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$ where Σ is a fixed, finite alphabet. For an instance $(I, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.*

Given a problem $\mathcal{P}$ and a parameter k, we will denote $(\mathcal{P}, k)$ the corresponding parameterized problem.

2.2.1 Toolkit for Positive Results

We start with the common ways to characterize that a parameterized problem is tractable.

The FPT and XP Classes

First we define the FPT class the following way:

Definition 2. *A problem $\mathcal{P}$ is called fixed-parameter tractable (FPT) parameterized by k if the corresponding parameterized problem can be solved by a deterministic algorithm in time $f(k) \cdot |I|^c$ where f is a (nondecreasing) computable function and c is a constant independent of both parameter k and instance size $|I|$.*

The complexity class containing all fixed-parameter tractable problems is called FPT.

When this is the case then we write that $\mathcal{P}$ is $FPT(k)$ as a shorthand notation. Note that asking f to be nondecreasing is a convenient hypothesis which changes little to the theory. Indeed one can replace f with $\hat{f} : k \mapsto (max_{0 \leq i \leq k} f(i))$, which is also computable. So for the rest of this document such computable functions will be assumed to be nondecreasing.

The main benefit of a FPT algorithm is that the super-polynomial component of the time complexity only depends on the parameter and thus is independent from the instance size. Essentially the difficulty of the problem is enclosed in parameter k. This contrasts with the XP class given below.

Definition 3. *A problem $\mathcal{P}$ is called slice-wise polynomial (XP) parameterized by k if the corresponding parameterized problem can be solved by a deterministic algorithm in time $f(k) \cdot |I|^{g(k)}$ where f and g are computable functions.*

The complexity class containing all slice-wise polynomial problems is called XP.

In other words each fixed value of k can be interpreted as a slice of instances, and an XP algorithm operates in polynomial time in each of these slices. However unlike with FPT the degree of the polynomial in the instance size depends

on the value of k. While this gives a looser notion of tractability, it is a common first step when studying any parameterized problem.

Parameterized Reductions

Next we define what a parameterized reduction is:

Definition 4. *Let $\mathcal{P}$ be a decision (resp. optimization) problem and let k, k' be two parameters. A parameterized reduction from $(\mathcal{P}, k)$ to $(\mathcal{P}', k')$ is an algorithm that, given an instance (I, k) of the first problem, outputs an instance (I', k') of the second problem such that:*

1. *(I, k) is a positive (resp. optimal) instance if and only (I', k') is a positive (resp. optimal) instance,*
2. *$k' \leq g(k)$ for some computable function g,*
3. *the running time is $f(k) \cdot poly(|I|)$ for some computable function f.*

This is reminiscent of usual polynomial-time reductions, with an added constraint on the value of the parameter in the destination problem. This extra restriction guarantees the transfer of positive results between different parameterized problems:

Claim 5. *[CFK+15] Suppose we have a parameterized reduction from $(\mathcal{P}, k)$ to $(\mathcal{P}', k')$. If $(\mathcal{P}', k')$ is FPT (resp. XP) then $(\mathcal{P}, k)$ is also FPT (resp. XP).*

Kernelization Algorithms

Finally we introduce kernelization algorithms:

Definition 6. *A kernelization algorithm, or simply a kernel, for a parameterized problem $(\mathcal{P}, k)$ is an algorithm $\mathcal{A}$ that, given an instance (I, k), works in polynomial time and returns an equivalent instance (I', k') such that $|I'|$ and k' are both bounded by $g(k)$ for some computable function g.*

Then, given such a kernelization algorithm, one can solve parameterized problem $(\mathcal{P}, k)$ in time $poly(|I|) + g'(k)$ for some computable function g'. While this seems stronger than being FPT, it turns out that both notions are equivalent:

Claim 7. *[CFK+15] A parameterized problem $(\mathcal{P}, k)$ is FPT if and only if it admits a kernelization algorithm.*

Despite this, note that in practice kernels are usually harder to find than FPT algorithms.

2.2.2 Toolkit for Negative Results

Next we present the common ways to characterize that a parameterized problem is **not** tractable.

The para-NP Class

The para-NP Class has the following definition:

Definition 8. *A problem $\mathcal{P}$ is in class para-NP with respect to parameter k if the corresponding parameterized problem can be solved by a **nondeterministic** algorithm in time $f(k) \cdot |I|^c$ where f is a computable function and c is a constant independent of both parameter k and instance size $|I|$.*

As the name suggests, para-NP is the literal parameterized version of the NP class. In fact the following equivalences hold:

Claim 9. *[FG98]*

(*i*) *(FPT = para-NP) if and only if ($P = NP$).*

(*ii*) *A problem $\mathcal{P}$ is para-NP-hard parameterized by k if and only if it is NP-hard for a fixed value of k.*

The latter result motivates the notation "para-NP-hard($k = a$)" that we use to mention that a problem is NP-hard when k is fixed with a value a. It also has notable implications relatively to the XP class. Indeed proving a parameterized problem $(\mathcal{P}, k)$ to be para-NP-hard means that at least one slice of instances is NP-hard. Assuming ($P \neq NP$) this rules out any XP algorithm (and thus any FPT algorithm) for $(\mathcal{P}, k)$. However in practice a lot of parameterized problems are in XP while still not believed to be tractable (i.e. not FPT). This calls for more elaborated ways to track intractability.

The W-hierarchy

The W-hierarchy is defined via layers of restrictions of the WEIGHTED CIRCUIT SATISFIABILITY problem. All these layers are believed to be non-FPT and of (strictly) increasing difficulty. We direct the reader to [CFK$^+$15] for the full definition. Instead we rely on some commonly used complete problems for these classes.

WEIGHTED t-NORMALIZED SATISFIABILITY is a variant of the SAT problem which takes as input a t-normalized boolean formula and asks whether there exists an affectation of the variable with exactly k of them set to true which makes the formula true. t-normalized formulas are defined recursively. 0-normalized formulas are single literals. Then for $t \geq 1$, t-normalized formulas are a conjunction of disjunctions of an arbitrary number of $(t-1)$-normalized formulas. This problem is complete with respect to the corresponding level of the W-hierarchy:

Claim 10. *[FG98] For every integer $t \geq 2$, WEIGHTED t-NORMALIZED SATISFIABILITY is $W[t]$-complete.*

Note that parameterized reductions can be used to transfer negative results between different parameterized problems:

Claim 11. *[CFK+15] Suppose we have a parameterized reduction from $(\mathcal{P}, k)$ to $(\mathcal{P}', k')$. If $(\mathcal{P}, k)$ is para-NP-hard (resp. $W[t]$-hard for some t) then $(\mathcal{P}', k')$ is also para-NP-hard (resp. $W[t]$-hard for the same t).*

Below are the most commonly used complete problems in the parameterized reductions of this paper:

Claim 12. *[FG98, CFK+15]*

- *k-CLIQUE and k-INDEPENDENT SET are $W[1]$-complete.*
- *k-DOMINANT SET is $W[2]$-complete.*
- *k-COLORING is para-NP-complete.*

Then, assuming $FPT \neq W[1]$ - i.e. a stronger hypothesis than $(P \neq NP)$ - proving a parameterized problem to be $W[t]$-hard for some t rules out any FPT algorithm.

The $XNLP$ Class

We define the $XNLP$ class introduced by Elberfeld et al. in [EST15]:

Definition 13. *A problem $\mathcal{P}$ is in class $XNLP$ with respect to parameter k if the corresponding parameterized problem can be solved by a **nondeterministic** algorithm in time $f(k) \cdot |I|^c$ and space $g(k) \cdot log(|I|)$ where f, g are computable functions and c is a constant independent of both parameter k and instance size $|I|$.*

In [BGNS22] several parameterized problems (including a parallel machine scheduling one) were proved to be $XNLP$-complete. The next result sets $XNLP$ as a middle ground between the W-hierarchy and para-NP:

Claim 14. *[BGNS22]*

(*i*) $\forall t \in \mathbb{N}, W[t] \subseteq XNLP \subseteq$ *para-NP. In particular any $XNLP$-hard problem is $W[t]$-hard for all $t \in \mathbb{N}$.*

(*ii*) $XNLP \subseteq XP$.

Note that unlike with para-NP a parameterized problem can be $XNLP$-hard while still being in XP. And assuming $FPT \neq W[1]$, a problem being $XNLP$-hard rules out any FPT algorithm. Also note that with class $XNLP$ parameterized reductions require an additional space constraint.

Refined Run Time Lower Bounds

Now if we wish to go beyond parameterized class memberships and set sharper run time lower bounds, we need a stronger assumption.

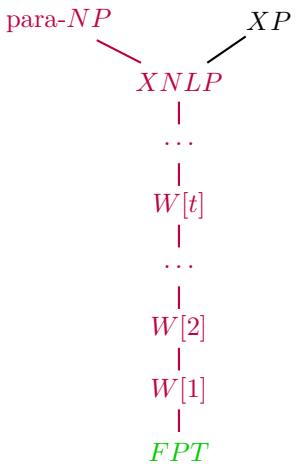


Figure 2.6: Known relations between parameterized complexity classes.

Definition 15 (Exponential Time Hypothesis (ETH))**.** *There is a positive real δ such that the 3-SAT problem with n variables and m clauses cannot be solved in time $2^{\delta n} \cdot (n+m)^{\mathcal{O}(1)}$.*

In other words such a hypothesis rules out any sub-exponential deterministic algorithm for 3-SAT and any other NP-hard problem.

Note that this is indeed stronger than the assumptions previously mentioned in this section.

Claim 16. *[CFK+15]*
$ETH \implies (FPT \neq W[1]) \implies (P \neq NP)$.

An assumption like ETH is often necessary to obtain lower bounds essentially matching the best known algorithms. In short the accuracy of our desired negative result must be in accordance to the strength of our complexity assumption.

Kernel Size Lower Bounds

While a kernelization algorithm operates in polynomial time of the instance size, the resulting kernel size can still be super-polynomial in the parameter. Thus a distinction can be made depending on the size of the resulting kernel.

Definition 17. *A polynomial kernel is a kernelization algorithm which outputs kernels of size polynomial in the parameter value of the original instance.*

In [BJK14] Bodlaender et al. proposed a technique called *cross-composition* in order to negate the existence of such polynomial kernels. In this work it is only used once, at the end of Chapter 4. The full definition will be given there. Assuming ($NP \not\subseteq coNP/poly$), cross-composition rules out any polynomial kernel for the given parameterized problem:

Claim 18. *[BJK14] Suppose that an NP-hard problem cross-composes into a parameterized problem $(\mathcal{P}, k)$. Then, assuming $(NP \not\subseteq coNP/poly)$, $(\mathcal{P}, k)$ admits no polynomial kernel.*

Note that as expected the required hypothesis is stronger than $(P \neq NP)$:

Claim 19. *[CFK+15]*
$(NP \not\subseteq coNP/poly) \implies (P \neq NP)$.

2.2.3 Problems on a Set Parameter and Parameters on a Set Problem

Throughout this document we support that parameterized complexity is inherently bi-dimensional, in that both the problem and parameter choice are equally important in the search for FPT algorithms. To foster this idea we propose two natural approaches: choosing a parameter then going over a range of problems, or setting a problem then considering various parameters for it.

In the first approach we set a parameter k then determine which problems are $FPT(k)$. We provide problem maps of our mainly studied parameters at the end of Chapters 3 to 6. We denote $(\mathcal{P} \to \mathcal{P}')$ if $\mathcal{P}$ is a subproblem of $\mathcal{P}'$. Such an edge is labeled by the properties added when going from $\mathcal{P}$ to $\mathcal{P}'$. A problem $\mathcal{P}$ is in a blue box if it can be solved in polynomial time. The box is green if $\mathcal{P}$ is $FPT(k)$ and red if $\mathcal{P}$ is non-FPT with respect to k - under some assumption like $P \neq NP$. Finally the box is orange if $\mathcal{P}$ is NP-hard and its parameterized complexity with respect to k is open. Such maps notably display the problem frontiers associated to a given parameter k. These help to emphasize which properties do not fit well with parameter k, and in turn find the settings in which this parameter is the most relevant.

In the second approach we set a problem $\mathcal{P}$ then determine for which parameters $\mathcal{P}$ is FPT. We introduce a natural way to relate different parameters on a set problem $\mathcal{P}$:

Definition 20. *Let $\mathcal{P}$ be a problem and let k, k' be two parameters.*

(*i*) *We say that k is weaker than k' on problem $\mathcal{P}$ if there is a parameterized reduction from $(\mathcal{P}, k')$ to $(\mathcal{P}, k)$. Alternatively we also say that k' is stronger than k on problem $\mathcal{P}$.*

(*ii*) *We say that k and k' are equivalent on problem $\mathcal{P}$ if k is weaker than k' and k' is weaker than k.*

This induces a partial order relation which can be determined and visualized for any problem $\mathcal{P}$. Throughout this document we provide such parameters maps for several scheduling problems. For example take Figure 2.7 which gives the parameter map of a parallel machine scheduling problem with precedence constraints and job time windows. We denote $(k \to k')$ if k is weaker than k' on problem $\mathcal{P}$. A parameter k is green if $\mathcal{P}$ is $FPT(k)$. Conversely k is red if $\mathcal{P}$ is non-FPT with respect to k - under some assumption like $P \neq NP$. Finally

k is black if the parameterized complexity of $\mathcal{P}$ with respect to k is open. Such maps notably help to visualize the parameter frontiers on a given problem $\mathcal{P}$ and find the weakest parameters k for which $\mathcal{P}$ is $FPT(k)$.

On top of evaluating the relative strength of each parameter on problem $\mathcal{P}$, such maps allow us to transfer complexity results easily between different parameters:

Claim 21. *Let $\mathcal{P}$ be a problem and let k, k' be two parameters.*

(*i*) *Suppose that k is weaker than k' on problem $\mathcal{P}$. If $(\mathcal{P}, k')$ is FPT (resp. XP) then $(\mathcal{P}, k)$ is also FPT (resp. XP). And if $(\mathcal{P}, k)$ is para-NP-hard (resp. $W[t]$-hard for some t) then $(\mathcal{P}, k')$ is also para-NP-hard (resp. $W[t]$-hard for the same t).*

(*ii*) *Suppose that k and k' are equivalent on problem $\mathcal{P}$. Then $(\mathcal{P}, k)$ is FPT (resp. XP, para-NP-hard, $W[t]$-hard for some t) if and only if $(\mathcal{P}, k')$ is FPT (resp. XP, para-NP-hard, $W[t]$-hard for the same t).*

Proof. Consequence of Claims 5 and 11. □

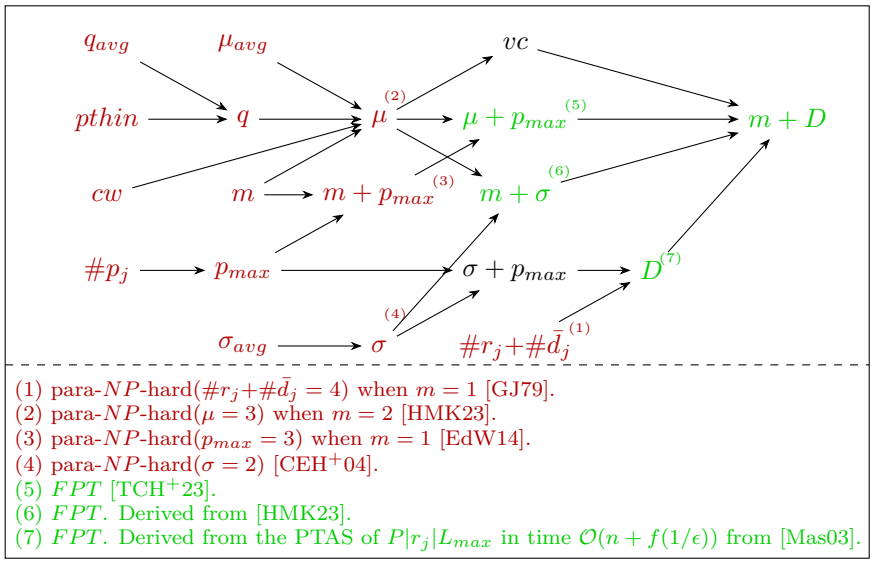


Figure 2.7: Parameter map of $P|r_j, \bar{d}_j|\star$.

This concludes the description of the parameterized complexity notions used in this work.

2.3 State of the Art

In this section we give an overview of the parameterized scheduling results obtained in the literature - except for the parameters that we discuss in the following chapters in detail. The state of the art concerning maximum delay value ℓ_{max}, pathwidth μ, slack σ and proper level q will be presented in the introduction of their respective chapter.

2.3.1 Scheduling with no Job Type Bounding

The first scheduling problem studied in the context of parameterized complexity was a parallel machine scheduling problem with unit-time jobs - denoted by $P|prec, p_j = 1|C_{max} < D$ in Graham's notation. The problem was shown to be NP-hard when $D = 3$ in 1978 by Lenstra et al. [LRK78] by Lenstra et al. and $W[2]$-hard parameterized by the number of machines m in [BF95] by Bodlaender and Fellows.

Dealing with precedence constraints, a natural parameter is the width w of the precedence graph, which measures the maximum number of incomparable jobs with respect to the partial order defined by the precedence constraints. However Bodlaender et al. recently proved this problem $XNLP$-complete parameterized by $m+w$ [BGNS22]. This gives no hope of finding a fixed-parameter tractable algorithm for this problem with these parameters.

In [vBBB$^+$16] van Bevern et al. considered some variants of this problem with respect to w. They showed that with two machines and processing times one and two, or with three machines and job sizes one and two - i.e. the number of parallel machines required to process the job -, the problem is still $W[2]$-hard(w). They also showed that $P2|chains|C_{max}$ is NP-hard with three chains (i.e. $w = 3$).

In response they introduced a non-structural parameter called the allowed lag λ, which is the maximum difference between the starting time of a job and its earliest possible starting time according to precedence constraints. While problem $P|prec, p_j = 1|C_{max}$ is para-NP-hard($\lambda = 1$) from the same reduction as in [LRK78], a FPT algorithm for RCPSP is proposed with parameter $w + \lambda$. This could infer that in RCPSP the more difficult part is related to precedence constraints and time windows rather than the complexity of resource constraints.

When there are time windows and no precedence relations scheduling problems remain difficult when jobs are not unit-time. For example in [EdW14] Elffers and de Weerdt proved that $1|p_j \in \{p, q\}, r_j, \bar{d}_j|\star$ is strongly NP-hard for any $p > q > 1$ fixed (e.g. $p = 3$ and $q = 2$).

In the unrelated parallel machine setting Lenstra et al. showed that job time windows are not even needed to make problem $R||C_{max} < D$ para-NP-hard($p_{max} = 3$) and para-NP-hard($D = 2$) [LST90]. Even though this problem becomes $FPT(p_{max})$ when the parallel machines are identical, it can be shown $W[1]$-hard(m) even when job processing times are given in unary. This is a corollary of [JKMS13] where Jansen et al. showed that UNARY BIN PACKING

is $W[1]$-hard parameterized by the number of bins. This result also implies that $P|p_j = 1, size_j|C_{max} < D$ is $W[1]$-hard parameterized by D.

More recently in [CJZ18] Chen et al. gave several lower bounds for exact and approximate resolution of $P||C_{max}$. They showed that under ETH the existing exact and approximation algorithms for $P||C_{max}$ are essentially the best possible For example while there is an exact algorithm for $Pm||C_{max}$ in time $2^{\mathcal{O}(m\cdot\sqrt{|I|})}$ by O'Neil [O'N11], they showed that for any positive real δ there is no exact algorithm for this problem in time $2^{\mathcal{O}(m^{1/2-\delta}\cdot\sqrt{|I|})}$ (unless ETH fails).

When delays are added on top of precedence relations there is little hope to find a FPT algorithm without bounding the delay values. In [BvdW20] Bodlaender and van der Wegen considered chains of unit-time jobs with chain time windows and exact or minimum precedence delays given in unary. They studied two parameters: the number of chains c - which is equivalent to width w - and thickness th which is the maximum number of overlapping chain time windows (note that $th \leq c$). On a single machine we denote this problem $1|chains(\ell_{i,j}), p_j = 1, r_C, d_C|\star$. With exact delays although it is $W[t]$-hard(th) for all t and $W[2]$-hard(c) they showed that it is $XP(c)$ even when delays are given in binary. On parallel identical machines with exact or minimum delays they showed that the problem is both $W[2]$-hard(c) and $XP(th)$.

2.3.2 Partial Job Type Bounding

Recent approaches suggest making use of general frameworks based on mixed-integer programming (MIP). If a scheduling problem can be translated into a mixed-integer linear (resp. convex) program with a number of integer variables bounded by some parameter, then a result by Lenstra [LJ83] (resp. by Dadush et al. [DPV11]) guarantees the existence of a FPT algorithm with respect to this parameter. This can be typically achieved when the number of distinct properties between the jobs is limited in some way, for example by bounding the maximum processing time p_{max} or the number of distinct processing times $\#p_j$. This was first done by Mnich and Wiese who showed that $P||C_{max}$ is $FPT(p_{max})$ and $R||C_{max}$ is $FPT(m + \#p_j)$ [MW15].

On a single machine Mnich and Wiese also studied two problems with job rejection in the same paper. First we have a problem with preemption denoted by $1|pmtn, r_j, reject|max(\sum_{j \text{ selected}} f_j(C_j))$ where f_j is a non-increasing profit function for job j. The considered parameters were the number of selected jobs and $\#p_j$. While they showed that this problem is respectively $W[1]$-hard and para-NP-hard with respect to each parameter individually, they gave a FPT algorithm when both are combined.

Second they analyzed problem $1|reject|(\sum_{j \text{ rejected}} e_j + \sum_j w_j C_j)$ where e_j is the rejection cost of job j. The considered parameters were the number of rejected jobs, $\#p_j$ and the number of distinct weights $\#w_j$. While this problem is proved $W[1]$-hard parameterized by the number of rejected jobs, they showed that it is FPT with respect to any pair of these three parameters.

Hermelin et al. obtained similar results on problem $1||\sum w_j C_j$ with param-

eters $\#d_j, \#p_j$ and $\#w_j$ [HKPS21]. While this problem was proved para-NP-hard($\#d = 1$) by Karp [Kar72], $W[1]$-hard($\#p_j$) and $W[1]$-hard($\#w_j$) by Heeger and Hermelin [HH24], Hermelin et al. showed that it is FPT with respect to any pair of these three parameters.

On multiple machines several frameworks were proposed to generalize the existing approaches. In the unrelated parallel machine setting Chen et al. proposed the rank ρ of the processing time matrix as a parameter [CMYZ17]. They noted that case $\rho = 1$ corresponds to the uniform machine environment Q, where each machine i has a speed s_i. So parameter ρ serves as a possible generalization to the unrelated parallel machine environment R. While $R||C_{max}$ was shown the be APX-hard even when $\rho = 3$ Chen et al. proposed a FPT algorithm parameterized by $(p_{max} + \rho)$ in time $2^{2^{\mathcal{O}(\rho \cdot log(p_{max}))}} + n^{\mathcal{O}(1)}$. They also showed that under ETH there is no algorithm for $R||C_{max}$ in time $2^{2^{o(\rho \cdot log(p_{max}))}}$.

In [KZ20] Kouteckỳ and Zink considered a couple other objective functions and the number of job types d as an extra parameter. They showed that $R||X$ with $X \in \{C_{max}, \ell_2 - norm, \sum w_j C_j\}$ is $W[1]$-hard(d) even when $\rho = 2$ and processing times are given in unary. In machine environment Q they proved that the problem remains $W[1]$-hard(d) on objectives C_{max} and ℓ_2, even when processing times and machine speeds are given in unary. In contrast a FPT algorithm with respect to d is proposed in the identical parallel machine setting.

On another note the N-fold integer programming framework introduced in [KK18b] by Knop and Kouteckỳ has shown great success. Given $n, r, s, t \in \mathbb{N}$ an N-fold IP is of the form $\max\{c^T x, \mathcal{A}x = b, \ell \leq x \leq u, x \in \mathbb{Z}^{nt}\}$ where $\mathcal{A} \in \mathbb{Z}^{(r+sn)\times nt}$ consists of n arbitrary matrices $A^{(i)} \in \mathbb{Z}^{r\times t}$ on a horizontal line, and n arbitrary matrices $B^{(j)} \in \mathbb{Z}^{s\times t}$ on a diagonal line.

The authors were able to obtain multiple FPT results on machine environments Q, R and objectives $C_{max}, \sum w_j C_j$. First they showed that $Q||C_{max}$ is $FPT(p_{max})$ and $R||C_{max}$ is $FPT(p_{max} + \kappa)$. Then the latter result was adapted to the weighted objective with parameters $(p_{max} + w_{max} + \kappa)$ and $(\#p_j + \#w_j + m)$ successively. They complete their analysis by showing that bounding the number of machines alone is unlikely to lead to a FPT algorithm. This was done by proving both $P||C_{max}$ and $P||\sum w_j C_j$ to be $W[1]$-hard(m), even when processing times and weights are given in unary.

The same problems were studied more recently by Fisher et al. in [FGM22]. They proposed several run time improvements on the makespan objective and, under ETH or ($P \neq NP$), gave lower bounds for $Q||C_{max}$. They also provided $FPT(d+p_{max})$ algorithms for multiple variants - namely $Q|r_j|C_{max}$, $Q|\bar{d}_j|C_{max}$, $Q||C_{min}$ and $Q||\sum w_j C_j$.

2.3.3 The High-Multiplicity Setting

The high-multiplicity setting is a continuation of the job type bounding approaches presented in the previous section. Instead of describing the properties of the n jobs (resp. m machines) individually in the input, we characterize d job types (resp. κ machine types) and give the number of elements associ-

ated to each type. Then the size of the instance mainly depends on the number of job types rather the total number of jobs. In parameterized scheduling this setting was considered for the first time by Brauner et al. on problem $1|HM(n), FSE|C_{max}$, where FSE stands for "Forbidden Start and End instants" in [GRB16]. They showed that this problem is FPT parameterized by the number of forbidden instants.

Recent work studied connections with the CUTTING STOCK problem, which is the high-multiplicity version of the BIN PACKING problem. With this analogy Goemans and Rothvoß considered problem $R|HM(m,n), r_j^{(i)}, \bar{d}_j^{(i)}, c^{(i)}|X$ where $c^{(i)}$ is the cost for using a machine of type i and objective X aims at minimizing the total machine cost. They showed that this problem is $XP(d + \kappa)$ [GR20]. On more traditional problems Koutecký and Zink used a BIN PACKING variant to obtain several para-NP-hardness results with parameter $d = 7$. The concerned problems are $Q|HM(n)|C_{max}$, $Q|HM(n)|\ell_2$ and $R|HM(n)|\sum w_jC_j$. When processing times are given in unary they showed that $R|HM(n)|X$ is $XP(d)$ for all three objectives $X \in \{C_{max}, \ell_2, \sum w_jC_j\}$ [KZ20].

In [KKL+19] Knop et al. proposed a high-multiplicity variant of N-fold integer programming, which they called the Multitype Integer Monoid Optimization (MIMO) framework. They separated scheduling objectives into two kinds $\mathcal{C}_{lin}$ and $\mathcal{C}_{poly}$ where $\mathcal{C}_{lin} = \{C_{max}, C_{min}, F_{max}, L_{max}, \sum w_jU_j\}$ and $\mathcal{C}_{poly} = \{\sum w_jC_j, \sum w_jF_j, \sum w_jT_j, \ell_p^C\text{-norm}\}$. They showed that any scheduling problem expressible in the MIMO framework is FPT parameterized by $(d + p_{max})$ and $(d + m)$ with a single-exponential dependency on the parameter for any objective from $\mathcal{C}_{lin} \cup \mathcal{C}_{poly}$ and, if processing times are given in unary, FPT parameterized by $(d + \kappa)$ with a double-exponential dependency on the parameter for any objective from $\mathcal{C}_{lin}$. As an example they successfully applied their method to a problem on unrelated machines with time windows - denoted by $R|HM(m,n), r_j^{(i)}, \bar{d}_j^{(i)}|X$ where X is any objective in $\mathcal{C}_{lin} \cup \mathcal{C}_{poly}$.

Since then several run time improvements were proposed on specific problems under the high-multiplicity setting. In [BJ22] Brinkop and Jansen considered high multiplicity scheduling on uniform machines with three objectives: C_{max}, C_{min} and C_{envy} which aims at minimizing the difference between the maximum completion time and the minimum completion time. They gave an algorithm on objectives C_{max} and C_{min} which runs in time $(p_{max})^{\mathcal{O}(d^2)} \cdot poly(|I|)$. Then they showed that with objective C_{envy} the problem is FPT when maximum speed s_{max} is combined with d and p_{max} as a parameter. They also investigated the restricted assignment setting, which can be written as $R|HM(m,n), p_j^{(i)} \in \{p_j, \infty\}|X$. On objectives C_{max} and C_{min} they proposed a FPT algorithm with run time $(d \cdot p_{max})^{\mathcal{O}(d^3)} \cdot poly(|I|)$.

More recently in [JKZ24] Jansen et al. went back to problem $Q|HM(m,n)|X$ with the same three objectives C_{max}, C_{min} and C_{envy}. They gave several run time improvements with parameter $(d+p_{max})$ for all three objectives and proved multiple lower bounds on objective C_{max} under ETH. They also proposed an approximation scheme with additive error at most ϵp_{max} in time $(mp_{max})^{\mathcal{O}(\frac{1}{\epsilon})}$

and gave a matching lower bound under ETH. Finally while it is still unknown whether makespan minimization in high-multiplicity scheduling on identical parallel machines is $FPT(d)$, they showed that it is equivalent to whether high-multiplicity scheduling on uniform machines is $FPT(d + \kappa)$.

2.4 Concluding Remarks

This concludes the parameterized state of art of RCPSP and this opening chapter. We began by introducing RCPSP and its known classical complexity results, both with and without job time windows and/or precedence delays. Then we presented the parameterized complexity notions which are used throughout this work. Finally we summarized the current state of parameterized complexity in scheduling.

While scheduling and parameterized complexity have their respective well-established community, the intersection between the two has been developing only recently. As a result several scheduling frameworks are left to be explored in the context of parameterized complexity theory. Indeed while numerous FPT algorithms were found in high-multiplicity scheduling, settings with (little to) no job type bounding have been trailing behind, especially when it comes to investigating new structural parameters. We intend to fill this gap for scheduling problems with job time windows and/or precedence delays.

Chapter 3

Results with Maximum Delay Value ℓ_{max}

3.1 Introduction

In this chapter we investigate the inclusion of precedence delays in several subproblems of RCPSP. Such delays specify a time value on top of existing precedence relations. Given two jobs i, j and a precedence constraint $i \to j$ forcing j to be started after i is completed, we can add a minimum - resp. exact, maximum - delay ℓ to ask that j must be started at least - resp. exactly, at most - ℓ time units after i is completed.

Precedence delays are a special case of the time lags defined by Brucker et al. in [BHH99]. Indeed time lags are not necessarily tied to a precedence relation and thus are more general. Other connex notions include communications delays - for which the delay is only applied when i and j are scheduled on different machines - and sequence-dependent setup times, the delay values of which depend on the order of the jobs (see for example [BdWH21] by Baart et al.).

Among the three precedence delay types studied in this work the exact ones have been the most explored, especially in the context of coupled task scheduling (i.e. when *prec* is only composed of isolated edges). This is not surprising considering that the parameterized problem is already difficult when the precedence graph is only composed of chains (see Section 3.2). The coupled task properties are usually denoted by a triple (a_j, ℓ_j, b_j) where a_j (resp. b_j) is the processing time of the first task (resp. second task) and ℓ_j is the precedence delay in between. Explanations for the notations of the particular cases are available in Annex A.2. A comprehensive overview of the classical complexity results in the literature was given by Khatami et al. in [KSC20] and Chen and Zhang in [CZ21]. In [Bap10] Baptiste showed that $1|(p, \ell, p)|C_{max}$ can be solved in time $\mathcal{O}(f(\ell) \cdot log(n))$, lowering the dependency in the input size at the expense of an exponential blowout with respect to ℓ. Then in [KOS23] Khatami et al. extended the method to show that $Pm|(a, \ell, b)|C_{max}$ is FPT

parameterized by ℓ. In [BG19] Bessy and Giroudeau explored the inclusion of a compatibility graph G_c which dictates when two coupled tasks are allowed to interleave. They showed that $1|(a_j, \ell_j, b_j), G_c|C_{max} < D$ is NP-hard even when G_c is a star, and that it is FPT parameterized by $max_j(a_j + \ell_j + b_j)$ plus the vertex cover of G_c. When minimizing the number of tardy jobs they showed that $1|(1, \ell, 1), G_c| \sum U_j$ is $W[1]$-hard parameterized by $\sum U_j$ but becomes FPT when ℓ_{max} is fixed.

Minimum precedence delays allow more problems to be solved efficiently. While scheduling unit-time jobs on a single machine was proved strongly NP-hard on general precedence relations with a single delay value by Leung et al. [LVW84], Bruno et al. showed that the problem can be solved in polynomial time when restricting *prec* to an inforest and adding deadlines [BJS80] (or restricting *prec* to an outforest and adding release dates). Still in [Eng00] Engels proved that variants like $1|chains(0, \ell), p_j = 1|C_{max} < D$ and $1|chains(\ell), p_j \in \{1, 2\}|C_{max} < D$ remain difficult to solve - respectively strongly NP-hard and para-NP-hard with respect to ℓ (with $\ell = 2$).

Maximum precedence delays have been the least studied precedence delay type. However it is worth noting that $1|prec(\ell), p_j = 1|C_{max} < D$ equivalent to the DIRECTED BANDWIDTH graph problem which was first studied by Garey et al. [GGJK78]. They showed that the problem is strongly NP-hard with trees of indegree one and outdegree at most two. In the case of polytrees - i.e. directed acyclic graphs whose underlying undirected graph is a tree - Bodlaender showed that the problem is $W[t]$-hard for all $t \in \mathbb{N}$ when the underlying undirected graph is a caterpillar with hair length at most one [Bod21]. While k-BANDWIDTH was recently proved $XNLP$-complete by [BGNS22], it is open whether k-DIRECTED BANDWIDTH is also $XNLP$-complete.

While this work focuses on job time windows, in [BvdW20] Bodlaender and van der Wegen obtained several interesting results when time windows are defined on the precedence chains instead. With exact or minimum precedence delays they showed that $1|chains(\ell_{i,j}), p_j = 1, r_C, \bar{d}_C|\star$ is NP-hard even when delays are given in unary. In the parallel-machine setting with delays given in unary they also gave an XP algorithm parameterized by thickness th - i.e. the maximum number of chain time windows which can include a time unit.

The vast majority of these results suggests that restricting the delays is often not enough in itself to make the problem easy enough. The results presented in this chapter will reinforce this intuition in the context of job time windows - or with no time windows at all.

This chapter is organized as follows. In Section 3.2 we set several results with exact and maximum delays, some of them being closely related to bin packing and bandwidth results in the literature. Then in Section 3.3 we focus on minimum precedence delays with unit-time jobs and prove multiple negative results. Finally in Section 3.4 we summarize the results obtained in this chapter and give our concluding remarks.

3.2 Results with Exact and Maximum delays

In this section we show a number of results with exact and maximum delays. Most of them are inferred from the literature in a straightforward way.

First we adapt the classical reduction idea from strongly NP-hard problem 3-PARTITION, which was first proposed in [GJ79]. Fill jobs delimited the partitions and each integer was represented by a job of matching processing time. Here we reduce from UNARY 3-PARTITION and represent each integer by a chain of unit-time jobs of matching length.

Claim 22. *With exact or maximum delays* $1|chains(\ell), p_j = 1, r_j, \bar{d}_j|\star$ *is para-NP-hard parameterized by* ℓ *(with* $\ell = 0$*) even with chains given in unary.*

Proof. We reduce from UNARY 3-PARTITION. Given number of partitions B, target sum T and positive integers $(a_j)_{1\leq j\leq 3B}$ all given in unary we proceed in a similar fashion as in [BvdW20] (where time windows were defined on the chains instead of the jobs individually). We set $D = B(T+1)-1$ and a fill job at time units $i(T+1)$, $1 \leq i \leq B-1$. The fill jobs leave exactly B intervals of length T to schedule the other jobs. Then each integer a_j is represented by a chain of unit-time jobs of length a_j with delay 0 everywhere. The i^{th} job in the chain has release date $(i-1)$ and deadline $(D-a_j+i)$. Then the equivalence between this scheduling instance and the UNARY 3-PARTITION instance is straightforward. □

In the case of exact delays the parameterized problem remains difficult in the absence of job deadlines or release dates.

Claim 23. *With exact delays* $1|chains(\ell), p_j = 1|C_{max} < D$ *is* $W[1]$*-hard parameterized by* ℓ *even with chains given in unary.*

Proof. We reduce from the k-UNARY BIN PACKING problem, which was proved $W[1]$-hard in [JKMS13]. Given bin capacity T, positive integers $(a_j)_{1\leq j\leq n}$ given in unary and parameter k - i.e. the number of bins - we set $D = kT$ and $\ell = k-1$. Each integer a_j is represented by a chain of unit-time jobs of length a_j with delay ℓ everywhere. Then all jobs from a chain have the same time position modulo k. So the i^{th} bin is represented by all time positions modulo k within interval $[0, D-1]$. Then the equivalence between our scheduling instance and the k-BIN PACKING instance is straightforward. □

Note that this is not the case with maximum delays. In fact having chains with arbitrary delay values does not hinder polynomiality.

Claim 24. *With maximum delays* $1|chains(\ell_{i,j}), p_j = 1|C_{max} < D$ *can be solved in time* $\mathcal{O}(n)$*.*

Proof. Chains can be scheduled one after the other with delay 0 everywhere in the schedule. So one can simply add up the chain lengths then check if this sum is lower than or equal to D. □

However adding either job deadlines or release dates is enough to reach NP-hardness when at least two delay values are available.

Claim 25. *With maximum delays* $1|chains(0,\ell), p_j = 1, r_j|C_{max} < D$ *and* $1|chains(0,\ell), p_j = 1, \bar{d}_j|\star$ *are* NP*-hard.*

Proof. We consider the problem with release dates first. We reduce from UNARY 3-PARTITION. Given number of partitions B, target sum T and positive integers $(a_j)_{1\leq j\leq 3B}$ all given in unary we set $D = B(T+1)$ and a chain of fill jobs $(f_i)_{1\leq i\leq B}$ with delay ℓ everywhere. The i^{th} fill job f_i has release date $(i-1)(T+1)$, $1 \leq i \leq B$. Then each integer a_j is represented by a chain of unit-time jobs of length a_j with delay 0 everywhere. The i^{th} job in the chain has release date i.

We show that in every feasible schedule all fill jobs f_i are scheduled at their release date $(i-1)(T+1)$. We start with f_1. By contradiction suppose f_1 was scheduled later than time 0 in some feasible schedule. Then all other jobs have a release date greater than 0. So we would have $B(T+1)$ jobs scheduled in $B(T+1)-1$ time positions, which contradicts the feasibility of the schedule. Thus f_1 must be scheduled at its release date, and so is every fill job f_i by induction on i according to the maximum delays. This leaves exactly B intervals of length T to schedule the other jobs. Then the equivalence between this scheduling instance and the UNARY 3-PARTITION instance is straightforward.

The problem with deadlines is proved NP-hard the same way by setting each job deadline to D minus the release date it was given in the reduction above. □

The parameterized complexity of problems $1|chains(0,\ell), p_j = 1, r_j|C_{max} < D$ and $1|chains(0,\ell), p_j = 1, \bar{d}_j|\star$ is open, and so is their classical complexity when restricted to a single delay value.

3.3 Hardness Results with Minimum Delays

In this section we prove several hardness results with minimum delays. In Subsection 3.3.1 we show that single machine scheduling with general precedence and equal-length minimum precedence delays is hard even with bounded ℓ_{max} and bounded precedence graph width w. Next in Subsection 3.3.2 we adapt this reduction to the special case of precedence chains when extra minimum precedence delays of length zero are allowed. Finally in Subsection 3.3.3 we strengthen the latter result in the identical parallel machine setting.

3.3.1 On a Single Machine With General Precedence

In this subsection we consider single machine scheduling of unit jobs with general precedence and equal delays of length ℓ. We prove the following negative result:

Theorem 26. *$1|prec(\ell), p_j = 1|C_{max} < D$ with minimum delays is $XNLP$-hard parameterized by $\ell + w$.*

We reduce from $P|prec, p_j = 1|C_{max} < D$ parameterized by $m + w$, which was proved $XNLP$-complete in [BGNS22]. Let $I = \langle \mathcal{J}, prec, m, D \rangle$ be an instance of $P|prec, p_j = 1|C_{max} < D$ with $\mathcal{J} = (J_j)_{1 \leq j \leq |\mathcal{J}|}$. We build an instance $I' = \langle \mathcal{J}', prec', D' \rangle$ of $1|prec(\ell), p_j = 1|C_{max} < D$ which will be feasible if and only if I is feasible.

We propose to linearize the parallel machine instance. Each time unit in the parallel machine setting is represented as a time segment with length the number of available machines. By setting minimum delay value ℓ as the number of machines, this guarantees that two jobs J_i, J_j related by precedence cannot be scheduled in the same time segment of length ℓ - i.e. not at the same time in the parallel machine setting.

Now this alone would not represent a parallel machine setting faithfully. Indeed if $J_i \to J_j$ in the parallel machine setting then J_j can be scheduled on any machine in the time unit after the completion of job J_i. If the time segments are too close from each other in our single machine representation then the precedence delays could forbid some time units in the next time segment. One way to fix this is to separate the time segments by at least m time units and add fill jobs in between.

Definition 27.
Given $I = \langle \mathcal{J}, prec, m, D \rangle$ an instance of $P|prec, p_j = 1|C_{max} < D$ we define $I' = \langle \mathcal{J}', prec', D' \rangle$ instance of $1|prec(\ell), p_j = 1|C_{max} < D$. We set $\ell = m$ and $D' = (2D-1) \cdot (m+1) + 1$. We define the set of jobs $\mathcal{J}'$ as $\mathcal{J}$ plus the following jobs:

- *skeleton jobs a_i, $0 \leq i \leq 2D - 1$,*
- *fill jobs $b_{i,j}$, $0 \leq i \leq D - 2$, $0 \leq j \leq m - 1$.*

We set precedence graph $prec'$ as $prec$ with minimum delay ℓ on every precedence relation, plus the following relations:

- $a_i \xrightarrow{\geq \ell} a_{i+1}$, $0 \leq i \leq 2D - 1$,
- $a_{2i} \xrightarrow{\geq \ell} b_{i,j} \xrightarrow{\geq \ell} a_{2i+3}$, $0 \leq i \leq D - 2$, $0 \leq j \leq m - 1$.

Note that among the added jobs you cannot find more than $(m+1)$ of them which are not related to each other by $prec'$. So the width w' of precedence graph $prec'$ is bounded by the width w of $prec$ plus $(m + 1)$.

Lemma 28. *In any feasible schedule τ' of I':*

(*i*) $\tau'(a_i) = i \cdot (m + 1)$ *for all* $0 \leq i \leq 2D - 1$,

(*ii*) $(2i + 1) \cdot (m + 1) < \tau'(b_{i,j}) < (2i + 2) \cdot (m + 1)$ *for all* $0 \leq i \leq D - 2$ *and* $0 \leq j \leq m - 1$.

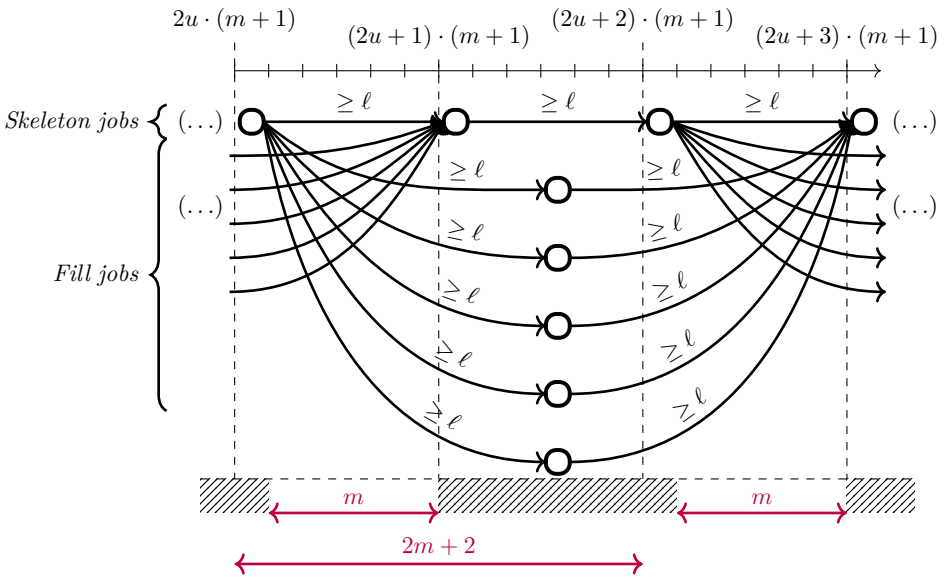


Figure 3.1: Illustration of skeleton jobs and fill jobs.

Proof. (i) Let $0 \leq i \leq 2D-1$. If job a_i is scheduled earlier than $i \cdot (m+1)$ in τ' then job a_0 would have to be scheduled at a time earlier than 0, which is not possible. And if a_i is scheduled later than $i \cdot (m+1)$ then job a_{2D-1} would have to be scheduled at a time later than $2D \cdot (m+1) - m$, i.e. at deadline D' or later. So necessarily $\tau'(a_i) = i \cdot (m+1)$.

(ii) Let $0 \leq i \leq D-2$ and consider fill job $b_{i,j}$ for any j. The result is directly inferred from point (i) and the fact that a_{2i} (resp. a_{2i+3}) is a predecessor (resp. successor) of $b_{i,j}$. □

Proposition 29. *Instance I is feasible if and only if instance I' is feasible.*

Proof. Let us name the machines from 1 to m arbitrarily.

($\Longleftarrow$) Let τ' be a feasible schedule of I'. By Lemma 28 skeleton jobs and fill jobs fully occupy times 0, $D'-1$ and time intervals of the form $[(2i+1) \cdot (m+1), (2i+2) \cdot (m+1)]$ with $0 \leq i \leq D-2$. So given a job J_j from $\mathcal{J}$, $\tau'(j)$ is necessarily of the form $2i \cdot (m+1) + k$ with $0 \leq i \leq D-1$ and $1 \leq k \leq m$. We propose schedule τ on instance I where job J_j would then be scheduled at time i on machine k. Since τ' is feasible this guarantees that there is at most one job per couple (time unit, machine). Plus if we had $J_i \xrightarrow{\geq \ell} J_j$ in I' then, in schedule τ', J_j was scheduled at a later time segment of length m than J_i. So, in schedule τ, J_j is scheduled at a later date than J_i and precedence relation $J_i \rightarrow J_j$ is met. Thus τ is feasible.

($\Longrightarrow$) Let τ be a feasible schedule of I. We propose schedule τ' defined the following way:

- for every job J_j in $\mathcal{J}$ if it was scheduled on machine i then we set $\tau'(j) = 2\tau(j) \cdot (m+1) + i$,
- $\tau'(a_i) = i \cdot (m+1)$ for all $0 \leq i \leq 2D-1$,
- $\tau'(b_{i,j}) = (2i+1) \cdot (m+1) + (j+1)$ for all $0 \leq i \leq D-2$ and $0 \leq j \leq m-1$.

We show that τ' is feasible. First note that skeleton jobs and fill jobs do not interfere with each other or with the jobs from $\mathcal{J}$. Now since τ is feasible there is at most one job J_j per couple (time unit, machine). Plus every job J_j is scheduled earlier than D in τ, so it is scheduled earlier than $2(D-1) \cdot (m+1) + m + 1 = D' - 1$ in τ'.

Now only precedence relations are left. The concerned skeletons jobs and fill jobs are placed at least $m + 1 = \ell + 1$ time units from each other, so no issue with them. Now given two jobs J_i, J_j from $\mathcal{J}$ such that $J_i \to J_j$ they are respectively scheduled in time segments $[2\tau(i) \cdot (m+1) + 1, 2\tau(i) \cdot (m+1) + m]$ and $[2\tau(j) \cdot (m+1) + 1, 2\tau(j) \cdot (m+1) + m]$. So their relative position is preserved and they are at least $m+2 = \ell+2$ time units from each other. So the precedence constraint - minimum delay included - is fulfilled. Thus τ' is feasible. □

This concludes the $XNLP$-hardness proof of the corresponding parameterized problem.

3.3.2 On a Single Machine with Chains

In this subsection we restrict to chains of precedence and add delays of value 0 as an option. We show that this parameterized problem remains hard:

Theorem 30. *With minimum delays $1|chains(0,\ell), p_j = 1|C_{max} < D$ is W[2]-hard parameterized by $\ell + \#chains$.*

We reduce from the instances of $P|prec, p_j = 1|C_{max} < D$ which were used in [BF95] to prove that the problem is W[2]-hard parameterized by m - via a reduction from k-DOMINATING SET. Given $G = (V, E)$ they gave I an instance of $P|prec, p_j = 1|C_{max} < D$ which is feasible if and only there exists a dominating set of size at most k in G. Below is the definition of instance I:

Definition 31. *[BF95] Given $G = (V, E)$ we define $I = \langle \mathcal{J}, prec, m, D \rangle$ instance of $P|prec, p_j = 1|C_{max} < D$ the following way. Let $n = |V|$ and $c = n^2 + 1$. We set the number of machines $m = 2k + 1$ and deadline $D = (kn) \cdot c + 2n$. The set $\mathcal{J}$ of jobs contains:*

- *floor jobs a_i, $1 \leq i \leq D$,*
- *floor gadgets b_j for any j of the form $n - 1 + \alpha c + in$ with $1 \leq i \leq n$ and $0 \leq \alpha \leq kn - 1$,*
- *selector path jobs $c_{i,j}$, $1 \leq i \leq k$, $1 \leq j \leq D - n$,*

- *selector gadgets* $d_{i,j}$, $1 \leq i \leq k$ *for any* j *of the form* $n-1+\alpha c+i'n-j'$ *with* $1 \leq i' < j' \leq n$ *such that* $\{v_{i'}, v_{j'}\} \notin E$.

We define precedence graph prec as the following relations:

- $a_i \rightarrow a_{i+1}$, $1 \leq i \leq D-1$,
- $a_{j-1} \rightarrow b_j \rightarrow a_{j+1}$ *for every floor gadget* b_j,
- $c_{i,j} \rightarrow c_{i,j+1}$, $1 \leq i \leq k$, $1 \leq j \leq D-n-1$,
- $c_{i,j-1} \rightarrow d_{i,j} \rightarrow c_{i,j+1}$ *for every selector gadget* $d_{i,j}$.

Proposition 32. *[BF95] I is feasible if and only there exists a dominating set of size at most k in G.*

We build I' an instance of $1|chains(0,\ell), p_j = 1|C_{max} < D$ which will be feasible if and only if I is feasible. As in Section 3.3.1 we represent the parallel-machine by a series of time intervals corresponding to the number of available machines at each time unit, plus enough consecutive fill jobs in between them to convert precedence delays as "jump to later time intervals" constraints and nothing more. Note that in instance I, without the floor and selector gadgets the precedence graph would be a union of chains. So we show how to replicate such gadgets with chains and two distinct delay values - 0 and $\ell = m$.

On the one hand the purpose of floor gadgets is to have $m-2$ available machines at specific time units instead of $m-1$ everywhere else. So we propose to replicate this effect with two chains. First as in Section 3.3.1 a chain of "skeleton" jobs will help delimit the time intervals. Then a second chain will cover every other time interval with fill jobs. We highlight the time units with $m-2$ available machines in I:

Definition 33. *Let* $\beta \in [0, D-1]$. *We say that* β *is critical if it is of the form* $n-1+\alpha c+in$ *with* $1 \leq i \leq n$ *and* $0 \leq \alpha \leq kn-1$.

Figure 3.2 (resp. 3.3) illustrates how a noncritical (resp. critical) time unit in instance I is simulated in instance I'.

On the other hand selector gadgets will simply be integrated to their respective selector path with a delay of length ℓ on one side and one of length 0 on the other side. Then a selector gadget and its corresponding selector path can be scheduled in the same time interval in instance I' while nearly keeping the same selector path neighbors in the precedence relations. This will be enough to simulate accurately their scheduling possibilities in instance I.

Instance I' is defined the following way:

Definition 34. *Given* $G = (V, E)$ *and instance* $I = \langle \mathcal{J}, prec, m, D\rangle$ *defined in Definition 31 we define* $I' = \langle \mathcal{J}', prec', \ell, D'\rangle$ *instance of* $1|chains(0,\ell), p_j = 1|C_{max} < D$ *the following way. Let* $n = |V|$ *and* $c = n^2+1$. *Let* $m = 2k+1$ *and* $D = (kn)\cdot c + 2n$. *We set* $\ell = m$ *and* $D' = (2m+2)\cdot D + kn^2 + 1$. *The set* $\mathcal{J}'$ *of jobs contains:*

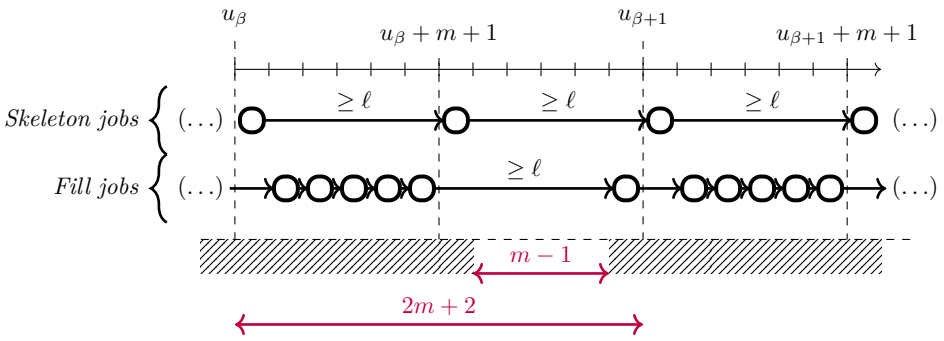


Figure 3.2: Illustration of skeleton jobs and fill jobs in the case of a **noncritical** time unit β. The precedence relations with no label have minimum delay 0.

- *skeleton jobs* $e_{2\beta}, e_{2\beta+1}$, $0 \leq \beta \leq D-1$ *plus an extra job* $e'_{2\beta+1}$ *if* β *is critical, and a final skeleton job* e_{2D},
- *fill jobs* $f_{\beta,j}$, $0 \leq \beta \leq D-1$, $0 \leq j \leq m$ *plus an extra job* $f'_{\beta,m}$ *if* β *is critical,*
- *selector path jobs* $c'_{i,j}$ *and selector gadgets* $d'_{i,j}$ *replicas of the jobs* $c_{i,j}, d_{i,j}$ *given in Definition 31.*

We define precedence graph prec′ as the following relations:

- $\left\{ \begin{array}{ll} e_{2\beta} \xrightarrow{\geq \ell} e'_{2\beta+1} \xrightarrow{\geq 0} e_{2\beta+1} \xrightarrow{\geq \ell} e_{2\beta+2} & \text{if } \beta \text{ is critical,} \\ e_{2\beta} \xrightarrow{\geq \ell} e_{2\beta+1} \xrightarrow{\geq \ell} e_{2\beta+2} & \text{otherwise} \end{array} \right\}, 0 \leq \beta \leq D-1,$
- $f_{\beta,j} \xrightarrow{\geq 0} f_{\beta,j+1}$, $0 \leq \beta \leq D-1$, $0 \leq j \leq m-2$,
- $\left\{ \begin{array}{ll} f_{\beta,m-1} \xrightarrow{\geq \ell} f'_{\beta,m} \xrightarrow{\geq 0} f_{\beta,m} & \text{if } \beta \text{ is critical,} \\ f_{\beta,m-1} \xrightarrow{\geq \ell} f_{\beta,m} & \text{otherwise} \end{array} \right\}, 0 \leq \beta \leq D-1,$
- $f_{\beta,m} \xrightarrow{\geq 0} f_{\beta+1,0}$, $0 \leq \beta \leq D-2$,
- $\left\{ \begin{array}{ll} d'_{i,j-1} \xrightarrow{\geq \ell} c'_{i,j} & \text{if } d'_{i,j-1} \text{ exists,} \\ c'_{i,j-1} \xrightarrow{\geq \ell} c'_{i,j} & \text{otherwise} \end{array} \right\}, 1 \leq i \leq k,\ 1 \leq j \leq D-n-1,$
- $c'_{i,j} \xrightarrow{\geq 0} d'_{i,j} \xrightarrow{\geq \ell} c'_{i,j+1}$ *when* $d'_{i,j}$ *exists,* $1 \leq i \leq k$, $1 \leq j \leq D-n-1$.

We check that our parameters ℓ and $\#chains$ are respectively bounded by a function of m in instance I'. ℓ is equal to m so we are good with ℓ. Plus there are exactly $k+2 = \frac{m+3}{2}$ chains in I': k chains with selector path job and selector gadget replicas, one with the skeleton jobs and one with the fill jobs. So we are also good with $\#chains$.

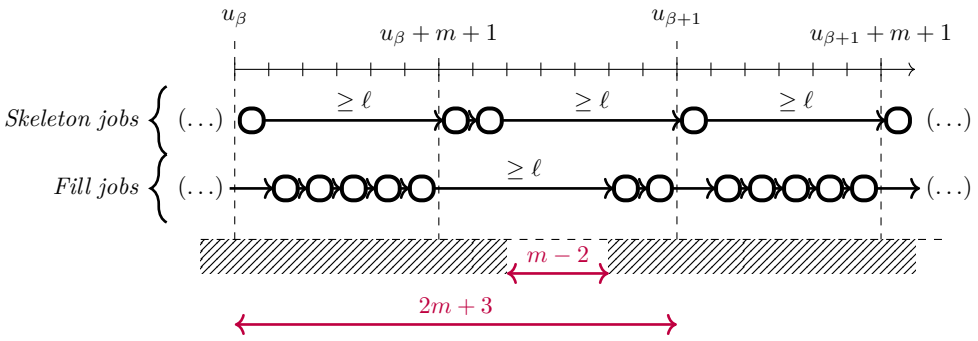


Figure 3.3: Illustration of skeleton jobs and fill jobs in the case of a **critical** time unit β. The precedence relations with no label have minimum delay 0.

Note that while most time units in I take a time span of length $2m + 2$ to be represented in I', critical time units require one more fill job - and thus one additional time unit. So for better readability in the upcoming proofs we define the sequence $(u_\beta)_{0 \leq \beta \leq D-1}$ of starting times for skeleton jobs $(e_{2\beta})_{0 \leq \beta \leq D-1}$.

Definition 35. *Let $\beta \in [0, D]$. If $\beta \leq n - 1$ then we define $u_\beta = (2m + 2) \cdot \beta$. Else let α be the highest $\alpha' \in [0, kn]$ such that $n-1+\alpha c \leq \beta$. Let i be the highest $i' \in [0, n]$ such that $n-1+\alpha c+in < \beta$. Then we define $u_\beta = (2m+2)\cdot\beta+\alpha n+i$.*

It is especially useful to remember that $u_{\beta+1} - u_\beta$ is equal to $(2m + 3)$ if β is critical and $(2m + 2)$ otherwise. Now we show that skeleton jobs and fill jobs work as intended:

Lemma 36. *In any feasible schedule of instance I' skeleton job e_{2D-2} must be scheduled at time $D' - 1$. The other skeleton jobs and the fill jobs must be scheduled at - and fully cover - time units:*
$$\left\{ \begin{array}{ll} [u_\beta, u_\beta + m + 2], u_\beta + 2m + 1, u_\beta + 2m + 2 & \text{if } \beta \text{ is critical,} \\ [u_\beta, u_\beta + m + 1], u_\beta + 2m + 1 & \text{otherwise} \end{array} \right\}, 0 \leq \beta < D.$$

Proof. Let τ' be a feasible schedule of I'. First we show that for all $0 \leq \beta \leq D$ skeleton job $e_{2\beta}$ is scheduled at time u_β. The definition of $prec'$ infers that $\tau'(e_{2\beta+2}) - \tau'(e_{2\beta})$ is at least $(2m + 3)$ if β is critical and $(2m + 2)$ otherwise. Then by direct induction on β it means that $e_{2\beta}$ must be scheduled at time u_β or later. By contradiction if one of these skeleton jobs is scheduled strictly later, by direct induction on β job $e_{2\beta}$ would be scheduled later than $u_D = (2m + 2) \cdot D + kn^2 + 1 = D' - 1$ - contradicting the feasibility of τ'. Thus skeleton jobs of the form $e_{2\beta}$ are scheduled at time u_β.

Next we infer the starting time of the remaining skeleton jobs. Let $0 \leq \beta \leq D - 1$. If β is critical then the precedence delays in $prec'$ imply that $\tau'(e'_{2\beta+1}) = u_\beta + m + 1$ and $\tau'(e_{2\beta+1}) = u_\beta + m + 2$. Otherwise they imply that $\tau'(e_{2\beta+1}) = u_\beta + m + 1$.

Finally we infer the starting time of all fill jobs in a similar fashion: we show that the wanted times are the minimal ones, and that scheduling any of our jobs

strictly later would make the last job in the chain to be scheduled later than $D'-1$. We get that jobs $f_{\beta,0}$ to $f_{\beta,m-1}$ cover time interval $[u_\beta+1, u_\beta+m]$. If β is critical then $\tau'(f'_{\beta,m}) = u_\beta + 2m + 1$ and $\tau'(f_{\beta,m}) = u_\beta + 2m + 2$. Otherwise $\tau'(f_{\beta,m}) = u_\beta + 2m + 1$. □

This means that within each time interval $[u_\beta, u_{\beta+1})$ only the time units in $[u_\beta + m + 3, u_\beta + 2m]$ are available if β is critical and in $[u_\beta + m + 2, u_\beta + 2m]$ otherwise. This successfully leaves $m-2$ available time units if β is critical and $m-1$ of them otherwise.

Proposition 37. *Instance I is feasible if and only if instance I' is feasible.*

Proof. Name the machines from 1 to m arbitrarily.

($\Longleftarrow$) Let τ' be a feasible schedule of I'. By Lemma 36 all selector paths $c'_{i,j}$ and selector gadgets $d'_{i,j}$ are scheduled in time intervals of the form $[u_\beta+m+3, u_\beta+2m]$ when $\beta \in [0, D-1]$ is critical and in $[u_\beta+m+2, u_\beta+2m]$ otherwise. So their starting time is of the form $u_{\beta'} + 2m - j'$ for some $\beta' \in [0, D-1]$, $j' \in [1, m-2]$ if β' is critical and $j' \in [1, m-1]$ otherwise.

We propose schedule τ of instance I where every original job $c_{i,j}$ or $d_{i,j}$ is scheduled at time β' on machine j'. And all floor jobs/gadgets are scheduled at their only acceptable time unit. Since τ' is feasible there are at most $m-2$ jobs scheduled at any critical time and at most $m-1$ jobs at any other time unit in τ. Plus *prec* restricted to selector paths and gadgets is included in *prec'*, so all precedence constraints are met in τ. We conclude that τ is feasible.

($\Longrightarrow$) Let τ be a feasible schedule of I. Without loss of generality we can assume that, in schedule τ, a selector gadget $d_{i,j}$ is always performed by a machine of higher index than the corresponding selector path job $c_{i,j}$. We propose schedule τ' of instance I' where if job $c_{i,j}$ (resp. $d_{i,j}$) is scheduled at time β on machine j' in τ then replica $c'_{i,j}$ (resp. $d'_{i,j}$) is scheduled at time $u_\beta + 2m - j'$. And all skeleton/fill jobs are scheduled at their only acceptable time unit accordingly to the proof of Lemma 36.

Since τ is feasible all replicas $c'_{i,j}, d'_{i,j}$ are scheduled at different times and they do not conflict with skeleton/fill jobs. Then according to *prec* only the precedence constraints of the form $c'_{i,j} \xrightarrow{\geq 0} d'_{i,j}$ remain to be checked. Such precedence relations were taken care of by our hypothesis on schedule τ at the beginning. We conclude that τ' is feasible. □

By combining Propositions 32 and 37 this concludes the $W[2]$-hardness proof of the corresponding parameterized problem.

3.3.3 On Parallel Machines with Chains

On parallel machines with job time windows we show that the problem becomes para-NP-hard parameterized by ℓ_{max} (with $\ell_{max} = 1$) .

Theorem 38. *With minimum delays $P|chains(\ell_{i,j}), p_j = 1, r_j, \bar{d}_j|\star$ is para-*NP*-hard parameterized by maximum delay ℓ_{max}.*

We reduce from 3-COLORING. (Note that the same base idea will be reused in the reductions of Subsection 4.3.2.) Given $G = (V, E)$ with n nodes and m edges, we build an instance I of $P|chains(0, \ell), p_j = 1, r_j, \bar{d}_j|\star$ which represents the color choice of each node by a chain. Then I will be feasible if and only if G admits a valid 3-coloring. A full example is given in Figure 3.5.

Definition 39 (Vertex chain $\mathcal{C}'_i$ with with $\ell_{max} = 1$)**.** *We segment time into $m+2$ segments: a color choice segment $[0, 6n)$, m edge check segments of length 6 along $[6n, 6(n+m))$ and a closing segment $[6(n+m), 6(n+2m))$. We define $\mathcal{C}'_i$ as a chain of $3(2n+m-2i-1)+deg(v_i)+1$ jobs, where $deg(v_i)$ is the degree of node v_i in G. These jobs will fulfill two roles:*

- *Propagators $O^i_{j,k}$: The position of job $O^i_{i,0}$ between 0, 1 and 2 will give the color choice of node v_i. The other $3(2n+m-2i)$ jobs $O^i_{j,k}$ ($i \leq j \leq 2n+m-i-1$, $0 \leq k \leq 2$) and $O^i_{2n+m-i-1,0}$ will propagate this color choice along the whole chain while keeping the maximum delay value at 1. Job $O^i_{j,k}$ will have time window $[6j+2k, 6j+2k+3)$.*
- *Edge jobs J^i_j: The $deg(v_i)$ jobs J^i_{n+j} will represent the color choice of node v_i in every edge e_j where node v_i is in ($0 \leq j \leq m-1$). Job J^i_{n+j} will have time window $[6(n+j)+1, 6(n+j)+4)$.*

Then these jobs form chain $\mathcal{C}'_i$ with the following precedence relations:

- *For $j \in [0, n+2m-2], j = n+j'$ such that node v_i is part of edge $e_{j'}$ (i.e. in the edge check segment associated to an edge $e_{j'}$ containing node v_i):*
$$O^i_{j,0} \xrightarrow{\geq 0} J^i_j \xrightarrow{\geq 0} O^i_{j,1} \xrightarrow{\geq 1} O^i_{j,2} \xrightarrow{\geq 1} O^i_{j+1,0}$$
- *For every other $j \in [0, n+2m-2]$:*
$$O^i_{j,0} \xrightarrow{\geq 1} O^i_{j,1} \xrightarrow{\geq 1} O^i_{j,2} \xrightarrow{\geq 1} O^i_{j+1,0}$$

Now we only have access to minimum delays. So we must ensure that the color choice is faithfully propagated. We intend to do so by forcing all vertex chain delays to be equal to their minimum value in any feasible schedule. This is done via the addition of the following gadget chains:

Definition 40 (Gadget chain $\mathcal{C}'_{i,1}$ (resp. $\mathcal{C}'_{i,2}$))**.** *Gadget chain $\mathcal{C}'_{i,1}$ is defined in parallel with vertex chain $\mathcal{C}'_i$, with $3(2n+m-2i-1)+1$ propagators $O^{i,1}_{j,k}$ of time window $[6j+2k, 6j+2k+3)$. For every $j \in [0, n+2m-2]$ we have the following precedence relations:*

$$O^{i,1}_{j,0} \xrightarrow{\geq 1} O^{i,1}_{j,1} \xrightarrow{\geq 1} O^{i,1}_{j,2} \xrightarrow{\geq 1} O^{i,1}_{j+1,0}$$

Gadget chain $\mathcal{C}'_{i,2}$ is defined the same way with $3(2n+m-2i-1)+1$ propagators $O^{i,2}_{j,k}$.

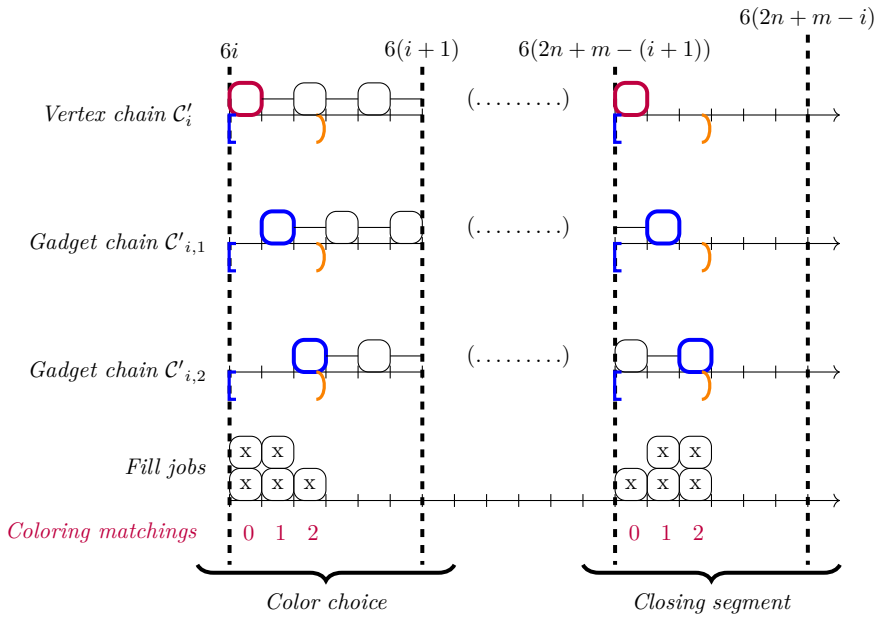


Figure 3.4: A toy situation with three machines and the three chains related to a node in the $P|chains(\ell_{i,j}), p_j = 1, r_j, d_j|\star$ reduction with minimum delays. In any feasible schedule featuring these chains, for $k \in \{0, 1, 2\}$ exactly one chain starts at time $6i+k$, and then this chain has to end at time $6(n+2m-(i+1))+k$.

Then Figure 3.4 illustrates how such gadget chains are expected to operate. In the color choice segment they fill the positions that correspond to discarded colors. Then, according to the minimum precedence delays that we put, the chain starting at the rightmost time in the color choice segment has no choice but to end at the rightmost available time in the closing segment. In turn this forces the chain starting at the middle time to end at the middle time in the closing segment. Eventually this forces the chain starting at the leftmost time to end at the leftmost time in the closing segment.

Now this toy situation only has exactly three available machines in both the color choice and the closing segment intervals in which the reasoning is done. Therefore fill jobs are added in these segments so that this reasoning can be applied successively to nodes v_0, v_1 etc.. Note that the number of fill jobs per time unit is nontrivial since we must take into account the propagators from other vertex/gadget chains.

Furthermore in the edge check segments we use fill jobs in conjunction with the edge jobs in order to make the schedule invalid if an edge features two jobs with the same color in the corresponding 3-coloring candidate.

The fill jobs in instance I are defined below. We recommend following along using the example given in Figure 3.5.

Definition 41 (Fill jobs)**.**

(1) *Color choice segment* $[0, 6n)$

Let $i \in [0, n-1]$*. In time segment* $[6i, 6(i+1))$*:*

- *At time* $6i$*: set* $M - 1 - 2i$ *fill jobs.*
- *At time* $6i + 1$*: set* $M - 1 - i$ *fill jobs.*
- *At time* $6i + 2$*: set* $M - 2 - 2i$ *fill jobs.*

(2) *Edge check segments* $[6n, 6(n+m))$

Let $j \in [0, m-1]$*. In time segment* $[6(n+j), 6(n+j+1))$*:*

- *At time* $6(n+j)+1$*: set* $M - n - 1$ *fill jobs.*
- *At time* $6(n+j)+3$*: set* $M - n - 1$ *fill jobs.*

(3) *Closing segment* $[6(n+m), 6(2n+m))$

Let $i \in [0, n-1]$*. In time segment* $[6(n+2m-(i+1)), 6(n+2m-i))$*:*

- *At time* $6(n+2m-(i+1))$*: set* $M - 2 - 2i$ *fill jobs.*
- *At time* $6(n+2m-(i+1))+1$*: set* $M - 1 - i$ *fill jobs.*
- *At time* $6(n+2m-(i+1))+2$*: set* $M - 1 - 2i$ *fill jobs.*

Now we show that vertex chains and gadget chains work as intended. First we get the following lemma when considering each chain individually:

Lemma 42. *Let* $0 \leq i \leq n-1$*. In any feasible schedule, if a chain starts at time* $6i + k$ *with* $k \in \{0, 1, 2\}$*, then all jobs* J *in this chain are scheduled at time* $r(J) + k$ *or later, where* $r(J)$ *is the release date of job* J*.*

Proof. First the result is proved for vertex chains $\mathcal{C}'_i$. Suppose we have a feasible schedule where vertex chain $\mathcal{C}'_i$ starts at time $6i + l$ with $l \in \{0, 1, 2\}$.

- Propagators $O^i_{j,k}$: by Definition 39 there is always either a unit-time minimum delay or a job J^i_j between two consecutive jobs $O^i_{j,k}$ in vertex chain $\mathcal{C}'_i$. Thus if the first job $O^i_{i,0}$ is scheduled at time $6i + l = r(O^i_{i,0}) + l$ (or later), then we know that the next propagator $O^i_{i,1}$ is scheduled at time $(6i + l) + 2 = r(O^i_{i,1}) + l$ or later, and so on. By induction on the couple (j, k) with $i \leq j \leq n + m - 1$ and $0 \leq k \leq 2$, we get that all the jobs $O^i_{j,k}$ in vertex chain $\mathcal{C}'_i$ are scheduled at time $(6i + l) + 2 \times (3(j - i) + k) = 6j + 2k + l = r(O^i_{j,k}) + l$ or later.

- Edge jobs J^i_j: let $j_0 < j_1 < \ldots < j_{deg(v_i)-1}$ be the indices of the edges e_j such that $v_i \in e_j$ (if there are any). Let $p \in [0, deg(v_i) - 1]$. According to Definition 39 job $J^i_{j_p}$ is scheduled right before job $O^i_{j_p,0}$ with a minimum delay of length zero between them. Therefore according to our previous point about jobs $O^i_{j,k}$, job $J^i_{j_p}$ is scheduled at time $(r(O^i_{j_p,0}) + l) + 1 = r(J^i_{j_p}) + l$ or later.

Gadget chain $\mathcal{C}'_{i,1}$ (resp. $\mathcal{C}'_{i,2}$) only features propagators. By Definition 40 there is always a unit-time minimum delay between two consecutive jobs $O^{i,1}_{j,k}$ (resp. $O^{i,2}_{j,k}$), so the result can be proven the same way as in the first item of the proof for vertex chains. □

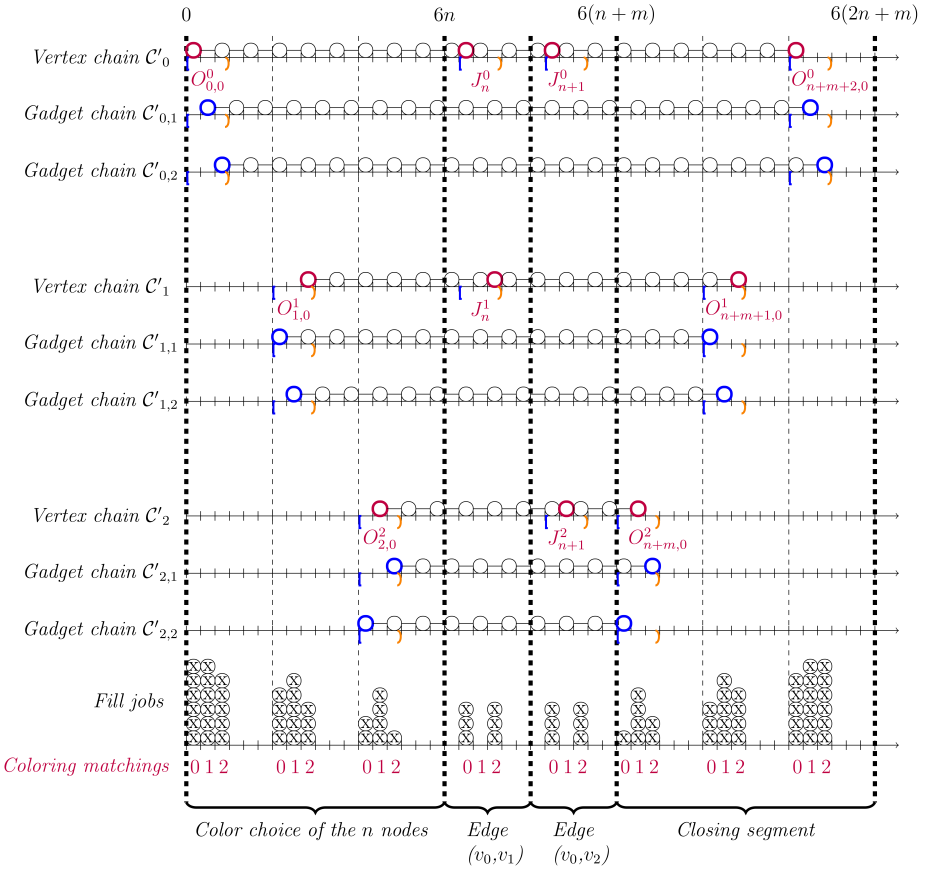


Figure 3.5: An instance of $P|chains(\ell_{i,j}), p_j = 1, r_j, d_j|\star$ with minimum delays and $M = 2n + 1 = 7$ machines representing a graph coloring. We have $G = (V, E)$ with $V = \{v_0, v_1, v_2\}$ and $E = (\{v_0, v_1\}, \{v_0, v_2\})$. This schedule corresponds to the coloring $(0, 2, 1)$.

Then we investigate the interaction between vertex chain $\mathcal{C}'_i$ and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$. We show that gadgets produce the desired effect:

Lemma 43. *Let $0 \leq i \leq n-1$. In any feasible schedule, if a chain starts at time $6i + k$ with $k \in \{0, 1, 2\}$, then all jobs J in this chain have to be scheduled at time $r(J) + k$, where $r(J)$ is the release date of job J.*

Proof. We prove by induction on $i \in [0, n-1)$ that for all $0 \leq j \leq i$ exactly one chain starts at each time $6j$, $6j+1$, $6j+2$ and all jobs J in a chain $\mathcal{C}'_j$, $\mathcal{C}'_{j,1}$, $\mathcal{C}'_{j,2}$ that starts at time $6j+k$ have to be scheduled at time $r(J)+k$.

- According to Definition 41 on time windows $[0, 6)$ and $[6(2n+m-1), 6(2n+m))$, the chain triplet $\mathcal{C}'_0$, $\mathcal{C}'_{0,1}$, $\mathcal{C}'_{0,2}$ is in the situation described in Figure 3.4 . Thus at least one chain must start at time 2 which means by Lemma 42 that this chain has to end at time $6(2n+m-1)+2$. Then time $6(2n+m-1)+2$ is blocked, so by the same lemma another chain cannot start at time 2. Thus the two other chains have to start at the two remaining time positions 0 and 1, one per chain. By Lemma 42 the chain that starts at time 1 ends at time $6(2n+m-1)+1$ (or later but the only other time position possible $6(2n+m-1)+2$ is already blocked). So time position $6(2n+m-1)+1$ is now blocked, which forces the chain that starts at time 0 to end at time $6(2n+m-1)$.

- Let $i \in [0, n-1)$. Assume the induction hypothesis to be true for chain triplets of index j with $0 \leq j \leq i-1$. By Definition 39 we know that only these chain triplets have propagators that might interfere in time windows $[6i, 6(i+1))$ and $[6(2n+m-(i+1)), 6(2n+m-i))$. By induction hypothesis we know that these propagators are fixed, and we deduce that the number of fixed jobs (propagators from other chains plus fill jobs) at the relevant time positions is the following:

  (1) Color choice segment, part $[6i, 6(i+1))$:
  - At time $6i$: $M-1-2i$ fill jobs and $2i$ propagators from other chains which add up to $M-1$ jobs.
  - At time $6i+1$: set $M-1-i$ fill jobs and i propagators from other chains which add up to $M-1$ jobs.
  - At time $6i+2$: set $M-2-2i$ fill jobs and $2i$ propagators from other chains which add up to $M-2$ jobs.

  (2) Closing segment, part $[6(n+2m-(i+1)), 6(n+2m-i))$:
  - At time $6(n+2m-(i+1))$: set $M-2-2i$ fill jobs and $2i$ propagators from other chains which add up to $M-2$ jobs.
  - At time $6(n+2m-(i+1))+1$: set $M-1-i$ fill jobs and i propagators from other chains which add up to $M-1$ jobs.
  - At time $6(n+2m-(i+1))+2$: set $M-1-2i$ fill jobs and $2i$ propagators from other chains which add up to $M-1$ jobs.

  Thus we are again in the situation described in Figure 3.4 and we can prove the result for the triplet $\mathcal{C}'_i$, $\mathcal{C}'_{i,1}$, $\mathcal{C}'_{i,2}$ the same way as in the initialization.

This concludes the proof of the lemma for all chains. □

Now we are able to prove the following equivalence:

Proposition 44. *G is 3-colorable if and only if there exists a feasible schedule for this instance of $P|chains(\ell_{i,j}), p_j = 1, r_j, d_j|\star$.*

Proof. ($\implies$) Suppose we have $(c_0, \ldots, c_{n-1}) \in \{0,1,2\}^n$ a 3-coloring of G where vertex v_i has color c_i. We propose a schedule τ where for all $0 \leq i \leq n-1$, chain $\mathcal{C}'_i$ starts at time $6i + c_i$ and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ start in the two remaining time positions $6i + l_1, 6i + l_2$ in $[3i, 3(i+1))$ (with $l_1 \neq l_2$). Plus we require all delays to match their minimum value.

Then, according to Definition 39, Definition 40 and going from left to right as we did in the proof of Lemma 42, we know that propagators $O^i_{j,k}$, $O^{i,1}_{j,k}$ and $O^{i,2}_{j,k}$ are respectively scheduled at times $6j+2k+c_i$, $6j+2k+l_1$ and $6j+2k+l_2$. In the same way we know that in every edge $e_j \in E$ where node v_i appears, edge job J^i_{n+j} of vertex chain $\mathcal{C}'_i$ is scheduled at time $6(n+j)+1+c_i$. Thus for all jobs J in our proposed schedule, if its chain starts at time $6i + l$ with $l \in \{0,1,2\}$ then it is scheduled at time $r(J) + l$.

We show that there are never more than $M = 2n + 1$ jobs scheduled at any time position. According to the previous paragraph we can infer that for every chain triplet two propagators are scheduled at every even time position and one propagator at every odd time position in time segment $[6i, 6(2n+m-(i+1))+3)$. Recall that only the chains $\mathcal{C}'_j, \mathcal{C}'_{j,1}, \mathcal{C}'_{j,2}$ with $j \leq i$ are present in the two time segments $[0, 6n)$, $6(n + 2m - (i+1))$ related to node v_i. With Definition 41 we count the number of propagators plus the number of fill jobs at every time position and show that it is always no more than $M - 1$:

(1) Color choice segment $[0, 6n)$

Let $i \in [0, n-1]$. In time segment $[6i, 6(i+1))$:

- At time $6i$: $M - 1 - 2i$ fill jobs and $2i$ propagators which add up to $M - 1$ jobs.
- At time $6i + 1$: set $M - 1 - i$ fill jobs and i propagators which add up to $M - 1$ jobs.
- At time $6i + 2$: set $M - 2 - 2i$ fill jobs and $2i$ propagators which add up to $M - 2$ jobs.
- At times $6i + 3$, $6i + 4$, $6i + 5$: respectively i, $2i$, i propagators.

(2) Edge check segments $[6n, 6(n + m))$

Let $j \in [0, m-1]$. In time segment $[6(n+j), 6(n+j+1))$:

- At time $6(n + j) + 1$: $M - n - 1$ fill jobs and n propagators which add up to $M - 1$ jobs.
- At time $6(n + j) + 3$: $M - n - 1$ fill jobs and n propagators which add up to $M - 1$ jobs.
- At times $6(n+j)$, $6(n+j)+2$, $6(n+j)+4$, $6(n+j)+5$: respectively $2n$, $2n$, $2n$, i propagators.

(3) Closing segment $[6(n+m), 6(2n+m))$

Let $i \in [0, n-1]$. In time segment $[6(n+2m-(i+1)), 6(n+2m-i))$:

- At time $6(n+2m-(i+1))$: set $M-2-2i$ fill jobs and $2i$ propagators which add up to $M-2$ jobs.
- At time $6(n+2m-(i+1))+1$: set $M-1-i$ fill jobs and i propagators which add up to $M-1$ jobs.
- At time $6(n+2m-(i+1))+2$: set $M-1-2i$ fill jobs and $2i$ propagators which add up to $M-1$ jobs.
- At times $6(n+2m-(i+1))+3$, $6(n+2m-(i+1))+4$, $6(n+2m-(i+1))+5$: respectively i, $2i$, i, propagators.

Thus only the two edge jobs $J^{i_1}_{n+j}$, $J^{i_2}_{n+j}$ from an edge $e_j = \{v_{i_1}, v_{i_2}\}$ could invalidate the schedule if both jobs were scheduled at the same time. This would mean that $6(n+j)+1+c_{i_1} = 6(n+j)+1+c_{i_2}$ and thus $c_{i_1} = c_{i_2}$, which is impossible since we started from a valid 3-coloring. Therefore at most $2n+1 = M$ jobs are scheduled at any time position.

($\Longleftarrow$) Suppose we have a feasible schedule. For all $0 \leq i \leq n-1$, let $s_i \in \{0,1,2\}$ be such that $6i+s_i$ is the starting time of chain $\mathcal{C}'_i$ (recall that it can only be an odd time because of the fill jobs defined in Definition 41). We show that $(s_0, \ldots, s_{n-1})$ is a 3-coloring of G. By contradiction suppose there is an edge $e_j = \{v_{i_1}, v_{i_2}\} \in E$ such that $s_{i_1} = s_{i_2}$. Then according to Lemma 43 jobs $J^{i_1}_{n+j}$ and $J^{i_2}_{n+j}$ are scheduled at the same time $6(n+j)+s_{i_1}$. However according to Definition 41 and Lemma 43 fill jobs and propagators add up to $M-1$ in all three positions $6(n+j)+1$, $6(n+j)+2$, $6(n+j)+3$.

Thus adding both edge jobs there are $M+1$ jobs scheduled at one of these three time positions, which would make the schedule not feasible. This leads to a contradiction. Thus $(s_0, \ldots, s_{n-1})$ is indeed a 3-coloring of G. □

This proves that $P|chains(\ell^{min}_{i,j}), p_j = 1, r_j, d_j|\star$ with $\ell_{max} = 1$ is NP-hard, which concludes the para-NP-hardness proof of the corresponding parameterized problem.

3.4 Summary & Concluding Remarks

Param.	Sect.	Problem / Setting	Result
ℓ_{max}	3.2	$1\|chains(\ell), p_j = 1, r_j, \bar{d}_j\|\star$	para-NP-hard with exact or maximum delays.
		$1\|chains(\ell), p_j = 1\|C_{max} < D$	$W[1]$-hard with exact delays.
		$1\|chains(\ell_{i,j}), p_j = 1\|C_{max} < D$	$\mathcal{O}(n)$ with maximum delays.
		$1\|chains(0,\ell), p_j = 1, r_j\|C_{max} < D$ $1\|chains(0,\ell), p_j = 1, \bar{d}_j\|\star$	NP-hard with maximum delays.
	3.3	$1\|prec(\ell), p_j = 1\|C_{max} < D$	$XNLP$-hard($\ell_{max} + w$) with minimum delays.
		$1\|chains(0,\ell), p_j = 1\|C_{max} < D$	$W[2]$-hard(ℓ_{max}+#chains) with minimum delays.
		$P\|chains(0,\ell), p_j = 1, r_j, d_j\|\star$	para-NP-hard($\ell = 1$) with minimum delays.

Figure 3.6: Summary of the results obtained in Chapter 3.

In this chapter we investigated the parameterized complexity of the subproblems of RCPSP enhanced with bounded precedence delays. We studied three delay types: exact, maximum and minimum. For all these delay types we established that the addition of precedence delays, even of bounded value ℓ_{max}, makes the problem difficult even on a single machine with chains of unit jobs. This suggests considering other parameters, either by themselves or paired with ℓ_{max}.

Considering parameter ℓ_{max} alone a lot of parameterized scheduling problems are left open, especially in the coupled task setting. Indeed all our reductions involving bounded delays heavily rely on the fact that the chains of jobs can be of arbitrary length. While a couple FPT algorithms have been proposed in the exact delay case by Khatami et al. on a fixed number of parallel machines [KOS23] and Bessy and Giroudeau on a single machine with a compatibility graph G_c [BG19], the parameterized complexity of coupled task scheduling with minimum or maximum delays has been trailing behind.

3.4.1 Problem Map with Parameter ℓ_{max} and Minimum Delays

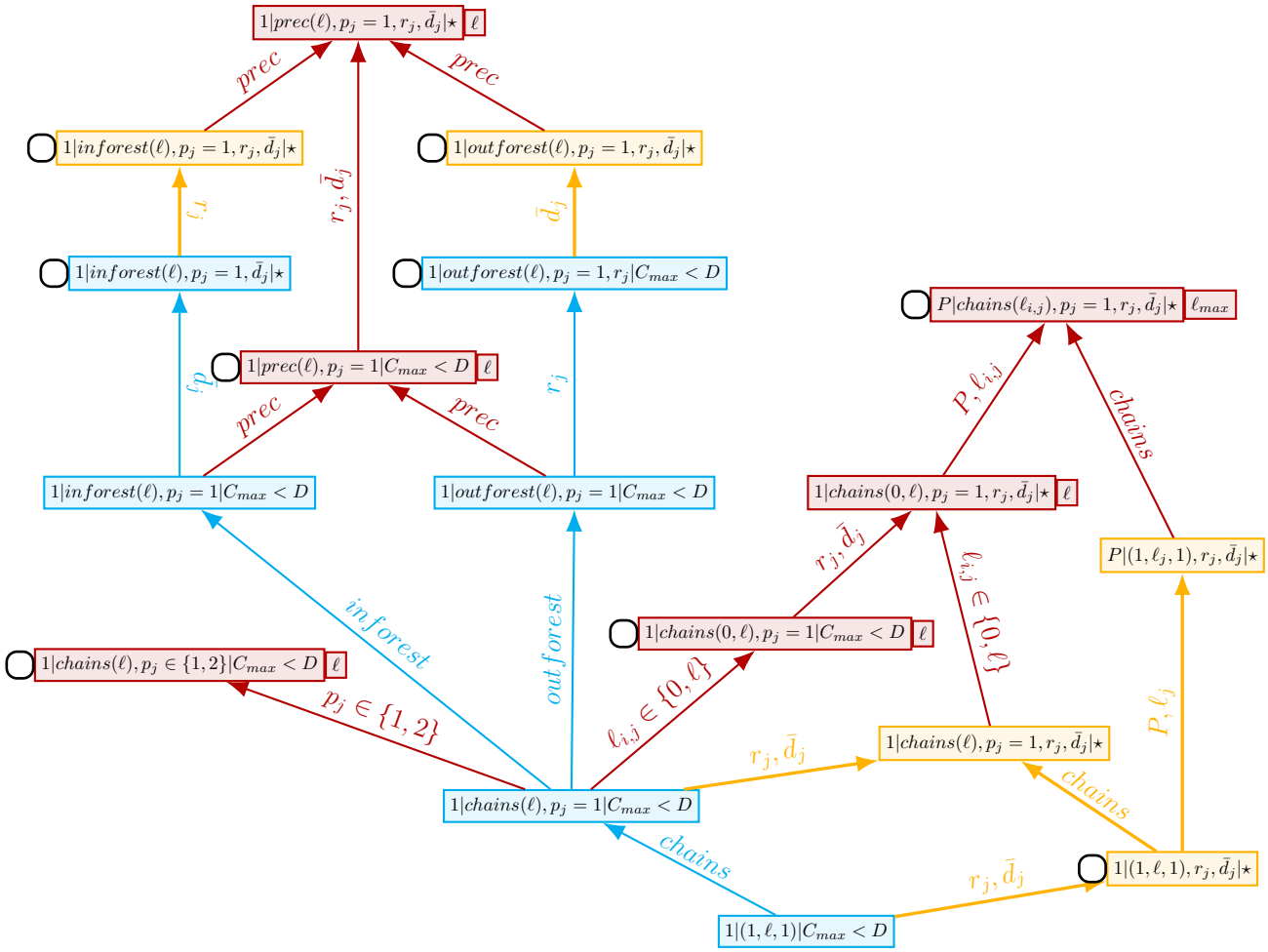


Figure 3.7: Problem map of parameter ℓ_{max} with minimum delays.

3.4.2 Problem Map with Parameter ℓ_{max} and Exact Delays

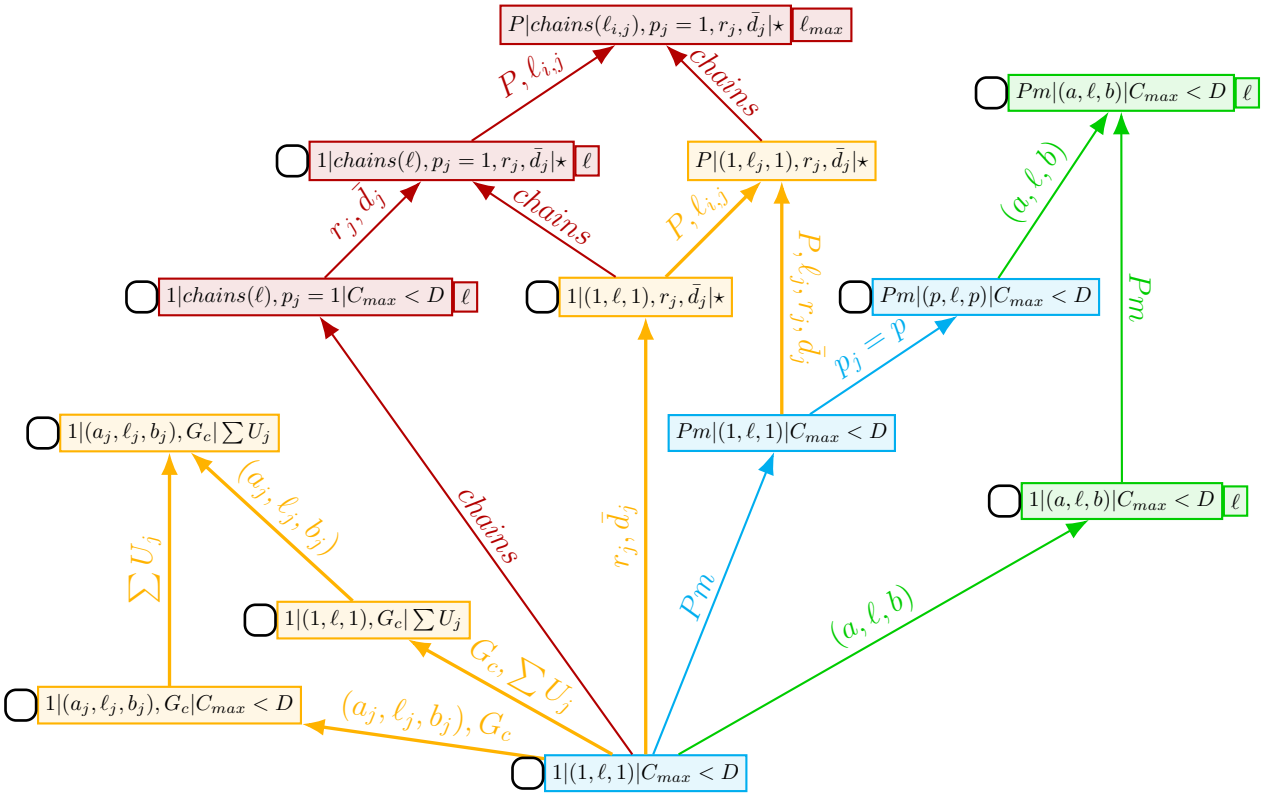


Figure 3.8: Problem map of parameter ℓ_{max} with exact delays.

3.4.3 Problem Map with Parameter ℓ_{max} and Maximum Delays

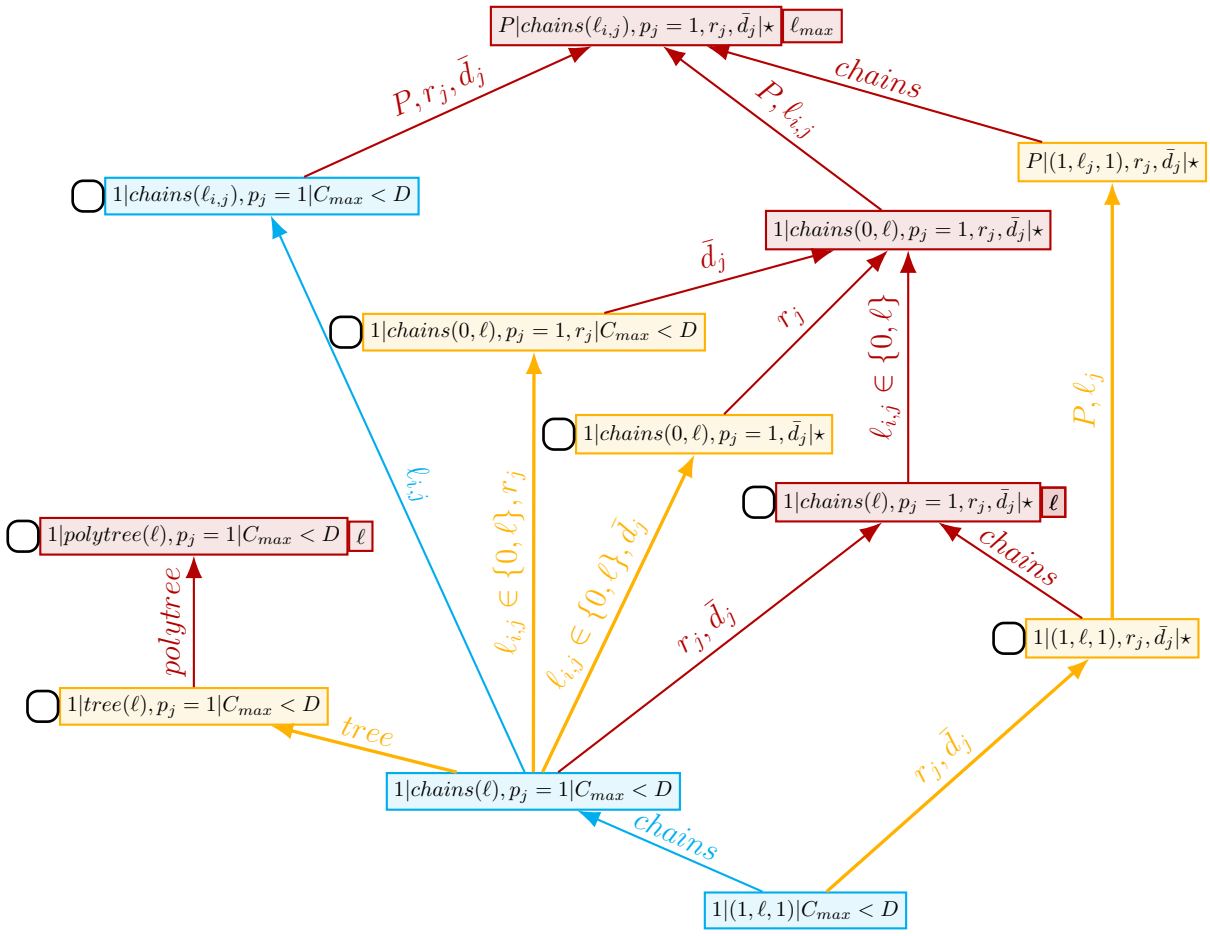


Figure 3.9: Problem map of parameter ℓ_{max} with maximum delays.

Chapter 4

Results with Pathwidth μ

4.1 Introduction

Pathwidth μ is the maximum number of job time windows which can overlap at any time t. This corresponds to the pathwidth pw (plus one) of the interval graph given by the job time window intersections. Hence the name. However our interpretation is closer to clique number ω, which is the size of the maximum clique in a graph. While both notions are equivalent in interval graphs ($\omega = pw + 1$) the clique number interpretation might make it easier to understand the relevance of parameter μ in the context of scheduling.

Indeed μ bounds the number of job candidates within any interval with no intermediate release dates or deadlines. So by considering these intervals individually, one can go over all possible job subsets which could be scheduled in each one of them. This suggests a dynamic programming approach, which has been successful in several occasions over the past few years. The first results with pathwidth μ were from [MK21] by Munier, where $P|prec, p_j = 1, r_j, \bar{d}_j|C_{max}$ and $P|prec, p_j = 1, r_j, \bar{d}_j|L_{max}$ were proved FPT. Since then dynamic programming FPT algorithms have been given on $P|prec, r_j, \bar{d}_j|\star$ parameterized by $\mu + p_{max}$ by Tarhan et al. [TCH$^+$23] and $P|\mathcal{M}_j(type), prec, r_j, \bar{d}_j|L_{max}$ parameterized by $\mu + min(p_{max}, \sigma)$ by Hanen and Munier [HMK23]. Note that both results require the addition of a parameter which bounds the impact of having arbitrary processing times. This is not surprising given that $P2|r_j, \bar{d}_j|\star$ was proved para-NP-hard parameterized by μ when $\mu = 4$ by Hanen and Munier [HMK23]. This is also highlighted in Figure 4.1 which illustrates the parameterized landscape of $P|prec, r_j, \bar{d}_j|\star$. However in the context of single-machine scheduling parameter μ seems to have more potential. Indeed in [BdWH21] Baart et al. showed that $1|r_j, s_{jk}, reject, \bar{d}_j| \sum_{j \notin R}(v_j - w_j T_j)$ has a FPTAS parameterized by μ. As the Section 4.2 will confirm, it appears that parameter μ is enough to schedule jobs of arbitrary length on a single machine efficiently.

Note that pathwidth μ can be considered in problems where release dates and deadlines are not explicitly given in the input. This was the case in [KT21]

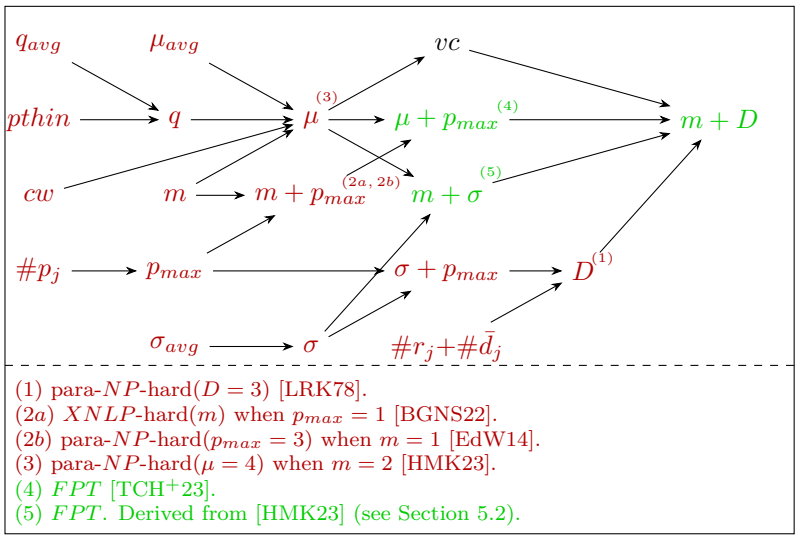


Figure 4.1: Parameterized landscape of $P|prec, r_j, \bar{d}_j|\star$.

where Munier and Tang studied problem $\bar{P}|prec, p_j = 1, c_{ij} = 1|C_{max} < D$. Precedence graph $prec$ and makespan upper bound D served as a basis to infer a release date and a deadline for each job. Then they showed that their problem was FPT parameterized by the pathwidth of the resulting time window graph.

While this work focuses on job time windows, in [BvdW20] Bodlaender and van der Wegen obtained several interesting results on problem $1|chains(\ell_{i,j}), p_j = 1, r_C, \bar{d}_C|\star$ with a similar parameter called thickness th. Its definition is near identical, except time windows are defined on the precedence chains instead of individual jobs. With exact or minimum precedence delays they showed that $1|chains(\ell_{i,j}), p_j = 1, r_C, \bar{d}_C|\star$ is NP-hard even when delays are given in unary. In the parallel-machine setting with delays given in unary they also gave an XP algorithm parameterized by the thickness - i.e. the maximum number of chain time windows which can include a time unit. This only reinforces our intuition that an extra property other than the delay values must be bounded when hoping to find a FPT algorithm.

In this chapter we focus on scheduling problems with release dates and deadlines explicitly given in the input. In Section 4.2 we propose a FPT algorithm on single machine scheduling and show that it works even in the presence of precedence relations. Then in Section 4.3 we prove that the problem becomes para-NP-hard when these relations are enhanced with delay values, even with chains of unit jobs. In Section 4.4 we pair μ with maximum delay value ℓ_{max} and show that scheduling unit jobs then becomes FPT even on identical parallel machines with general precedence. Ultimately in Section 4.5 we summarize the results obtained in this chapter and give our concluding remarks.

4.2 A FPT Algorithm on a Single Machine

In this section we show that $1|r_j, \bar{d}_j|C_{max}$ and $1|prec, r_j, \bar{d}_j|C_{max}$ are FPT parameterized by μ. This result was first presented during the MAPSP 2022 conference [HMMK22]. We begin with simpler problem $1|r_j, \bar{d}_j|C_{max}$.

Theorem 45. *$1|r_j, \bar{d}_j|C_{max}$ can be solved in time $\mathcal{O}(\mu \cdot log(\mu) \cdot 4^{\mu} \cdot n + n \cdot log(n))$ and space $\mathcal{O}((\mu + log(D)) \cdot 2^{\mu} \cdot n)$.*

We gather all the distinct values of release dates and deadlines. After sorting them in nondecreasing order we get sequence $(u_k)_{1 \leq k \leq K}$ where $K \leq 2n$. We segment time into the $K-1$ intervals $[u_k, u_{k+1})$. We propose a dynamic programming algorithm where we consider these intervals in order. At stage k we decide which jobs start within interval $[u_k, u_{k+1})$. A job which has been scheduled must be remembered so that it is scheduled exactly once. Once we are past its deadline it can be forgotten. So in each state we will only remember the jobs which are already scheduled but with a deadline greater than u_{k+1}.

Definition 46. *For all k in $[1, K-1]$ we define:*

- $Z_k = \{j \in \mathcal{J} | \bar{d}_j \leq u_{k+1}\}$,
- $\Gamma_k = \{j \in \mathcal{J} | (r_j \leq u_k) \wedge (u_{k+1} \leq \bar{d}_j)\}$.

First we show that the following schedules are dominant on $1|r_j, \bar{d}_j|C_{max}$:

Lemma 47. *Given a feasible schedule τ on an instance of $1|r_j, \bar{d}_j|C_{max}$ one can build a feasible schedule $\tau' = \gamma'_1 \dots \gamma'_{K-1}$ with a makespan lower than or equal to the makespan of τ and such that for all k in $[1, K-1]$:*

(i) γ'_k is a sequence of jobs which all belong to Γ_k,

(ii) all the jobs in sequence γ'_k start within interval $[u_k, u_{k+1})$,

(iii) the jobs in γ'_k are scheduled in nondecreasing order of their deadlines.

Proof. We prove this result by induction on k. Let $k \in [1, K-1]$. Suppose the result holds for all $k' < k$. Let γ_k be the sequence of jobs which start within interval $[u_k, u_{k+1})$ in τ.

First note that τ is feasible and all jobs have a positive processing time. So for every job j in γ_k, the starting time of which is in $[u_k, u_{k+1})$, has release date at most u_k and deadline at least u_{k+1}. Therefore all jobs in γ_k fulfill points (i) and (ii).

Now we intend to define sequence γ'_k as a reordering of the jobs in γ_k. let j be a job with highest deadline in γ_k. We have $\gamma_k = S'jS$. If S is not empty, we reorder to the sequence $S'Sj$ by pushing the jobs in sequence S by p_j time units to the left and insert j right after. This does not change the makespan. Plus for all jobs i in S we have $\bar{d}_i \leq \bar{d}_j$, so j can be inserted as desired. Now in the resulting sequence $S'Sj$ all jobs in $S'S$ are entirely processed within time

interval $[u_k, u_{k+1})$. So they can be sorted in-place in nondecreasing order of their deadlines without impacting feasibility.

We set γ'_k as the resulting reordering of the jobs in γ_k. Then point (i) is valid according to our second paragraph, as well as point (ii) by the definition of γ_k. Finally the jobs in γ'_k are clearly in nondecreasing order of their deadlines, so point (iii) is cleared. Then points (i) to (iii) hold for all $j \leq k$. This concludes the induction. □

This reduces the optimal makespan to a subset of feasible schedules. We give such schedules the following name:

Definition 48. *A schedule τ is called time-interval-ordered if it is feasible, active and satisfies the properties described in Lemma 47.*

If we build a prefix to such a schedule up to stage k, we show that the set of scheduled jobs can be partitioned between Z_k and L_k subset of $(\Gamma_k - Z_k)$.

Lemma 49. *Let $\tau = \gamma_1 \dots \gamma_{K-1}$ be a time-interval-ordered schedule on an instance I of $1|r_j, \bar{d}_j|C_{max}$.*

Then for every k in $[1, K-1]$ schedule $\gamma_1 \dots \gamma_k$ is a time-interval-ordered schedule on job subset $Z_k \cup L_k$ where $L_k = (\bigcup_{1 \leq h \leq k} set(\gamma_h)) \cap (\Gamma_k - Z_k)$.

Proof. Feasibility, activeness and Lemma 47 are stable properties by prefix operation. What is left to show is that the set of jobs featured in partial schedule $\gamma_1 \dots \gamma_k$ is indeed $Z_k \cup L_k$.

($\subseteq$) Let $\ell \in [1, k]$ and $j \in set(\gamma_\ell)$. If $j \in Z_k$ then we are done. Else it means that $\bar{d}_j \geq u_{k+1}$. And since $j \in set(\gamma_\ell)$ we know that $r_j \leq u_\ell \leq u_k$. So $j \in \Gamma_k$ and thus $j \in L_k$.

($\supseteq$) By definition L_k is included in $\bigcup_{1 \leq h \leq k} set(\gamma_h)$. Now let $j \in Z_k$. By contradiction suppose that j is not in schedule $\gamma_1 \dots \gamma_k$. Since τ is a feasible schedule on the whole instance there is some $\ell \in [k+1, K-1]$ such that j is scheduled in γ_ℓ. Then we would have $j \in \Gamma_k$, which would mean that $\bar{d}_j \geq u_{\ell+1} > u_{k+1}$. So $j \notin Z_k$, which would lead to a contradiction. Thus $j \in \bigcup_{1 \leq h \leq k} set(\gamma_h)$. □

So each dynamic programming state can be described as a couple (k, L_k) where k represents interval $[u_k, u_{k+1})$ and L_k is the subset of jobs in $(\Gamma_k - Z_k)$ which are already scheduled. Then a *time-interval-ordered* schedule $\gamma_1 \dots \gamma_{K-1}$ can be represented as a state path $(1, L_1) \to (\dots) \to (1, L_{K-1})$ where every L_k is a subset of $(\Gamma_k - Z_k)$. Given two consecutive states (k, L_k) and $(k+1, L_{k+1})$ we show that the set of added jobs γ_{k+1} can be retrieved from sets L_k, L_{k+1} and Γ_{k+1}.

Lemma 50. *Let $\tau = \gamma_1 \dots \gamma_{K-1}$ be a time-interval-ordered schedule on an instance I of $1|r_j, \bar{d}_j|C_{max}$. Let $(1, L_1) \to (\dots) \to (K-1, L_{K-1})$ be the corresponding state path accordingly to Lemma 49. Then for every k in $[1, K-2]$:*

$$\gamma_{k+1} = (Z_{k+1} \cup L_{k+1}) - (Z_k \cup L_k) = [(Z_{k+1} \cap \Gamma_{k+1}) \cup L_{k+1}] - L_k.$$

Proof. The first equality is a consequence of Lemma 49. We show that the second equality holds.

($\subseteq$) Let $j \in (Z_{k+1} \cup L_{k+1}) - (Z_k \cup L_k)$. If $j \in L_{k+1}$ then it is not in L_k, so $j \in (L_{k+1} - L_k)$. If $j \in Z_{k+1}$ then it is not in Z_k, so its deadline is exactly u_{k+2}. Hence its release date is lower than or equal to u_{k+1}, which means that $j \in \Gamma_{k+1}$. And since j is also not in L_k: $j \in ((Z_{k+1} \cap \Gamma_{k+1}) - L_k)$.

($\supseteq$) Let $j \in [(Z_{k+1} \cap \Gamma_{k+1}) \cup L_{k+1}] - L_k$. We show that $j \notin Z_k$. Suppose $j \in (Z_{k+1} \cap \Gamma_{k+1})$. By the definition of Γ_{k+1}: $\bar{d} \geq u_{k+2} > u_{k+1}$. So $j \notin Z_k$. Now suppose $j \in L_{k+1}$. By Lemma 49 L_{k+1} is a subset of $\Gamma_{k+1} - Z_{k+1}$. So $j \notin Z_{k+1}$, which means that $\bar{d}_j > u_{k+2} > u_{k+1}$. So $j \notin Z_k$. □

Finally in order to find the optimal makespan of our instance, we show that we only need to consider time-interval-ordered schedules which are optimal on every prefix $\gamma_1 \dots \gamma_k$.

Lemma 51. *Let $\tau = \gamma_1 \dots \gamma_{K-1}$ be a time-interval-ordered schedule with optimal makespan on an instance I of $1|r_j, \bar{d}_j|C_{max}$. Then one can build a time-interval-ordered schedule $\tau' = \gamma'_1 \dots \gamma'_{K-1}$ also with optimal makespan and such that for all $k \in [1, K-1]$ schedule $\gamma'_1 \dots \gamma'_k$ has optimal makespan among the schedules of state (k, L_k).*

Proof. This is proved by downward induction on $k \in [1, K-1]$. Suppose that for all $j > k$ schedule $\gamma_1 \dots \gamma_j$ is optimal among the schedules associated with the same state - which is true in base case $k = K-1$. Let τ_k be a schedule associated to the same state as schedule $\gamma_1 \dots \gamma_k$ and with optimal makespan. By Lemma 47 there is $\tau'_k = \gamma'_1 \dots \gamma'_k$ *time-interval-ordered* schedule with the same makespan as τ_k. We propose schedule $\tau' = \gamma'_1 \dots \gamma'_k \gamma_{k+1} \dots \gamma_{K-1}$, which is also *time-interval-ordered* and for which the result holds for all $j \geq k$. This concludes the induction. □

This motivates the procedure given in Algorithm 1. Set dummy final value $u_{K+1} = \infty$ and final state $(K, \emptyset)$. For $k = 1$ to $K-1$ use the optimal makespan from states (k, L_k) and extend them to the states $(k+1, L_{k+1})$.

Each of these state pairs is treated by Algorithm 2, which operates accordingly to Lemma 50. Once the unique set of jobs which start within time interval $[u_k, u_{k+1})$ is retrieved, they are appended in nondecreasing order of their deadline and in an active manner (i.e. as soon as possible). Eventually either the schedule computed by Algorithm 2 is invalid, or it gives an optimal makespan candidate for the successor state. At the end of the procedure the optimal makespan of the whole instance is given by the value of state $(K, \emptyset)$.

Before we analyze the time complexity our dynamic programming algorithm we bound the size of states and successors:

Lemma 52. *$\forall k \in [1, K-1]$, $|\Gamma_k| \leq \mu$. Plus every state has at most 2^μ successors.*

Proof. By the definition of Γ all its tasks must include time r_j in their time window. So by the definition of pathwidth μ this bounds the size of Γ by μ.

Algorithm 1 main()

▷ Solving $1|r_j, \bar{d}_j|C_{max}$.
1: preprocessing() ▷ Compute sequence $(u_k)_{1\leq k\leq K}$. Compute job sets Γ_k.
2: $u_{K+1} \leftarrow \infty$; $\Gamma_K \leftarrow \emptyset$
3: Create table T with K columns (1 to K) and $2^{|\Gamma_k|}$ slots in each column k.
4: Initialize T with ∞ everywhere.
5: $T[1, \emptyset] \leftarrow 0$
6: **for** $k = 1$ to $K - 1$ **do**
7: **for** each $(L_k, L_{k+1}) \subseteq (\Gamma_k - Z_k) \times (\Gamma_{k+1} - Z_{k+1})$ **do**
8: $T[k + 1, L_{k+1}] \leftarrow \min(T[k + 1, L_{k+1}], \text{extend}(k, L_k, L_{k+1}))$
9: **return** $T[K, \emptyset]$

Algorithm 2 extend(k, L_k, L_{k+1})

▷ Input: $k \in [1, K - 1], L_k \subseteq \Gamma_k, L_{k+1} \subseteq \Gamma_{k+1}$.
▷ Goal: start from an optimal schedule of the form $\gamma_1 \dots \gamma_k$ and attempt to extend it to a feasible schedule of the form $\gamma_1 \dots \gamma_{k+1}$.
▷ By Lemma 47 all the added jobs must start at u_k or later.
1: $Cmax \leftarrow \max(u_k, T[k, \gamma_k])$
2: **if** $Cmax = \infty$ **then return** ∞
▷ Ensure that all jobs in L_k with a deadline higher than u_{k+2} are still in L_{k+1}.
3: **if** $(L_k - Z_{k+1}) \not\subseteq L_{k+1}$ **then return** ∞
▷ Add jobs accordingly to Lemma 47 and 50.
4: $add \leftarrow [(Z_{k+1} \cap \Gamma_{k+1}) \cup L_{k+1}] - L_k$
5: sort(add, nondecreasing $\bar{d}_j$)
6: **for** j in add (in order) **do**
7: $Cmax \leftarrow Cmax + p_j$
8: **if** $Cmax > \bar{d}_j$ **then return** ∞
▷ Ensure that all the added jobs start earlier than u_{k+1}.
9: $last \leftarrow$ last job in add.
10: **if** $Cmax - p_{last} \geq u_{k+1}$ **then return** ∞
11: **return** $Cmax$

Now given a state $(k, L_k) \in [1, K - 1] \times (\Gamma_k - Z_k)$ his successors are of the form $(k + 1, L_{k+1})$ where L_{k+1} is a subset of $(\Gamma_{k+1} - Z_{k+1})$. Thus state (k, L_k) has at most 2^μ successors. □

Now that the number of states and successors have been bounded we compute the time and space complexity of the proposed dynamic programming algorithm. In the preprocessing phase of Algorithm 1 sequence $(u_k)_{1\leq k\leq K-1}$ and sets Γ_k are computed. This takes time $\mathcal{O}(n \cdot log(n))$ and space $\mathcal{O}(\mu \cdot n)$. Now by Lemma 52 the number of states associated to each interval $[u_k, u_{k+1})$ is bounded by 2^μ, and for each one at most 2^μ candidate successors are explored. For each couple $((k, L_k), (k + 1, L_{k+1}))$ considered in Algorithm 1 line 7, Algorithm 2 adds at most μ jobs to the schedule represented by state (k, L_k). By Lemma 50 they can be retrieved from sets L_k and L_{k+1} (line 4) and appended accordingly to Lemma 47 (lines 5 to 12) in time $\mathcal{O}(\mu \cdot log(\mu))$.

Then either some deadline is invalidated or the jobs are successfully added and a new candidate makespan value is proposed to the successor. So the main loop of Algorithm 1 takes time $\mathcal{O}(\mu \cdot log(\mu) \cdot 4^\mu \cdot n)$. Now we consider space

complexity. For each state (k, L_k) we must remember index k, some encoding of set L_k and the minimum makespan currently found. By Corollary 52 this requires space $\mathcal{O}(\mu + log(D))$ where $D = max_{j\in\mathcal{J}}(\bar{d}_j)$. Finally once the whole state graph has been computed, a feasible schedule with optimal makespan can be retrieved by starting from the final state and going backwards in the state graph. This leads the time and space complexity given in Theorem 45.

Note that if only the optimal value is wanted and not a schedule, it is possible to reduce the space complexity to $\mathcal{O}((\mu + log(D)) \cdot 2^{\mu})$. Indeed only the optimal makespan of the states associated to stage k must be remembered at any time in order to find the optimal makespan of all states associated to stage $k+1$. And when all the states associated to stage k have interacted with their successors, the space they take can be freed and used for the states associated to stage $k+2$. Finally instead of a preprocessing phase sequence $(u_k)_{1\leq k\leq K-1}$ and sets Γ_k are only computed whenever needed and forgotten once all the states associated to the corresponding job have interacted with all their successors.

Now it is possible to extend instances of problem $1|r_j, \bar{d}_j|L_{max}$. Due dates d_j are given on top of deadlines $\bar{d}_j$ and we aim to minimize maximum lateness $L_{max} = max_{j\in\mathcal{J}}(C_j - d_j)$. We perform a binary search by solving a series of decision problems with our makespan algorithm. Deadlines are updated accordingly to the current maximum lateness threshold in the binary search. This means that the processing phase must be done before each step. Finally note that we require deadlines to be given in the input on top of due dates. Indeed the value of pathwidth μ may change at each step of the search, but it can only go down if the instance given in the input is feasible (thus checked in the first step of the binary search). So such initial deadlines allow us to bound the value of the parameter in every step by its initial value.

Corollary 53. *$1|r_j, \bar{d}_j|L_{max}$ can be solved in time $\mathcal{O}(\mu \cdot log(\mu) \cdot 4^{\mu} \cdot n \cdot log(D) + n \cdot log(n) \cdot log(D))$ where $D = \max_j \bar{d}_j$.*

Note that precedence relations may be added to the instance with little extra cost. But because we have both time windows constraints and precedence relations in our problem, first we must ensure that the time constraints are compatible with partial order *prec* (which we denote $\rightarrow$ here).

Definition 54. *An instance $I = \langle \mathcal{J}, prec, p_j, r_j, \bar{d}_j \rangle$ of $1|prec, r_j, \bar{d}_j|C_{max}$ is called prec-consistent when:*

$$\forall (i,j) \in \mathcal{J}^2, (i \rightarrow j) \implies [(r_i + p_i \leq r_j) \wedge (\bar{d}_i \leq \bar{d}_j - p_j)].$$

Given an instance of $1|prec, r_j, \bar{d}_j|C_{max}$ which is not *prec*-consistent, its release times and deadlines can be adjusted in time $\mathcal{O}(n^2)$ to fulfill this property by using path algorithms. Note that by doing so we only remove time window sections which are inaccessible according to precedence relations. So feasibility and the optimal makespan value are unchanged. Once we get a *prec*-consistent instance equivalent to the original instance, the following property holds.

Lemma 55. *On any prec-consistent instance of $1|prec, r_j, \bar{d}_j|C_{max}$ sorting jobs by nondecreasing deadlines is a topological sorting of prec.*

Proof. Direct consequence of Definition 54. □

This means that in every schedule $\gamma_1 \dots \gamma_{K-1}$ from Lemma 47 there is no inner precedence conflict in any γ_k $(1 \leq k \leq K-1)$. It also means that there is no precedence conflict from any job in γ_{k+1} to any job in Z_k. Let $(1, L_1) \to (\dots) \to (1, L_{K-1})$ be the corresponding state path. By Lemma 49 this implies that some precedence conflict could only be from some job in γ_{k+1} to some job in L_k. Since both set sizes are bounded by μ according to Lemma 52, precedence checks can be completed in time $\mathcal{O}(\mu^2)$ every time we attempt to extend a state (k, L_k) to some potential successor $(k+1, L_{k+1})$.

Corollary 56. *$1|prec, r_j, \bar{d}_j|C_{max}$ can be solved in time $\mathcal{O}(\mu^2 \cdot 4^\mu \cdot n + n^2)$. $1|prec, r_j, \bar{d}_j|L_{max}$ can be solved in time $\mathcal{O}(\mu^2 \cdot 4^\mu \cdot n \cdot log(D) + n^2 \cdot log(D))$ where $D = \max_j \bar{d}_j$..*

4.3 Hardness Results with Precedence Delays

In the light of the results obtained in the previous section and in the literature [MK21, KT21, TCH+23, HMK23] parameter μ can lead to FPT algorithms while featuring precedence relations and putting no restriction on them. As a result we thought of delay values to the precedence relations and investigate when the problems would remain FPT parameterized by μ. Unfortunately we show that parameter μ becomes ineffective in the presence of precedence delays, even on single machine scheduling with time windows and restricted precedence constraints (chains, coupled tasks).

In this section we focus on $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$. We show that this problem is para-NP-hard parameterized by μ for each of the three delay types (exact, minimum, maximum). This result was first presented during the ROADEF 2022 conference [Mal22] in the case of exact/minimum delays. Unlike with previous parameterized reductions, we do need to reduce from a parameterized problem:

Claim 57. *[FG98] If a (nontrivial) problem $\mathcal{P}$ is NP-hard with a fixed value of some parameter k, then the parameterized problem $(\mathcal{P}, k)$ is para-NP-hard.*

As a result for all three delay types we show that $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ is NP-hard with a fixed value of μ. All reductions start from the (strongly) NP-hard 3-COLORING graph problem [Kar72]. Let $G = (V, E)$ be the input graph (with no self-loops). Let $v_0, \dots, v_{n-1}$ be the vertices in V and $e_0, \dots, e_{m-1}$ be the edges in E. Let $n = |V|$ and $m = |E|$. The colors are named 0, 1 and 2.

Note that by restricting further to coupled tasks (i.e. chains of length 2) and a single available delay value the problem remains para-NP-hard parameterized by μ. The corresponding reductions are given in Section 5.3. They use the same base ideas as the reductions in this section but with more sophisticated gadgets. So we strongly recommend that you read this section before you tackle Section 5.3.

4.3.1 With Exact Delays

We start by proving the result with exact delays.

Theorem 58. $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ *with exact delays is para-*NP*-hard parameterized by pathwidth* μ*.*

We build an instance I_{ex} of $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ where $\mu = 2$. An example is given in Figure 4.2. We have n *vertex chains* $\mathcal{C}_i$ with $deg(v_i)+1$ jobs in chain $\mathcal{C}_i$, $0 \leq i \leq n-1$ with $deg(v_i)$ the degree of node v_i in G. We define vertex chain $\mathcal{C}_i$ the following way:

Definition 59 (Vertex chain $\mathcal{C}_i$)**.** *We segment time into* $m+1$ *segments: a color choice segment* $[0, 3n)$ *and* m *edge check selection segments of length* 3 *along* $[3n, 3(n+m))$*. We describe the chain from left to right:*

1. *Color choice segment* $[0, 3n)$
   - *The first job of chain* $\mathcal{C}_i$ *has time window* $[3i, 3(i+1))$*.*
     (a) *If* v_i *appears in no edge of* G*: end the chain.*
     (b) *Else: set* $3(n-i)-1$ *as the current exact delay after this job.*
2. *Edge check segment* $[3(n+j), 3(n+j+1))$, $0 \leq j \leq m-1$
   For j *in* $[0, m-1]$*:*
   Let edge $e_j = \{v_{i_1}, v_{i_2}\}, i_1 < i_2$*.*
   (a) *Vertex chain* $\mathcal{C}_i$ *with* $i \notin \{i_1, i_2\}$
      - *Add* 3 *to the current exact delay after the currently latest job of chain* $\mathcal{C}_i$*.*
   (b) *Vertex chain* $\mathcal{C}_i$ *with* $i = i_1$ *or* $i = i_2$
      - *Set a job with time window* $[3(n+j), 3(n+j+1))$
        i. *If* e_j *is the last edge where* v_i *appears: end the chain.*
        ii. *Else: set* 2 *as the current exact delay after this job.*

Remark 60. *The created instance has pathwidth* 2*. In the color choice segment: for* $i \in [0, n-1]$ *there is exactly one job to be scheduled in time window* $[3i, 3(i+1))$*: the first job of vertex chain* $\mathcal{C}_i$*. In the edge check segments: for* $j \in [0, m-1]$ *if edge* $e_j = \{v_{i_1}, v_{i_2}\}$ *then there are exactly two jobs to be scheduled in time window* $[3(n+j), 3(n+j+1))$*: one from vertex chain* $\mathcal{C}_{i_1}$ *and one from vertex chain* $\mathcal{C}_{i_2}$*. Thus there are indeed at most two overlapping time windows at any given time.*

Let $i \in [0, n-1]$. Vertex chain $\mathcal{C}_i$ has three possible starting times in $[3i, 3(i+1))$ which corresponds to the three color choices of node v_i. Then this color choice is propagated to every edge check segment $[3(n+j), 3(n+j+1))$ where node v_i is a part of edge e_j, $j \in [0, m-1]$. The following lemma ensures that the color choices are faithfully propagated.

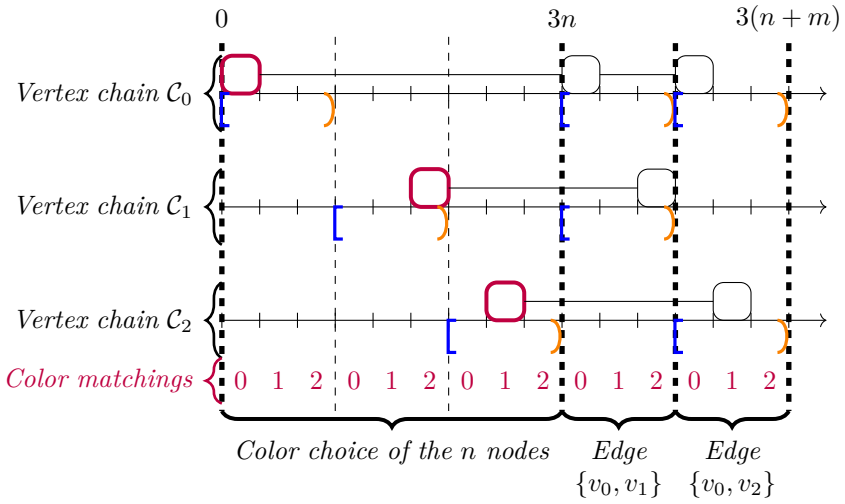


Figure 4.2: An instance of $1|chains(\ell_{ij}), p_j = 1, r_j, d_j|\star$ with exact delays representing a graph coloring. We have $G = (V, E)$ with $V = \{v_0, v_1, v_2\}$ and $E = (\{v_0, v_1\}, \{v_0, v_2\})$. This schedule corresponds to coloring $(0, 2, 1)$.

Lemma 61. *Let $0 \leq i \leq n-1$. In any feasible schedule of I_{ex}, if vertex chain $\mathcal{C}_i$ starts at time $3i + k$ with $k \in \{0, 1, 2\}$, then all jobs J in this chain are scheduled at time $r(J) + k$, where $r(J)$ is the release date of job J.*

Proof. Suppose we have a feasible schedule where vertex chain $\mathcal{C}_i$ starts at time $3i + k$ with $k \in \{0, 1, 2\}$.

- If vertex v_i is part of no edge in the graph:

  Then by Definition 59 vertex chain $\mathcal{C}_i$ only has one job. It is scheduled at time $3i + k$, which is indeed the release date of this job plus k.

- If vertex v_i is part of at least one edge in the graph:

  Let $j_0 < j_1 < \ldots < j_{deg(v_i)-1}$ be the indices of the edges e_j such that $v_i \in e_j$. We prove by induction on $l \in [0, deg(v_i) - 1]$ that the job of vertex chain $\mathcal{C}_i$ which has time window $[3(n + j_l), 3(n + j_l + 1))$ is scheduled at time $3(n + j_l) + k$.

  - Consider the job of vertex chain $\mathcal{C}_i$ which has time window $[3(n + j_0), 3(n+j_0+1))$. By Definition 59 this job is the successor of the first job in the chain and the exact delay between them is $3(n-i)-1+3j_0$. Thus if the first job of the chain is scheduled at time $3i+k$, then this following job is scheduled at time $(3i+k)+1+(3(n-i)-1+3j_0) = 3(n + j_0) + k$, which is indeed the release date of this job plus k.
  - Let $l \in [1, deg(v_i) - 1]$. Suppose the job of vertex chain $\mathcal{C}_i$ which has time window $[3(n + j_{l-1}), 3(n + j_{l-1} + 1))$ is scheduled at time

$3(n+j_{l-1})+k$. Then by Definition 59 there is an exact delay $2+3(j_l-j_{l-1}-1)$ before the next job of the chain. Thus the next job of the chain is scheduled at time $(3(n+j_{l-1})+k)+1+(2+3(j_l-j_{l-1}-1)) = 3(n+j_l)+k$, which is indeed the release date of this job plus k.

This proves the lemma for all the jobs of vertex chain $\mathcal{C}_i$ in an edge check segment. □

Then for each edge $e_j = \{v_{i_1}, v_{i_2}\}, i_1 < i_2$, the color choices of v_{i_1} and v_{i_2} are confronted in edge check segment $[3(n+j), 3(n+j+1))$. If both nodes chose the same color then both jobs in this edge check segment would be scheduled at the same time, which would invalidate our schedule in this single-machine instance. Conversely if we start from a valid coloring, then there will never be two jobs scheduled at the same time in an edge check segment. This is the key ingredient behind the reduction.

Proposition 62. *G is 3-colorable if and only if there exists a feasible schedule for I_{ex}.*

Proof. ($\Longrightarrow$) Suppose we have $(c_0, \ldots, c_{n-1}) \in \{0,1,2\}^n$ a 3-coloring of G where vertex v_i has color c_i. We propose the schedule where for all $0 \leq i \leq n-1$, chain $\mathcal{C}_i$ starts at time $3i + c_i$. Then in every edge $e_j \in E$ where vertex v_i appears, we schedule the job of chain $\mathcal{C}_i$ which is in edge check segment $[3(n+j), 3(n+j+1))$ at time $3(n+j) + c_i$.

We show that the jobs in different chains do not interfere with each other. Since the time windows do not overlap in the color choice segment, only the edge check segments remain to be checked. Let $e_j = \{v_{i_1}, v_{i_2}\}$ be an edge in E. By definition of the vertex chains, only chains $\mathcal{C}_{i_1}$ and $\mathcal{C}_{i_2}$ have a job to be scheduled in time window $[3(n+j), 3(n+j+1))$. In our schedule the job of chain $\mathcal{C}_{i_1}$ is scheduled at time $3(n+j) + c_{i_1}$ and the job of chain $\mathcal{C}_{i_2}$ at time $3(n+j) + c_{i_2}$. Since $(c_0, \ldots, c_{n-1})$ is a 3-coloring and $\{v_{i_1}, v_{i_2}\} \in E$, we have $c_{i_1} \neq c_{i_2}$. Thus both jobs are scheduled at different times and the jobs in edge check segment $[3(n+j), 3(n+j+1))$ do not interfere with each other. Thus the proposed schedule is feasible.

($\Longleftarrow$) Suppose we have a feasible schedule. For all $0 \leq i \leq n-1$, let $s_i \in \{0,1,2\}$ be such that $3i + s_i$ is the starting time of chain $\mathcal{C}_i$. We show that $(s_0, \ldots, s_{n-1})$ is a 3-coloring of G. By contradiction suppose there is an edge $e_j = \{v_{i_1}, v_{i_2}\} \in E$ such that $s_{i_1} = s_{i_2}$. Then by Lemma 61 the jobs of chains i_1 and i_2 that must be scheduled in edge check segment $[3(n+j), 3(n+j+1))$ are scheduled at the same time $3(n+j) + s_{i_1}$. Thus the schedule is not feasible, which leads to a contradiction. Thus $(s_0, \ldots, s_{n-1})$ is indeed a 3-coloring of G. □

This proves that $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ with exact delays is NP-hard when $\mu = 2$, which concludes the para-NP-hardness proof of the corresponding parameterized problem.

4.3.2 With Minimum Delays

We adapt the reduction to instances with minimum delays:

Theorem 63. $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ *with minimum delays is para-*NP*-hard parameterized by pathwidth* μ*.*

We build an instance I_{min} of $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ with $\mu = 3$. We begin in a similar way: for each node we have a vertex chain $\mathcal{C}'_i$ with three possible starting times, each corresponding to a color choice, then we want to propagate this color choice. However now that the delays are not exact anymore, the color choice cannot be propagated properly as previously. More constraints are needed in order to deal with the extra flexibility coming from the minimum delays.

One way is to add a closing segment $[3(n+m), 3(2n+m))$ at the end and two gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ per node, each composed of two jobs. As shown in 4.3 the gadget chains will fill the two gaps at the start and at the end of each vertex chain.

Definition 64 (Vertex chain $\mathcal{C}'_i$)**.** *We segment time into* $m+2$ *segments: a color selection segment* $[0, 3n)$*,* m *edge check selection segments along* $[3n, 3(n+m))$ *and a closing segment* $[3(n+m), 3(2n+m))$*. We describe the chain from left to right:*

1. *Color selection segment* $[0, 3n)$
   - *The first job of chain* $\mathcal{C}'_i$ *has time window* $[3i, 3(i+1))$*.*
   - *Set* $3(n-i)-1$ *as the current minimum delay after this job.*
2. *Edge check segment* $[3(n+j), 3(n+j+1))$*,* $0 \leq j \leq m-1$

   For j *in* $[0, m-1]$*:*

   Let edge $e_j = \{v_{i_1}, v_{i_2}\}, i_1 < i_2$*.*

   (a) *Vertex chain* $\mathcal{C}'_i$ *with* $i \notin \{i_1, i_2\}$
      - *Add* 3 *to the current minimum delay after the currently latest job of chain* $\mathcal{C}'_i$

   (b) *Vertex chain* $\mathcal{C}'_i$ *with* $i = i_1$ *or* $i = i_2$
      - *Set a job with time window* $[3(n+j), 3(n+j+1))$
      - *Set* 2 *as the current minimum delay after this job*
3. *Closing segment* $[3(n+m), 3(2n+m))$
   - *Add* $3i$ *to the current minimum delay of the currently latest job of chain* $\mathcal{C}'_i$*.*
   - *Set a job with time window* $[3(n+i+m), 3(n+i+1+m))$ *as the last job of vertex chain* $\mathcal{C}'_i$*.*

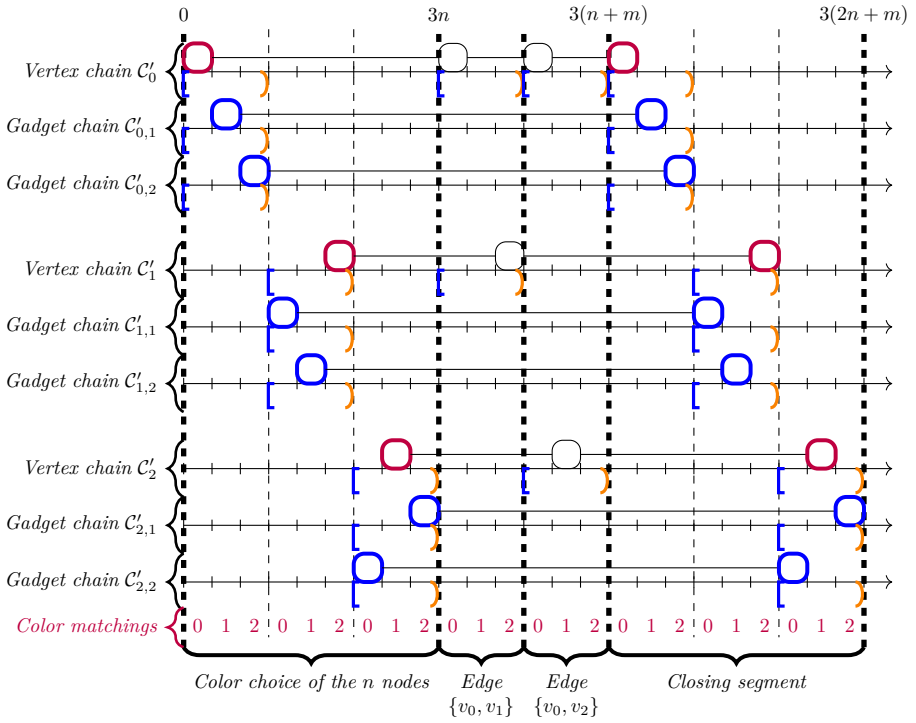


Figure 4.3: An instance of $1|chains(\ell_{ij}), p_j = 1, r_j, d_j|\star$ with minimum delays representing a graph coloring. We have $G = (V, E)$ with $V = \{v_0, v_1, v_2\}$ and $E = (\{v_0, v_1\}, \{v_0, v_2\})$. This schedule corresponds to coloring $(0, 2, 1)$.

Definition 65 (Gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$)**.** *For both gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ relative to vertex v_i, the first job must be scheduled in time window $[3i, 3(i+1))$, the second one in time window $[3(n+m+i), 3(n+m+i+1))$, and there is a minimum delay $3(n+m)-1$ between them.*

Remark 66. *The created instance has indeed pathwidth 2: gadget chains add two more time windows at the beginning and the end of each vertex chain, so there are at most three time windows overlapping at any time.*

A full example is given in Figure 4.3. For our proof the goal is to show that adding these gadget chains is enough to get an analogue result to Lemma 61. Note that if we only consider the n vertex chains like in the reduction of Section 4.3.1, we only get this weaker result:

Lemma 67. *Let $0 \leq i \leq n-1$. In any feasible schedule of I_{min}, if a chain starts at time $3i+k$ with $k \in \{0, 1, 2\}$, then all jobs J in this chain are scheduled at time $r_J + k$ or later.*

Proof. For gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ this comes from the minimum delay $3(n+m)-1$ between their two jobs. For vertex chain $\mathcal{C}'_i$: suppose we have a feasible schedule where vertex chain $\mathcal{C}'_i$ starts at time $3i+k$ with $k \in \{0,1,2\}$.

- If vertex v_i is part of no edge in the graph:

  Then by 64 vertex chain $\mathcal{C}'_i$ only has two jobs and there is a minimum delay $3(n-i)-1+3m+3i = 3(n+m)-1$ between them. Thus if the first job is scheduled at time $3i+k$, then the job in the closing segment is scheduled at time $(3i+k)+1+(3(n+m)-1) = 3(n+m+i)+k$, which is indeed the release date of this job plus k.

- If vertex v_i is part of at least one edge in the graph:

  Let $j_0 < j_1 < \ldots < j_{deg(v_i)-1}$ be the indices of the edges e_j such that $v_i \in e_j$. We prove the lemma for all the jobs of vertex chain $\mathcal{C}'_i$ in an edge check segment by induction on $l \in [0, deg(v_i)-1]$ the same way as in 61. Then only the job of the chain in the closing segment is left. By 64 there is a minimum delay $2+3(m-1-j_{deg(v_i)-1})+3i$ between this job and the one in edge check segment $[3(n+j_{deg(v_i)-1}), 3(n+j_{deg(v_i)-1}+1))$. Since we know from the induction that the latter job is scheduled at time $3(n+j_{deg(v_i)-1})+k$ or later, this means that the job of vertex chain $\mathcal{C}'_i$ in the closing segment is scheduled at time $(3(n+j_{deg(v_i)-1})+k)+1+(2+3(m-1-j_{deg(v_i)-1})+3i) = 3(n+m+i)+k$ or later, which is indeed the release date of this job plus k.

□

By taking into account the constraints added by the gadget chains at the beginning and at the end of each vertex chain, we are able to prove the needed key property.

Lemma 68. *In any feasible schedule of I_{min}, if a vertex chain $\mathcal{C}'_i$ starts at time $3i+k$ with $k \in \{0,1,2\}$, then all jobs J in vertex chain $\mathcal{C}'_i$ which are in an edge check segment have to be scheduled at time $r_J + k$.*

Proof. Consider a feasible schedule. Let $0 \leq i \leq n-1$. Three jobs are scheduled in time window $[3i, 3(i+1))$: the first job of vertex chain $\mathcal{C}'_i$ and the first job of the two gadget chains relative to it. Let $3i+k$ (resp. $3i+k'_1$, $3i+k'_2$) be the starting time of the first job of vertex chain $\mathcal{C}'_i$ (resp. gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$). We have k, k'_1 and k'_2 in $\{0,1,2\}$ and since we have a feasible schedule the three values are different from each other. Thus one chain starts at time $3i+2$. By Lemma 67 and the time window $[3(n+m+i), 3(n+m+i+1))$ of the last job, this means that this last job must be exactly scheduled at time $3(n+m+i)+2$. Now consider the chain which starts at time $3i+1$. By Lemma 67 its last job must be scheduled at time $3(n+m+i)+1$ or $3(n+m+i)+2$. However the later time position is already taken by the chain starting at time $3i+2$, which means that this last job must be exactly scheduled at time $3(n+m+i)+1$. Finally by the same reasoning we get that the chain starting at time $3i$ must have its last job scheduled at time $3(n+m+i)$.

This means that whatever the starting time $3i + k$ is for vertex chain $\mathcal{C}'_i$, its last job must be scheduled at time $3(n + m + i) + k$. So all the delays in the chain must be equal to their minimum and thus by Lemma 67 all the jobs J in the vertex chain must be scheduled at time $r(J) + k$. □

Now in any feasible schedule we proved that we have the same guarantee on the position of the edge check jobs as we had with Lemma 61 in the exact delay case. Thus we are able to propagate the color choices accurately and complete the reduction the same way.

Proposition 69. *G is 3-colorable if and only if there exists a feasible schedule for I_{min}.*

Proof. ($\Longrightarrow$) Suppose we have $(c_0, \ldots, c_{n-1}) \in \{0, 1, 2\}^n$ a 3-coloring of G where vertex v_i has color c_i. We propose a schedule where for all $0 \leq i \leq n-1$, vertex chain $\mathcal{C}'_i$ starts at time $3i + c_i$ and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ start in the two remaining time positions $3i + k_1, 3i + k_2$ in $[3i, 3(i + 1))$ (with $k_1 \neq k_2$). Plus we require all delays to match their minimum value.

Then, according to Definition 64 and going from left to right as we did in the proof of Lemma 67, we know that in every edge $e_j \in E$ where node v_i appears, the job of vertex chain $\mathcal{C}'_i$ which is in edge check segment $[3(n+j), 3(n+j+1))$ is scheduled at time $3(n + j) + c_i$, and the last job of $\mathcal{C}'_i$ is scheduled at time $3(n + m + i) + c_i$. Plus from Definition 65 we know that the last job of gadget chain $\mathcal{C}'_{i,1}$ (resp. $\mathcal{C}'_{i,2}$) is scheduled $3(n + mi) + k_1$ (resp. $3(n + m + i) + k_2$).

We show that the jobs in different chains do not interfere with each other. For the color choice segment we know that vertex chain $\mathcal{C}'_i$ and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ start respectively at times $3i + c_i, 3i + k_1$, and $3i + k_2$ with c_i, k_1 and k_2 in $\{0, 1, 2\}$ and different from each other. For the closing segment we determined that vertex chain $\mathcal{C}'_i$ and gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ end respectively at times $3(n + m + i) + c_i, 3(n + m + i) + k_1$, and $3(n + m + i) + k_2$, again with c_i, k_1 and k_2 in $\{0, 1, 2\}$ and different from each other. Thus only the edge check segments remain to be checked. Let $e_j = \{v_{i_1}, v_{i_2}\}$ be an edge in E. By definition of the vertex chains, only vertex chains $\mathcal{C}'_{i_1}$ and $\mathcal{C}'_{i_2}$ have a job to be scheduled in time window $[3(n + j), 3(n + j + 1))$. In our schedule the job of chain $\mathcal{C}i_1$ is scheduled at time $3(n + j) + c_{i_1}$ and the job of chain $\mathcal{C}i_2$ at time $3(n + j) + c_{i_2}$. Since $(c_0, \ldots, c_{n-1})$ is a 3-coloring and $\{v_{i_1}, v_{i_2}\} \in E$, we have $c_{i_1} \neq c_{i_2}$. Thus both jobs are scheduled at different times and the jobs in edge check segment $[3(n + j), 3(n + j + 1))$ do not interfere with each other. Thus the proposed schedule is feasible.

($\Longleftarrow$) Suppose we have a feasible schedule. We reuse the same coloring as in the proof of Proposition 62: for all $0 \leq i \leq n-1$, let $s_i \in \{0, 1, 2\}$ be such that $3i+s_i$ is the starting time of chain $\mathcal{C}'_i$. We show that $(s_0, \ldots, s_{n-1})$ is a 3-coloring of G. Considering any edge $e_j = \{v_{i_1}, v_{i_2}\} \in E$, Lemma 68 ensures that the job of vertex chain $\mathcal{C}'_{i_1}$ (resp. $\mathcal{C}'_{i_2}$) in edge check segment $[3(n + j), 3(n + j + 1))$ is scheduled at time $3(n + j) + s_{i_1}$ (resp. $3(n + j) + s_{i_2}$), with s_{i_1} (resp. s_{i_2}) the starting time of vertex chain $\mathcal{C}'_{i_1}$ (resp. $\mathcal{C}'_{i_2}$). Since this is a feasible schedule we

have: $3(n+j)+s_{i_1} \neq 3(n+j)+s_{i_2}$, which means: $s_{i_1} \neq s_{i_2}$. Thus the two nodes of edge e_j have indeed different colors. □

This proves that $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ with minimum delays is NP-hard when $\mu = 3$, which concludes the para-NP-hardness proof of the corresponding parameterized problem.

4.3.3 With Maximum Delays

We adapt the reduction to instances featuring maximum delays:

Theorem 70. *$1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ with maximum delays is para-NP-hard parameterized by pathwidth μ.*

We build an instance I_{max} of $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ with $\mu = 3$. We define I_{max} as the instance I_{min} from the previous reduction, except we have maximum delays instead of minimum ones. It turns out that gadget chains $\mathcal{C}'_{i,1}, \mathcal{C}'_{i,2}$ work in a symmetric way. Indeed in both the minimum and maximum delay cases they block the first and last task of each vertex chain in the same fashion. This allows us to keep the key property behind the reduction.

Lemma 71. *In any feasible schedule of I_{max}, if a vertex chain $\mathcal{C}'_i$ starts at time $3i+k$ with $k \in \{0,1,2\}$, then all jobs J in vertex chain $\mathcal{C}'_i$ which are in an edge check segment have to be scheduled at time $r_J + k$.*

Proof. (sketch) This is proved the same way as Lemma 68. □

Now in any feasible schedule we proved that we have the same guarantee on the position of the edge check jobs as we had with Lemma 61 in the exact delay case. Thus we are able to propagate the color choices accurately and complete the reduction the same way.

Proposition 72. *G is 3-colorable if and only if there exists a feasible schedule for I_{min}.*

Proof. (sketch) This is proved the same way as Proposition 69. □

This proves that $1|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ with maximum delays is NP-hard when $\mu = 3$, which concludes the para-NP-hardness proof of the corresponding parameterized problem.

4.4 Combining with Maximum Delay Value ℓ_{max}

In this section we combine pathwidth μ with maximum delay value ℓ_{max} as a parameter. We show that this makes the problem from the previous section FPT even on multiple parallel machines and general precedence.

Theorem 73. *With minimum delays $P|prec(\ell_{i,j}), p_j = 1, r_j, d_j|\star$ is FPT parameterized by $\mu + \ell_{max}$.*

Let us consider the sorted list x_k, $k \in \{0, \ldots, K\}$ of the release times and deadlines in non decreasing order. We define a sub-sequence u_α, $\alpha \in \{0, \ldots, \kappa - 1\}$ of this sequence so that two consecutive terms - except the last one - are separated by at least ℓ_{max}. So we set $u_0 = x_0$, then $u_{\alpha+1} = x_k$ with k the minimum value in $\{1, \ldots, K\}$ such that $u_{\alpha+1} - u_\alpha \geq \ell_{max}$. Lastly we set $u_\kappa = x_K$.

Let us consider the instance with minimum delays described in Figure 4.4. For this instance we get the sequence $x_0 = 0$, $x_1 = 2$, $x_2 = 4$, $x_3 = 6$, $x_5 = 9$ and $x_6 = 11$ with $K = 6$. The associated pathwidth $\mu = 3$ is reached in the interval $[4, 6)$ crossed by intervals of jobs $3, 6, 7, 8$. We also have $\ell_{max} = 3$, so we get: $u_0 = x_0 = 0$, $u_1 = x_2 = 4$, $u_2 = x_5 = 9$ and $u_3 = x_6 = 11$.

We set $X_0 = \emptyset$ and for any $\alpha \in \{1, \ldots \kappa\}$ we define $X_\alpha = \{i \in \mathcal{T}, [r_i, d_i) \cap [u_{\alpha-1}, u_\alpha) \neq \emptyset\}$ the set of jobs that could be scheduled in interval $[u_{\alpha-1}, u_\alpha)$. The idea of sequence $(u_0, \ldots, u_\kappa)$ is that the number of jobs in each X_α is bounded by $\mu \cdot \ell_{\max}$ (see Lemma 75). We also define Z_α, $\alpha \in \{0, \ldots, \kappa\}$ the set of jobs with a deadline not greater than u_α, *i.e.* $Z_\alpha = \{i \in \mathcal{T}, d_i \leq u_\alpha\}$. In our example we have: $Z_0 = \emptyset$, $Z_1 = \{1, 2, 4, 5\}$, $Z_2 = Z_1 \cup \{3, 6, 7, 8, 9\}$ and $Z_3 = \mathcal{T}$.

We define a dynamic programming scheme for our problem. The stages of the scheme are $\{0, \ldots, \kappa\}$. For each stage $\alpha \in \{0, \ldots, \kappa\}$ we denote N_α the set of states of stage α. A state $s \in N_\alpha$ represents the minimum information from a feasible schedule spanning in $[0, u_\alpha)$ that is necessary to extend this schedule in interval $[u_\alpha, u_\kappa)$.

Hence a state $s \in N_\alpha$ with $\alpha \in \{1, \ldots, \kappa - 1\}$ is a tuple $s = (\beta, Y)$, where:

- $Y \subseteq X_\alpha - Z_\alpha$ is a subset of jobs such that $Y \cup Z_\alpha$ represents the set of jobs scheduled in $[0, u_\alpha)$.
- β is a complete schedule (*i.e* a set of jobs with their starting time) of the last ℓ_{max} time units before time u_α- i.e in interval $[u_\alpha - \ell_{max}, u_\alpha)$. We denote $J(\beta)$ the set of jobs scheduled in β. β is called a *border schedule*. Only such a schedule can influence the earliest starting times of the jobs not scheduled yet.

For $\alpha = \kappa$ we set $N_\kappa = \{s_\kappa\}$ with $s_\kappa = (\bullet, \emptyset)$ where $\bullet$ is an empty schedule. Moreover $X_\kappa - Z_\kappa = \emptyset$ and so N_κ can be reduced to only one element. Similarly, we set $N_0 = \{s_0\}$ with $s_0 = (\bullet, \emptyset)$ since $X_0 = \emptyset$ and no job may be executed in interval $[u_0, u_0) = \emptyset$.

As an example let us consider a feasible schedule τ pictured by Figure 4.5 for $m = 2$ identical machines associated to the instance given by Figure 4.4. The associates states are: $s_0 = (\bullet, \emptyset)$, $s_1 = (\beta_1, \{3\})$, $s_2 = (\beta_2, \emptyset)$ and $s_3 = (\bullet, \emptyset)$.

Now assume that $s = (\beta, Y) \in N_\alpha$ with $\alpha \in \{0, \ldots, \kappa\}$. The boolean function ExistSched(s) is set to *true* if and only if there exists a (partial) feasible schedule of jobs from $Y \cup Z_\alpha$ in time interval $[u_0, u_\alpha)$ that ends with schedule β.

We can now establish the recurrence equation for this function:

1. ExistSched(s_0) = *true*; indeed, $s_0 = (\bullet, \emptyset)$ and $Z_0 = \emptyset$, thus no job has to be scheduled.

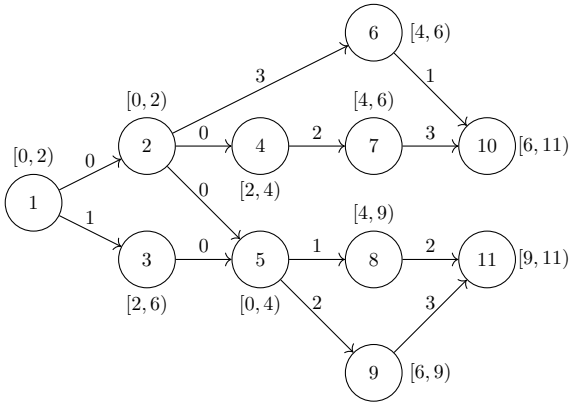


Figure 4.4: A precedence graph with minimum delays. Each precedence arc $e = (i, j)$ is labeled by the minimum delay ℓ_{ij}. Each node i is labelled by its time window $[r_i d_i)$.

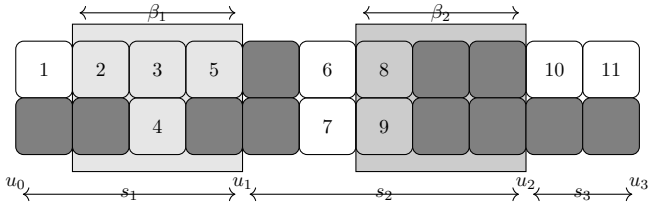


Figure 4.5: A feasible schedule σ associated with the example given in Figure 4.4 for $m = 2$ machines

2. Let us now consider $\alpha \in \{1, \ldots \kappa\}$. If ExistSched$(s) = true$ then there exists a feasible schedule in $[0, u_\alpha)$ which can be decomposed into a feasible schedule in $[0, u_{\alpha-1})$ associated with a state $s' \in N_{\alpha-1}$ and a schedule in the interval $[u_{\alpha-1}, u_\alpha)$ consistent with s and s'. The existence of such a schedule is denoted by the function Sched(s, s').

We now bound the complexity of computing Sched(s, s') from a tuple of states $(s', s) \in N_{\alpha-1} \times N_\alpha$.

Let $s = (\beta, Y)$ and $s' = (\beta', Y')$. Then boolean Sched(s', s) is true if and only if there exists a schedule of $Y \cup Z_\alpha - Y' - Z_{\alpha-1}$ in the interval $[u_{\alpha-1}, u_\alpha)$ that is consistent with the border schedule β' and ends with the border schedule β.

Lemma 74. *For any $\alpha \in \{1, \ldots, \kappa\}$ and $(s', s) \in N_{\alpha-1} \times N_\alpha$, the time complexity of Sched(s', s) is $\mathcal{O}(\mu^2 \cdot \ell_{max}^2 \cdot (\mu \cdot \ell_{max})!)$.*

To prove this lemma, two more technical lemmas are needed. These lemmas bound the total number of candidate jobs in time interval $[u_{\alpha-1}, u_\alpha)$ for $\alpha \in \{1, \ldots \kappa\}$:

Lemma 75. $\forall \alpha \in \{0, \ldots, \kappa\}, |X_\alpha| \leq \mu \cdot \ell_{\max}$.

Proof. We simply observe that, by the definition of u_α, there are at most $\ell_{max}-1$ release date/deadline values between $u_{\alpha-1}$ and u_α. Thus by the definition of μ the inequality holds. □

Lemma 76. *For any* $\alpha \in \{1, \ldots, \kappa - 1\}$, $|N_\alpha| \leq 2^{\mu \cdot \ell_{max}} \cdot (\ell_{max} + 1)^{\mu \cdot \ell_{max}}$.

Proof. The total number of schedules from a set $V = J(\beta)$ is bounded by $(\ell_{max} + 1)^{|V|}$. Thus, by Lemma 75, it is bounded by $(\ell_{max} + 1)^{\mu \cdot \ell_{max}}$. And because the number of sets $V \subseteq X_\alpha$ is bounded by $2^{|X_\alpha|} \leq 2^{\mu \cdot \ell_{max}}$, the lemma holds. □

Now we are able to prove Lemma 74:

Proof. The problem is to schedule jobs from $S = Y \cup Z_\alpha - (Y' \cup Z_{\alpha-1})$ in the interval $[u_{\alpha-1}, u_\alpha)$ so that the schedule is consistent with the two border schedules β' and β This can be done in several steps:

1. adjusting the release times of jobs of S with respect to the border schedule β': if j is a successor of $i \in J(\beta')$ then $r_j = \max(r_j, \beta'(i) + 1 + \ell_{i,j})$, and propagate to precedence constraints in S.

2. adjusting the deadlines of jobs of S with respect to the border schedule β: if i is a predecessor of $j \in J(\beta)$ then $d_i = \min(d_i, \beta(j) - \ell_{i,j})$, and propagate to precedence constraints in S

3. if a contradiction is detected at this step (a job j for which $r_j \geq \bar{d}_j$), $\text{Sched}(s', s) = false$.

4. Otherwise we can enumerate all active schedules (i.e. schedules in which no job can be scheduled earlier provided the other jobs are not delayed) of $S - J(\beta)$ and verify that one of them spans in $[u_{\alpha-1}, u_\alpha - \ell_{\max})$

The time complexity of the two first steps is $\mathcal{O}(|S|^2)$. For the last step it is known that any active schedule can be generated by list scheduling using a permutation of jobs [Sch70]. Thus the enumeration of active schedules can be done by a brute force algorithm that enumerates all permutations of jobs and then performs a list scheduling algorithm to check whether the schedule spans in the interval $[u_{\alpha-1}, u_\alpha - \ell_{\max})$.

At most m jobs are executed at each instant, and the number of iterations is bounded by $|S|$. For each iteration, we must check that all the precedence constraints (and associated minimum delays) are fulfilled, and thus one execution of this priority list has a complexity bounded by $\mathcal{O}(|S|^2)$.

The total number of permutations is $|S - J(\beta)|!$. Thus the overall complexity is bounded by $\mathcal{O}(|S|^2 \cdot |S - J(\beta)|!)$. And since $S \subseteq X_\alpha$, by Lemma 75 we get that $|S| \leq \mu \cdot \ell_{max}$ and the lemma holds. □

Finally we formalize the recurrence equation that yields a FPT algorithm when we have minimum delays: if $s \in N_\alpha$,

$$\text{ExistSched}(s) = \bigvee_{s' \in N_{\alpha-1}} \text{Sched}(s', s) \wedge \text{ExistSched}(s') \tag{4.1}$$

Proposition 77. *The answer to an instance I of $P|prec(\ell_{ij}), p_j = 1, r_j, \bar{d}_j|\star$ with minimum delays is "yes" if and only if ExistSched(s_κ) is true. Moreover the time complexity of the computation of ExistSched(s_κ) is:*

$$\mathcal{O}[(2\ell_{max} + 2)^{2\mu \cdot \ell_{max}} \cdot \mu^2 \cdot \ell_{max}^2 \cdot (\mu \cdot \ell_{max})! \cdot n + n \cdot log(n)].$$

Proof. If ExistSched$(s_\kappa) = true$, then a sequence of states $s_0, s_1, \ldots s_\kappa$ with $s_\alpha \in N_\alpha$ for $\alpha \in \{0, \ldots, \kappa\}$ and Sched$(s_{\alpha-1}, s_\alpha) = true$ for all $\alpha \in \{1, \ldots, \kappa\}$ can be built. And conversely such a sequence induces a feasible schedule.

Now, the number of calls of the function Sched necessary to compute the recurrence equation (4.1) is proportional to $\sum_{\alpha=1}^{\kappa} |N_{\alpha-1}| \cdot |N_\alpha|$. By Lemma 76, this value is bounded by $\kappa \cdot 2^{2\mu \cdot \ell_{max}} \cdot (\ell_{max}+1)^{2\mu \cdot \ell_{max}}$. Since $\kappa \leq 2n$, by Lemma 74 we get the theorem.

□

4.5 Summary & Concluding Remarks

Parameter	Section	Problem / Setting	Result
μ	4.2	$1\|prec, r_j, \bar{d}_j\|C_{max}$	FPT in time $\mathcal{O}(\mu^2 \cdot 4^\mu \cdot n + n^2)$.
	4.3	$1\|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j\|\star$	para-NP-hard for all three delay types.
$\mu + \ell_{max}$	4.4	$P\|prec(\ell_{i,j}), p_j = 1, r_j, \bar{d}_j\|\star$	FPT with minimum delays.

Figure 4.6: Summary of the results obtained in this chapter.

In this chapter we uncovered several capabilities and limitations of pathwidth μ as a parameter for scheduling problems. We proposed a FPT algorithm on single machine scheduling and showed that it worked even in the presence of precedence relations. However when these relations are enhanced with delay values, we proved that the problem becomes para-NP-hard even with chains of unit jobs. Luckily when pairing μ with maximum delay value ℓ_{max} we showed that scheduling unit jobs becomes FPT even on identical parallel machines with general precedence. This confirmed the benefit of adding job time windows to scheduling problems featuring precedence delays.

Despite this success on multiple machines with unit jobs, we note several limitations of parameter μ when upgrading to jobs of arbitrary duration. Even though we gave a FPT algorithm on a single machine in the absence of precedence delays, Hanen and Munier showed that the problem became para-NP-hard with respect to μ on two or more machines [HMK23]. Plus with precedence delays it is open whether the single machine case remains FPT parameterized by $\mu+\ell_{max}$ with jobs of arbitrary duration. This suggests considering a stronger parameter which could cover both settings and provide them with FPT algorithms.

4.5.1 Problem Map with Parameter μ

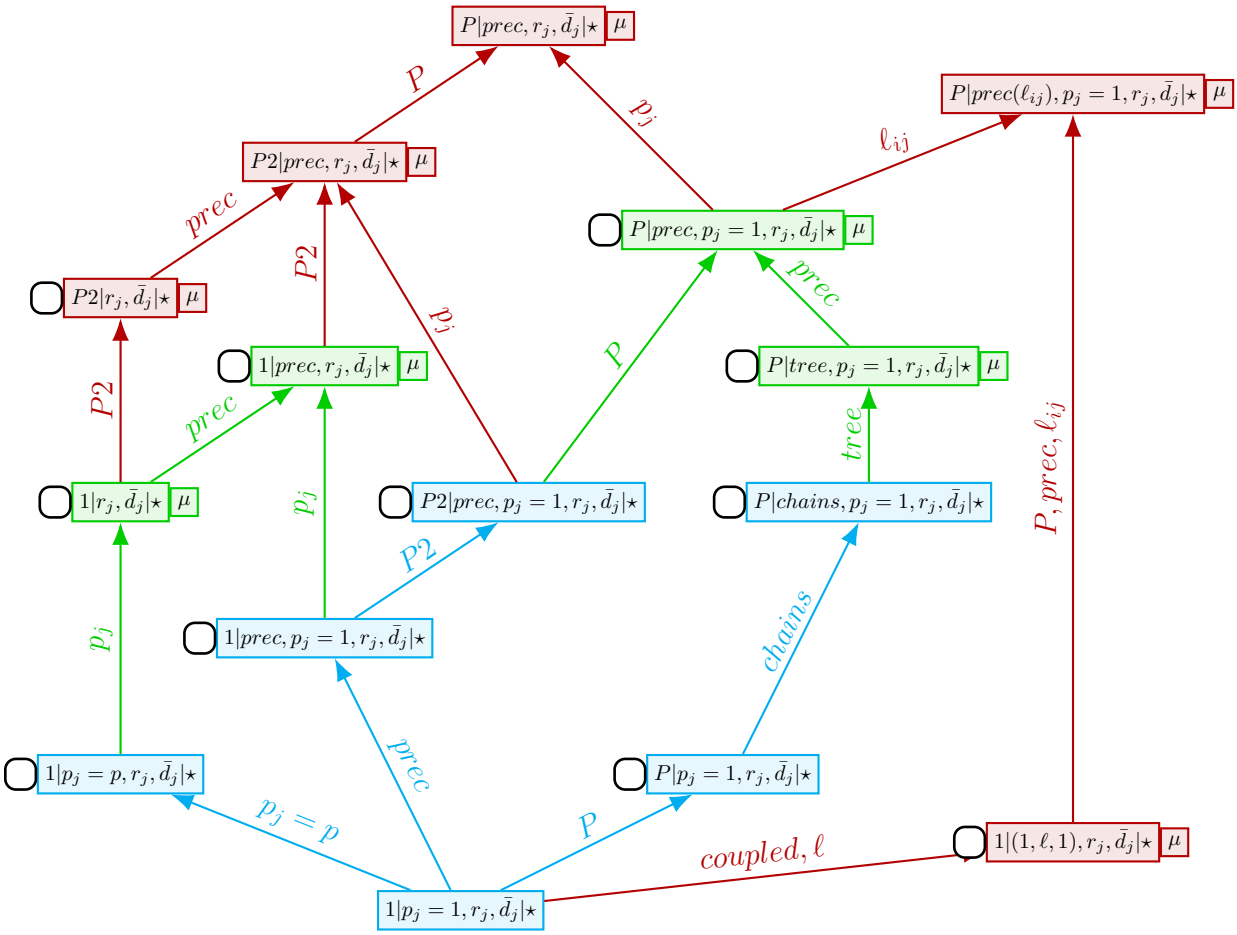


Figure 4.7: Problem map of parameter μ.

4.5.2 Problem Map with Parameter $\mu + p_{max}$

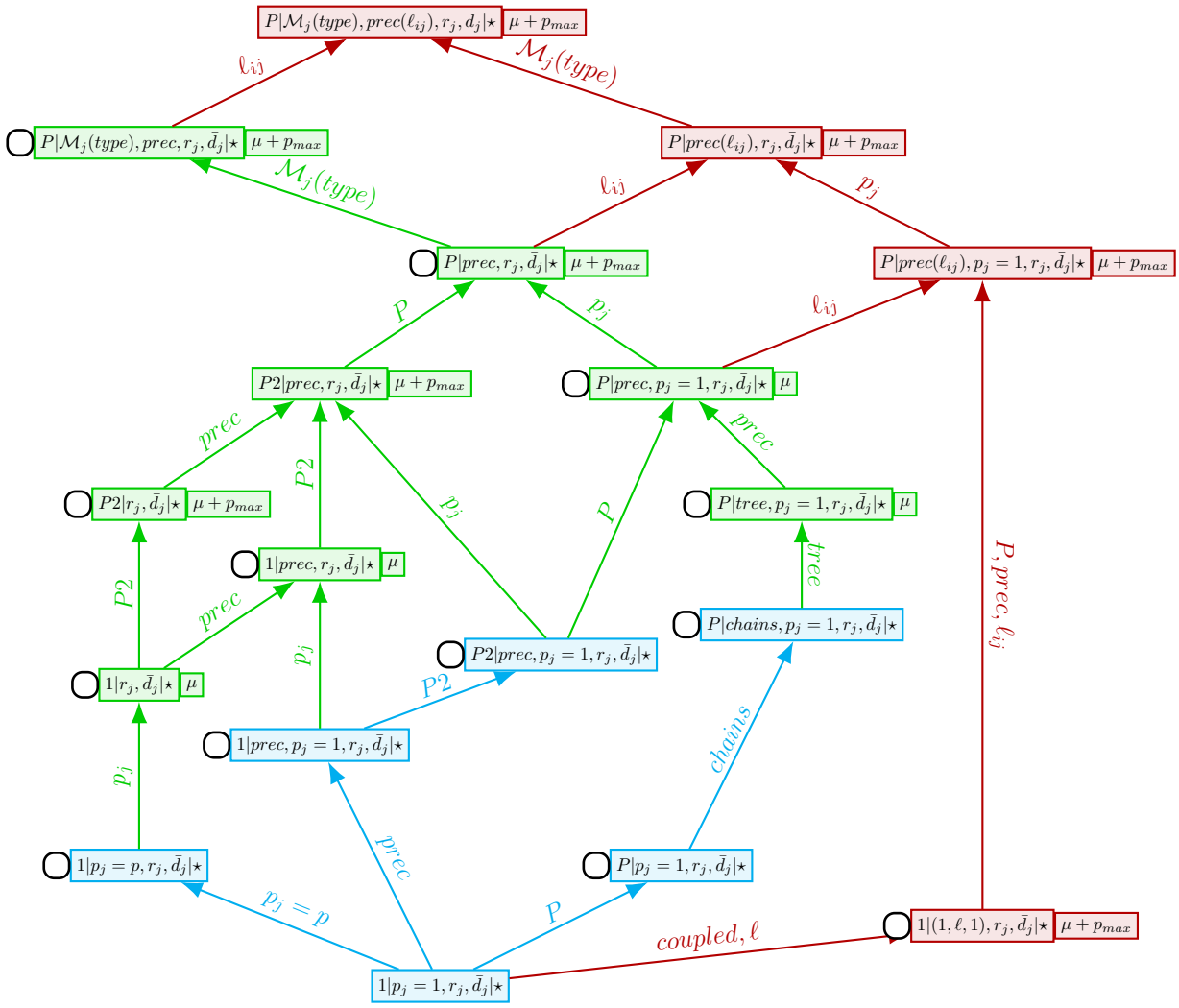


Figure 4.8: Problem map of parameter $\mu + p_{max}$.

4.5.3 Problem Map with Parameter $\mu + \ell_{max}$

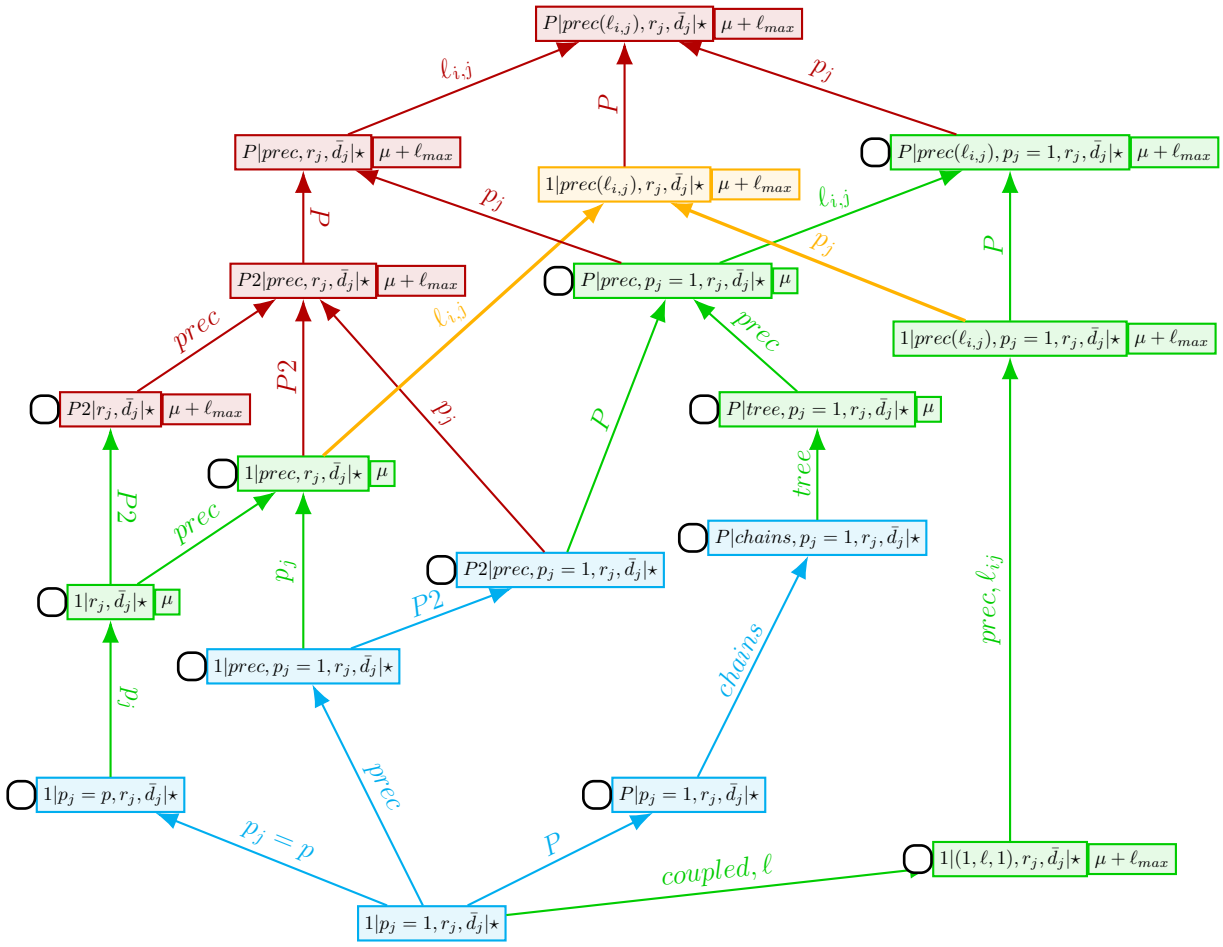


Figure 4.9: Problem map of parameter $\mu + \ell_{max}$.

Chapter 5

Results with Slack σ

5.1 Introduction

Slack σ is the maximum difference between the time window length of a job and their processing time. In other words it is the maximum number of possible starting times for any job (minus one). While this parameter has rarely been considered in the literature, its use in dynamic programming approaches has been successful. For problem $1|r_j, s_{i,j}, reject, \bar{d}_j| \sum_{j \notin R}(w_j T_j - v_j)$ Baart et al. proposed a FPT algorithm in time $\mathcal{O}(\sigma\mu^2 2^\mu \cdot n^2)$ [BdWH21]. In the parallel-machine setting with identical machines Hanen and Munier showed that $P|\mathcal{M}_j(type), prec, r_j, \bar{d}_j|L_{max}$ is FPT parameterized by $\mu + min(p_{max}, \sigma)$ [HMK23].

With a precedence graph and no job time windows van Bevern et al. proposed a variant called the allowed lag λ, which is the maximum difference between a job starting time in the schedule and their earliest possible starting time according to precedence constraints [vBBB+16]. While $P|prec, p_j = 1|C_{max}$ is NP-hard when $\lambda = 1$ (see [LRK78]), they showed that RCPSP is FPT parameterized by $w + \lambda$.

In this chapter we consider the use of parameter σ on single machine problems with job time windows and precedence delays. We first analyze in Section 5.2 the relation between slack σ and pathwidth μ. Then we consider problem $1|prec(\ell_{i,j}), r_j, \bar{d}_j|C_{max}$ with minimum, maximum and/or exact precedence delays. In Section 5.3 we show that for all three delay types the problem is para-NP-hard with respect to slack σ even with chains of unit-time tasks of length two with the same delay value on every precedence relation. Then in Section 5.4 we propose a FPT dynamic programming algorithm with parameters σ and ℓ_{max} combined on single machine scheduling with general delay precedence. Finally in Section 5.5 we summarize the results obtained in this chapter and give our concluding remarks.

5.2 Relations with Pathwidth μ

Existing FPT results suggest that the slack needs to be paired with some width parameter. However the value of pathwidth μ is bounded by slack σ in the following settings:

Result 78. (i) *If a single machine scheduling instance with job time windows is feasible, then* $\mu \leq 2\sigma$.

(ii) *If a parallel machine scheduling instance with identical machines and job time windows is feasible, then* $\mu \leq 2(\sigma+1) \cdot m - 1$.

Proof. (i) Let j be a job. By the definition of slack σ: $r_j \geq d_j - p_j - \sigma$ and $d_j \leq r_j + p_j + \sigma$. Let t be a time unit. If t is part of interval $[r_j, d_j)$ then $r_j \leq t < d_j$. Then by using both inequalities on the definition of slack σ we get: $r_j > t - p_j - \sigma$ and $d_j \leq t + p_j + \sigma$.

This means that whenever job j is scheduled, p_j consecutive time units will be taken by j in time interval $[t - \sigma - p_j + 1, t + \sigma + p_j)$. Thus at least one time unit will be taken by j in time interval $[t - \sigma, t + \sigma + 1)$. Since we only have one machine, this means that at most $2\sigma + 1$ jobs j can have t be a part of their time window $[r_j, d_j)$ in any feasible schedule of this instance. This yields the wanted inequality: $\mu \leq 2\sigma$.

(ii) By using the same argument with m identical parallel machines, given a time unit t at most $2(\sigma + 1) \cdot m$ jobs j can have t be a part of their time window $[r_j, d_j)$ in any feasible schedule of this instance. This yields the wanted inequality: $\mu \leq 2(\sigma + 1) \cdot m - 1$. □

Both inequalities can be checked in time $\mathcal{O}(n \cdot log(n))$ by computing parameters μ and σ. This means that several FPT results from the literature did not need to mention pathwidth μ in their parameter:

Corollary 79.

(i) $1|r_j, s_{i,j}, reject, \bar{d_j}| \sum_{j \notin R}(w_j T_j - v_j)$ *is* FPT *parameterized by* σ.

(ii) $Pm|prec, r_j, \bar{d_j}|L_{max}$ *is* FPT *parameterized by* σ.

(iii) *With minimum delays* $Pm|prec(\ell_{i,j}), p_j = 1, r_j, \bar{d_j}|L_{max}$ *is* FPT *parameterized by* $\sigma + \ell_{max}$.

Proof. Consequences of Result 78 from [BdWH21, HMK23, MHMK22a] respectively. □

Knowing that $P2|r_j, \bar{d_j}|\star$ (and thus $Pm|prec, r_j, \bar{d_j}|L_{max}$) is para-$NP$-hard parameterized by pathwidth μ [HMK23], point (ii) notably shows that σ is a strictly stronger parameter than μ in this setting.

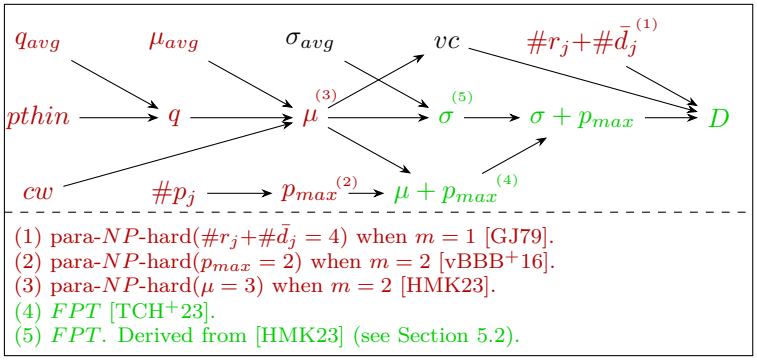


Figure 5.1: Parameterized landscape of $Pm|prec, r_j, \bar{d}_j|\star$.

5.3 Hardness Results

In this section, with slack σ as our parameter, we prove para-NP-hardness of problem $\mathcal{P}_{min}$ (resp. $\mathcal{P}_{max}, \mathcal{P}_{ex}$), which is problem $1|(1, \ell, 1), r_j, \bar{d}_j|\star$ with minimum (resp. maximum, exact) delays.

Theorem 80. *Problems $\mathcal{P}_{min}, \mathcal{P}_{max}$ and $\mathcal{P}_{ex}$ are para-NP-hard parameterized by slack σ.*

The proof of the theorem is developed in the next subsections. In subsection 5.3.1 we show that with minimum or maximum delays, considering coupled tasks is equivalent to allowing chains of length lower than or equal to two. Then we give in subsection 5.3.2 the main ideas behind the reductions. The next subsection gives the detailed reduction and proofs for minimum delays, then the reductions for maximum and exact delays before describing the sketch of the proof of Theorem 80.

5.3.1 Problem Relaxation

We show that adding isolated tasks to coupled task instances does not impact the problem difficulty for minimum and maximum delays.We do so while preserving the slack value. With minimum delays we define $\mathcal{P}'_{min}$ as problem $1|chains(\ell, length \leq 1), p_j = 1, r_j, \bar{d}_j|\star$.

Lemma 81. *$\mathcal{P}'_{min}$ can be reduced to $\mathcal{P}_{min}$ while keeping the same slack.*

Proof. Given an instance I of problem $\mathcal{P}_{min}$, we build an instance I' of problem $\mathcal{P}'_{min}$ by coupling a new task to every isolated task in I. Let D be the maximum deadline in instance I. We put our new tasks in time segment $[D, 2D)$. For each isolated task j in I, we set a coupled task (j, ℓ, j') in I' where $r_{j'} = r_j + D$ and $\bar{d}_{j'} = \bar{d}_j + D$. Then I' has indeed the same slack as I.

We show that I is feasible if and only if I' is feasible. I' is the same as I but with extra tasks, so the indirect implication is straightforward. Now suppose I

has a feasible schedule S. We propose schedule S' which mimics S on the jobs that were in I, and for all isolated tasks j extra task j' is scheduled at time $S(j) + D$. We show that S' is valid. Time segment $[0, D)$ is a replica of valid schedule S, so no problem there. Next in time segment $[D, 2D)$ we only have the extra tasks from the isolated tasks in I. As a result it is a copy of time segment $[0, D)$ but with potentially less tasks. So a problem could only come from one of the added delays. But the offset D between a former isolated task and its extra task is larger than delay ℓ, so there is no issue in time segment $[D, 2D)$ either. Thus S' is a valid schedule of I'. □

With maximum delays we also add extra tasks to be coupled with the isolated tasks. Except we need an extra assumption on the job time windows to complete the reduction. We define $\mathcal{P}'_{max} = 1|chains(\ell, length \leq 1), p_j = 1, r_j, \bar{d}_j|\star$ restricted to instances where for every job j there is an integer k such that time window $[r_j, \bar{d}_j)$ is included in interval $\Theta_k = [k\ell, (k+1)\ell)$. We then say that job j is "in" Θ_k.

Lemma 82. *$\mathcal{P}'_{max}$ can be reduced to $\mathcal{P}_{max}$ while keeping the same slack.*

Proof. Given an instance I of problem $\mathcal{P}'_{max}$, let D be the maximum deadline in instance I. We define $L = max(\ell + 1, D)$. We build an instance I' of problem $\mathcal{P}_{max}$ with time span $L' = 2L$ and where all coupled tasks will have the same delay $\ell' = 2\ell$. For every integer k in $[0, \frac{L}{\ell})$ we associate interval $\Theta_k = [k\ell, (k+1)\ell)$ in I with interval $\Theta'_k = [k\ell', (k+1)\ell')$ as follows.

Let j be a job of I in Θ_k. So, $r_j = k\ell + \rho_j, \bar{d}_j = k\ell + \delta_j$. Instance I' also contains job j, with $r'_j = k\ell' + \rho_j, \bar{d}'_j = k\ell' + \delta_j$, so that its time window is included in $[k\ell', k\ell' + l)$. Moreover, if j is an isolated task, we define a new task j' to make (j, ℓ', j') a coupled task in I'. The time window of j' is the one of j with an offset ℓ: $r'_{j'} = r'_j + \ell, \bar{d}'_{j'} = \bar{d}'_j + \ell$. It is thus included in the right part $[k\ell' + l, (k+1)\ell')$ of interval Θ'_k.

We show that I is feasible if and only if I' is feasible. Suppose I has a feasible schedule S. We propose schedule S' which mimics S on the jobs that were in I. If $S(j) = k\ell + t, 0 \leq t < \ell$ then we set $S'(j) = k\ell' + t$. And for any isolated task j of I, its associated new task j' is scheduled at time $S'(j) + \ell$.

We show that S' is valid. Consider the machine constraint: as in S' the interval $[k\ell', k\ell' + \ell)$ is a copy of interval $[k\ell, (k+1)\ell)$ in S the machine constraint is still satisfied. The new tasks of Θ'_k can only interfere with other new tasks. Their schedule is a copy of S' for jobs of Θ'_k so no resource conflict can occur either. Consider now the precedence constraints. Let (i, ℓ, j) be a coupled task of I with i in Θ_k. Notice that, due to maximum delay ℓ, j is either in Θ_k or in Θ_{k+1}. So either $S'(j) - S'(i) - 1 = S(j) - S(i) - 1 \leq \ell \leq \ell'$ or $S'(j) - S'(i) - 1 = S(j) - S(i) - 1 + \ell \leq 2\ell = \ell'$. If now j is an isolated task of I in I' its successor j' is scheduled at $S'(j) + \ell \leq S'(j) + 1 + \ell'$, so that the precedence constraint is met.

Now suppose I' has a feasible schedule S'. We propose schedule S which mimics S' on the jobs that were in I. If $S'(j) = k\ell' + t(j), 0 \leq t(j) < \ell'$ with j a task from I then we set $S(j) = k\ell + t(j)$. We show that S' is valid. Schedule

S is a copy of schedule S' in interval $[k\ell', k\ell' + \ell)$. So no resource conflict can occur. If (i, l, j) is a coupled task with $i, j \in \Theta_k$ then $S(j) - S(i) - 1 \leq \ell$. If $i \in \Theta_k, j \in \Theta_{k+1}$ then $S'(j) - S'(i) - 1 = (k+1)\ell' + t(j) - k\ell' - t(i) - 1 = \ell' + t(j) - t(i) - 1 \leq \ell'$, so that $S(j) - S(i) - 1 = \ell + t(j) - t(i) - 1 \leq \ell$. Thus S is a valid schedule of I. □

5.3.2 General Framework

In this subsection we describe the scheduling gadgets which will be used, then we explain how we intend to piece them together in our reductions.

Main gadgets

Definition 83. *A switch is a sequence $(a_j)_j$ of tasks with the following property: in any feasible schedule if the first task a_0 is scheduled at its deadline minus one, then all the tasks in the switch must be scheduled at their deadline minus one. And if the last task is scheduled at its release date, then all the tasks in the switch must be scheduled at their release date.*

A switch is called OFF when all its tasks are scheduled at their release date. And it is called ON when all its tasks are scheduled at their deadline minus one.

The purpose of a switch is to propagate some binary piece of information throughout the scheduling instance. A switch can be extended in two ways:

1. **delay propagation:** tasks a_j and a_{j+1} have the same time window length and are coupled with a delay equal to the release date of a_{j+1} minus the release date of a_j minus one. With minimum or exact delays we propagate information from left to right while with maximum delays we propagate from right to left - see Figure 5.2.

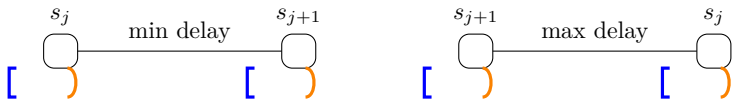


Figure 5.2: Illustration of delay propagation with minimum (resp. maximum) delays.

2. **overlap propagation:** the time windows of tasks a_j and a_{j+1} overlap by exactly one time unit - see Figure 5.3.

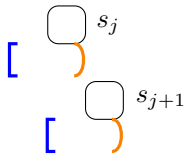


Figure 5.3: Illustration of overlap propagation.

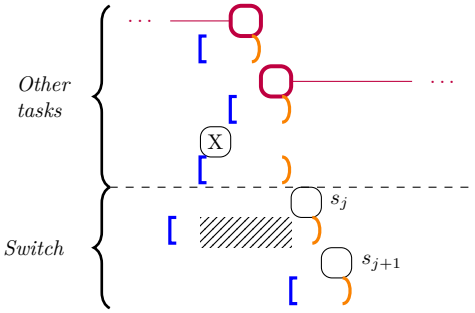


Figure 5.4: Example of overlap propagation with time windows of length more than two.

In the case of overlap propagation when time windows of length longer than two are desired then the intermediate time units can be blocked by fill tasks - marked with a "X" symbol - or tasks from other switches. In particular the situation given in Figure 5.4 will often be encountered in the reductions. With the purple switch and the fill task, there are three tasks to be scheduled in a window of length three. Thus in any valid schedule task a_j can only be scheduled at its release date or its deadline minus one.

Lemma 84. *Any sequence $(a_j)_j$ of tasks defined as a succession of overlap and/or delay propagation is a switch.*

Proof. This is proved by induction on j. Sequence $(a_i)_{0\leq i\leq j}$ has a single task, so it is trivially a switch. Now suppose that sequence $(a_i)_{0\leq i\leq j}$ forms a switch for some $j \geq 0$. Consider the connection between tasks a_j and a_{j+1}. In the case of overlap propagation when task a_j is scheduled at its deadline minus one it blocks the release date of task a_{j+1}. Assuming all but the last time slot are blocked by other tasks, a_{j+1} can only be scheduled its deadline minus one. Conversely if a_{j+1} is scheduled at its release date then a_j can only be scheduled at its release date. So the whole switch $(a_i)_{0\leq i\leq j}$ is OFF and all the corresponding tasks are scheduled at their release date.

In the case of delay propagation suppose a_0 is scheduled at its deadline minus one. Since $(a_i)_{0\leq i\leq j}$ is a switch it means that it is ON, and thus all its tasks are scheduled at their deadline minus one - including a_j. Now a_j and a_{j+1} have the same time window length. So in the case of minimum or exact delays the delay "pushes" task a_{j+1} and only leaves its deadline minus one as a possibility. In the case of maximum delays the delays "pulls" task a_{j+1} and also forces it to be scheduled at its deadline minus one. Conversely suppose a_{j+1} is scheduled at its release date. Then the precedence delay has the reverse effect and forces a_j to be scheduled at its release date. Thus switch $(a_i)_{0\leq i\leq j}$ is OFF and all its tasks are scheduled at their release date.

This proves that $(a_i)_{0\leq i\leq j+1}$ is a switch, which concludes the induction. □

Definition 85. *A redirect is a coupled task (τ, ℓ, τ') with the following property if ℓ is a minimum or exact delay (resp. a maximum delay): in any feasible*

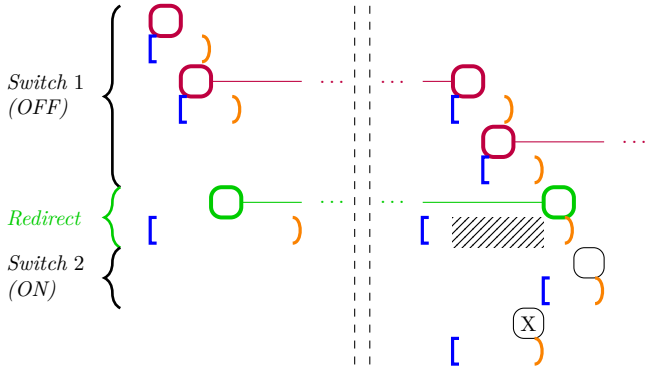


Figure 5.5: Example of a redirect which fits Lemma 86.

*schedule if τ (resp. τ') is **not** scheduled at its release date then τ' (resp. τ) can only be scheduled at its deadline minus one.*

Lemma 86. *Let (τ, ℓ, τ') be a coupled task such that: (a) τ and τ' have the same time window length, (b) $\ell = r_{\tau'} - r_\tau - 1$ and (c) τ or τ' (or both) can only be scheduled at their release date or their deadline minus one. Then this coupled task is a redirect.*

Proof. Consider first the case where ℓ is a minimum or exact delay. Suppose the left half τ can only be scheduled at its release date or its deadline minus one. If τ is not scheduled at its release date in some feasible schedule then it can only be scheduled at its deadline minus one. By points (a) and (b) τ' must also be scheduled at its deadline minus one.

Now suppose the right half τ' can only be scheduled at its release date or its deadline minus one. If τ' is not scheduled at its deadline minus one in some feasible schedule then it can only be scheduled at its release date. By points (a) and (b) τ must also be scheduled at its release date. In case of maximum delay, if τ' is not scheduled at its release date, then either it must be right shifted, which will pull τ to the right, or τ cannot be scheduled at its release date, so is right shifted. □

As the name suggests a redirect will help propagate a binary piece of information the other way around. Typically it will be the interface between two switches. We give an example with minimum delays in Figure 5.5. When Switch 1 is OFF then the left redirect half cannot be scheduled at its release date. So the right redirect half must be scheduled at its deadline minus one, which forces Switch 2 must be ON. And when the Switch 2 is OFF then the right redirect half cannot be scheduled at its deadline minus one. So the left redirect half must be scheduled at its release date, which forces Switch 1 to be ON.

If we want to connect two distant switches then an extra switch can be appended to the redirect in order to cover this distance without increasing the delay value in the redirect. Such a combination will be called a *bridge*.

Definition 87. *A bridge is a combination (an overlap propagation) of a switch and a redirect. It is OFF when all its tasks are scheduled at their release date. And it is ON when all its tasks are scheduled at their deadline minus one.*

In the example given in Figure 5.5 the combination of the redirect and Switch 2 form a bridge.

Reduction baseline

For all three delay types our para-NP-hardness results will be proved with a reduction from the 3-COLORING graph problem. Let $G = (V, E)$ with $V = [0, n)$ and $E = (e_j)_{0 \leq j < m} \subseteq V \times V$. Without loss of generality we suppose that there is no self loop in E.

When building our scheduling instance we segment time into n color choice zones plus m edge check zones. In each color choice zone the color of some node i is chosen in $\{0, 1, 2\}$. And in each edge check zone, for some edge $e_j = \{i_1, i_2\}, i_1 < i_2$ we verify that nodes i_1 and i_2 did not choose the same color.

In terms of tasks we will have:

- n color choice tasks $\mathcal{C}_i$, $i \in [0, n)$: task $\mathcal{C}_i$ represents the color choice of node i. $\mathcal{C}_i$ has three possible starting times and each one corresponds to a color choice.
- $3n$ color switches $C_{i,k}$, $i \in [0, n), k \in \{0, 1, 2\}$: if color k is chosen for node i, then color switch $C_{i,k}$ will be ON and it will propagate this piece of information in the edge check zones.
- $2n$ color choice bridges $B_{i,k}$, $i \in [0, n), k \in \{0, 2\}$. In the color choice zone associated to each node $i \in [0, n)$: bridge $B_{i,0}$ (resp. $B_{i,2}$) will "connect" color switch $C_{i,0}$ (resp. $C_{i,2}$) with color choice 0 (resp. 2) for color choice task $\mathcal{C}_i$.
- $3m$ edge check bridges $B_{i,k}$, $i \in [0, m), k \in \{0, 1, 2\}$. In the edge check zone associated to each edge $e_j = \{i_1, i_2\}, j \in [0, m)$: for each $k \in \{0, 1, 2\}$ bridge $B_{n+j,k}$ will "connect" color switches $C_{i_1,k}$ and $C_{i_2,k}$ so that both cannot be ON at the same time.
- some fill tasks - marked with an "X" - to set the bridges.

5.3.3 Reduction with Minimum Delays

We build I_{min} a scheduling instance of $\mathcal{P}'_{min}$ where all coupled tasks have the same minimum delay $\ell = \beta - 2$ with $\beta = 18n$. Then, we prove that it is feasible if and only if G is 3-colorable.

Color choice zones are set within time interval $[0, 3\beta)$ while edge check zones are set within time interval $[3\beta, (3 + 6m + 1)\beta)$. Given j in $[0, m)$ and k in $\{0, 1, 2\}$ we define $b_{j,k} = 3 + 6j + 2k$ in order to describe concisely the part of

the $(j+1)^{th}$ edge check zone where color k is checked for edge e_j. Then this segment can be written as $[b_{j,k} \cdot \beta, (b_{j,k}+2) \cdot \beta)$. The time windows of all tasks in I_{min} are given below:

Definition 88 (I_{min})**.**
For each i in $[0,n)$:

- *Color choice task $\mathcal{C}_i$ with $i \in [0,n)$. It has release date $\beta + 18i + 6$ and a time window of length three.*
- *Color switch $C_{i,k}$ with $i \in [0,n), k \in \{0,1,2\}$. It contains $3+6m-k$ coupled tasks. Given $b \in [k, 3+6m)$ the left half of the $(b-k+1)^{th}$ coupled task has release date $b \cdot \beta + 18i + 6k + 2$ and a time window of length two. The right half has the same time window with an offset $\beta - 1$.*

  At the beginning of $C_{i,0}$ we also add an isolated task with release date $18i+1$ and a time window of length two.
- *Other tasks in the $(i+1)^{th}$ color choice zone - see Figure 5.6*
  - *Bridge $B_{i,0}$ (resp. $B_{i,2}$). The left half of its redirect has release date $18i+1$ (resp. $\beta+18i+7$) and a time window of length five. The right half has the same time window with an offset $\beta-1$. The bridge switch has a single task with release date $\beta + 18i + 4$ (resp. $2\beta + 18i + 10$) and a time window of length three (resp. five).*
  - *Six fill tasks: two with a time window of length three to set the redirects (release date $\beta + 18i + 1$ and $2\beta + 18i + 7$) and four with a time window of length one to set the bridge switches (release date $\beta + 18i + 5$, $2\beta + 18i + 11$, $2\beta + 18i + 12$ and $2\beta + 18i + 13$).*

For each couple (e_j, k) in $E \times \{0,1,2\}$ with $e_j = \{i_1, i_2\}, i_1 < i_2$:

- *Tasks in the $(j+1)^{th}$ edge check zone - see Figure 5.7*

  Bridge $B_{n+j,k}$ and $6(i_2-i_1)+1$ fill tasks to set the bridge defined as follows in each time subsection:

  In subsection (i_1, k):
  - *one task from the bridge switch with release date $b_{j,k} \cdot \beta + 18i_1 + 6k + 3$ and a time window of length four.*
  - *two fill tasks with a time window of length one (release date $b_{j,k} \cdot \beta + 18i_1 + 6k + 4$ and $b_{j,k} \cdot \beta + 18i_1 + 6k + 5$) to set this switch task.*

  In the $3(i_2 - i_1) - 1$ subsections (i', k') between (i_1, k) and (i_2, k):
  - *two bridge switch tasks: the first (resp. second) has release date $b_{j,k} \cdot \beta + 18i' + 6k'$ (resp. $b_{j,k} \cdot \beta + 18i' + 6k' + 4$) and a time window of length five (resp. three).*

– *two fill tasks : the first (resp. second) has a time window of length three (resp. one) and release date* $b_{j,k} \cdot \beta + 18i' + 6k' + 1$ *(resp.* $b_{j,k} \cdot \beta + 18i' + 6k' + 5$*) to set the first (resp. second) bridge switch task.*

In subsection (i_2, k)*:*

– *the bridge redirect: its left half with release date* $b_{j,k} \cdot \beta + 18i_2 + 6k$ *and a time window of length five. Its right half has the same time window with an offset* $\beta - 1$*.*

– *one fill task with a time window of length three to set the redirect (release date* $b_{j,k} \cdot \beta + 18i_2 + 6k + 1$*).*

Remark 89. *Note that all tasks have a time window of length five of less, so we have slack* $\sigma = 4$*.*

We show that all the switches and bridges in I_{min} work as intended.

Lemma 90. *(i) Let* $(i, k) \in [0, n) \times \{0, 1, 2\}$*. Then* $C_{i,k}$ *in* I_{min} *is a switch.*
(ii) Let $(i, k) \in [0, n) \times \{0, 2\}$*. Then* $B_{i,k}$ *in* I_{min} *is a bridge.*
(iii) Let $(j, k) \in [0, m) \times \{0, 1, 2\}$*. Then* $B_{n+j,k}$ *in* I_{min} *is a bridge.*

Proof. (*i*) $C_{i,k}$ is a succession of delay propagation and overlap propagation of tasks with time windows of length two. So by Lemma 84 it is a switch.

(*ii*) The switch in $B_{i,k}$ is a succession of delay propagation and overlap propagation. With help from Figure 5.6 it can be easily checked that for all these tasks their intermediate time units are blocked either by fill tasks with a time window of length one or by a combination of one fill task and two tasks from a color switch within a segment of length three. By Lemma 84 it is indeed a switch.

The coupled task (redirect) in $B_{i,k}$ overlaps the left extremity of the switch and its right half can only be scheduled at its release date or its deadline minus one. Then this coupled task meets all conditions from Lemma 86, which confirms that it is a redirect.

(*iii*) This is similar to point (*ii*) except the coupled task overlaps the right extremity of the switch and it is the left half which can only be scheduled at its release date or its deadline minus one. Figure 5.7 can be used as reference. □

Now we show that our scheduling instance simulates a coloring. First each node must choose a color. As such in I_{min} each of the three possibilities of color choice $\mathcal{C}_i$ forces the corresponding color switch to be ON. See Figure 5.6 to visualize the local connecting process.

Lemma 91. *Let* $i \in [0, n), k \in \{0, 1, 2\}$*. In any feasible schedule* S*, if color choice task* $\mathcal{C}_i$ *is scheduled at:*
(0) $r(\mathcal{C}_i)$ *then color switch* $C_{i,0}$ *must be ON,*
(1) $r(\mathcal{C}_i) + 1$ *then color switch* $C_{i,2}$ *must be ON,*
(2) $r(\mathcal{C}_i) + 2$ *then color switch* $C_{i,1}$ *must be ON.*

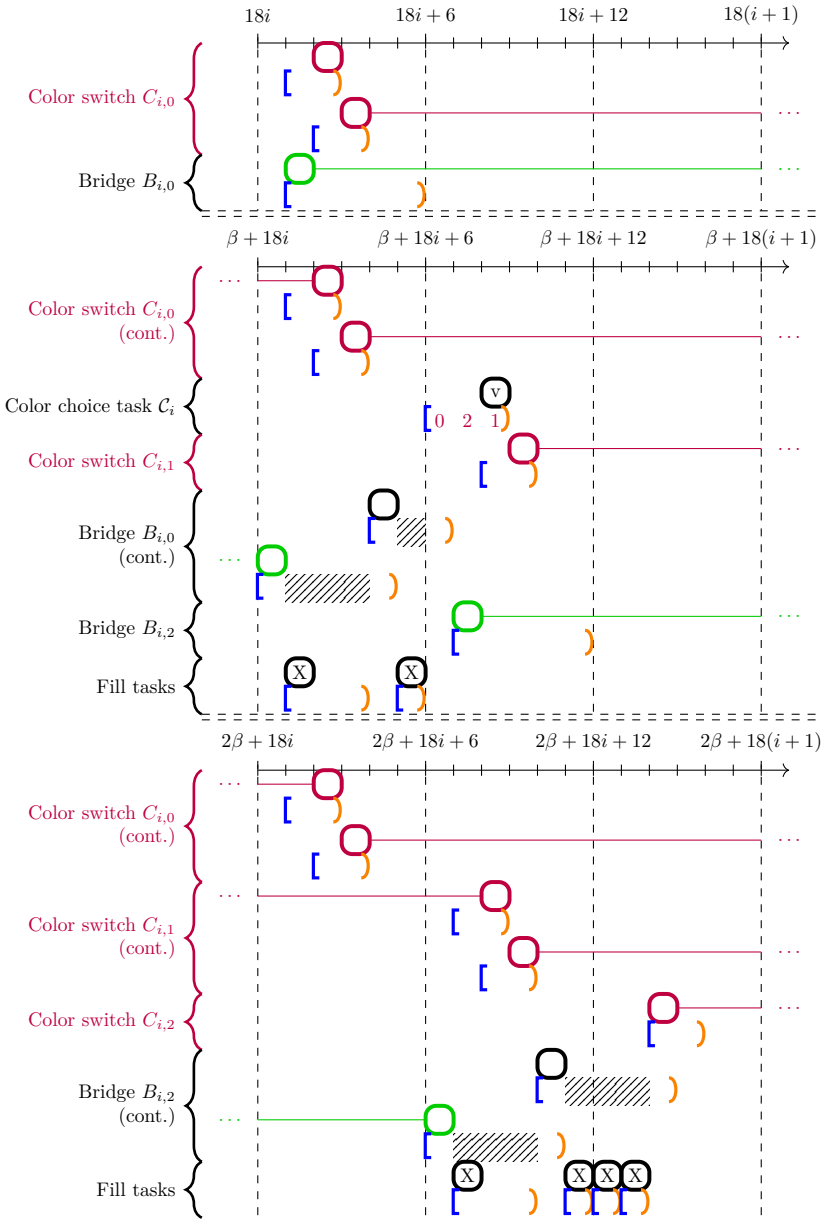


Figure 5.6: The $(i+1)^{th}$ color choice zone in the reduction to problem $\mathcal{P}'_{min}$.

Proof. Case (2) is straightforward. $\mathcal{C}_i$ blocks the release date of the leftmost task in $C_{i,1}$, so the latter must be scheduled at its deadline minus one. Thus by Lemma 90 $C_{i,1}$ must be ON.

In case (1) the left half of the redirect in bridge $B_{i,2}$ must be scheduled later than its release date. Then the right half can only be scheduled at its deadline minus one, which blocks the release date of the only task in this bridge switch. So it can also only be right shifted, which blocks the release date of the leftmost task in color switch $C_{i,2}$. This forces this task to be right shifted, which means that switch $C_{i,2}$ must be ON.

Finally case (0) is proved the same way as case (1) via the only switch task in bridge $B_{i,0}$, the bridge redirect and the leftmost task in color switch $C_{i,0}$. □

Finally given an edge $e_j = \{i_1, i_2\}$ and a color k both nodes i_1, i_2 must not choose color k at the same time. So we must show that color switches $C_{i_1,k}$ and $C_{i_2,k}$ cannot be both ON in a feasible schedule.

Lemma 92. *Let $e_j = \{i_1, i_2\}$ be an edge in E and $k \in \{0, 1, 2\}$ be a color. If any feasible schedule if $C_{i_1,k}$ is ON then $C_{i_2,k}$ is **not** ON. And if $C_{i_2,k}$ is ON then $C_{i_1,k}$ is **not** ON.*

Proof. Consider a feasible schedule. By contradiction suppose that both color switches $C_{i_1,k}$ and $C_{i_2,k}$ are ON. Since $C_{i_1,k}$ is ON some task in $C_{i_1,k}$ blocks the leftmost switch task of bridge $B_{n+j,k}$, so this bridge switch must be ON. This blocks the release date of the left half of the bridge redirect, which forces its right half to be scheduled at its deadline minus one. This corresponds to the deadline minus one of some task in color switch $C_{i_2,k}$. Since the latter is ON we have two tasks scheduled at the same time, which contradicts that the schedule is feasible. □

So in any feasible schedule by Lemma 91 each node must choose a color by having one of its three color switches be ON. Then by Lemma 92 both nodes in an edge cannot choose the same color by both having ON the color switch corresponding to the same color. This is enough to simulate a graph 3-coloring.

Proposition 93. *G is 3-colorable if and only if there exists a feasible schedule for I_{min}.*

Proof.

($\Longleftarrow$) Suppose there exists a feasible schedule s for instance I_{min}. Let s be the starting times of such a schedule. We propose a 3-coloring $(c(i))_{0 \leq i < n}$ based on the starting times of color choice tasks $\mathcal{C}_i$. For i in $[0, n)$, we set

$$c(i) = \begin{cases} 0 & if \quad s(\mathcal{C}_i) = r_{\mathcal{C}_i} \\ 2 & if \quad s(\mathcal{C}_i) = r_{\mathcal{C}_i} + 1 \\ 1 & if \quad s(\mathcal{C}_i) = r_{\mathcal{C}_i} + 2 \end{cases}$$

By contradiction suppose that $(c(i))_{0 \leq i < n}$ is not a valid 3-coloring. Then there is an edge $e_j = \{i_1, i_2\}$ such that $c(i_1) = c(i_2) = k$ with k a color in $\{0, 1, 2\}$.

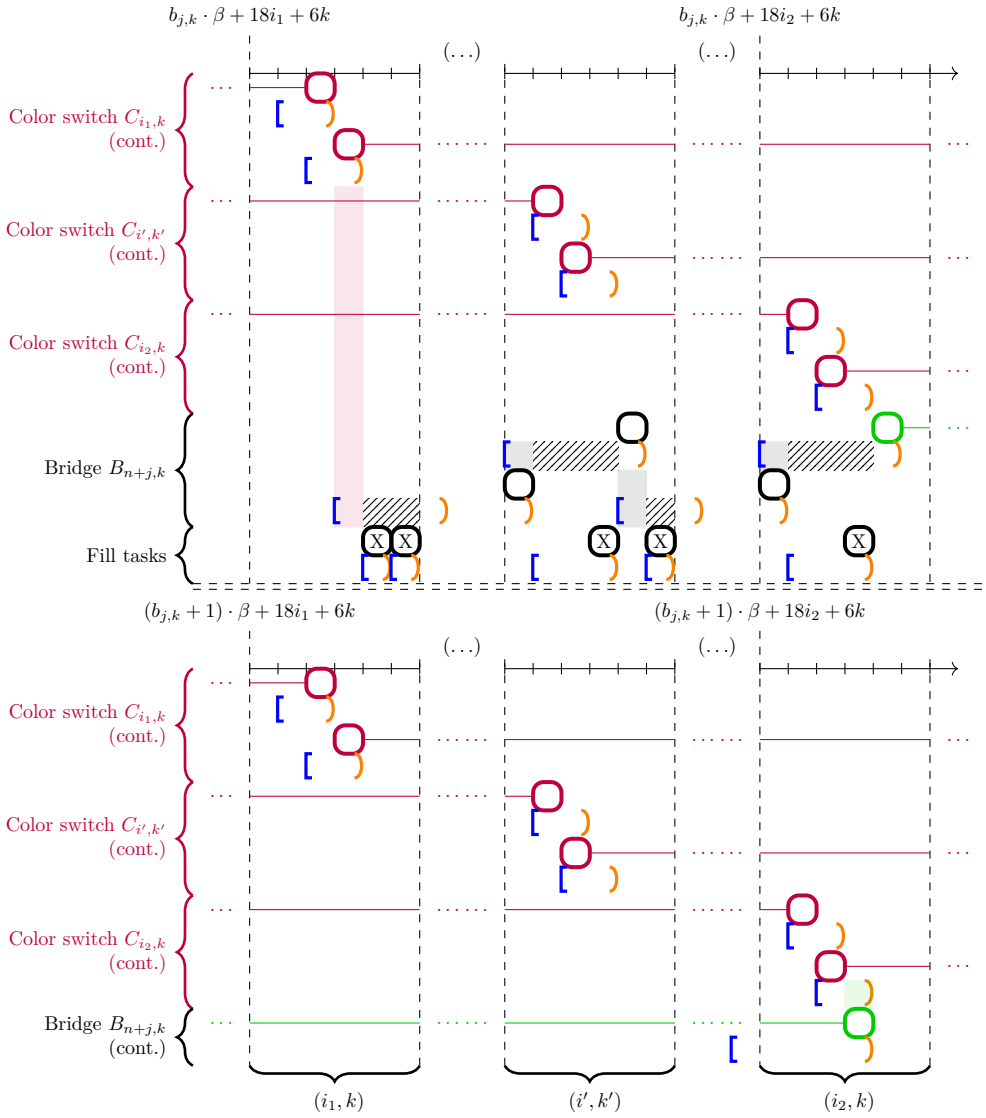


Figure 5.7: The $(j+1)^{th}$ edge check zone corresponding to (edge, checked color) couple (e_j, k) in the reduction to $\mathcal{P}'_{min}$. The middle part describes the tasks in all subsections (i', k') in between subsections (i_1, k) and (i_2, k) in the same section of length β.

Say we have $i_1 < i_2$ without loss of generality. Then by Lemma 91 both color choice switches $C_{i_1,k}$ and $C_{i_2,k}$ are ON in feasible schedule s, which contradicts Lemma 92.

($\implies$) Suppose there exists a valid 3-coloring $(c(i))_{0 \leq i < n}$ for graph G. We propose the following schedule s based on this 3-coloring:

- For $i \in [0, n)$ color choice task $\mathcal{C}_i$ is scheduled at time $r(\mathcal{C}_i)$ if $c(i) = 0$, $r(\mathcal{C}_i) + 1$ if $c(i) = 2$, $r(\mathcal{C}_i) + 2$ if $c(i) = 1$,
- For $i \in [0, n)$ and $k \in \{0, 1, 2\}$ color switch $C_{i,k}$ is ON if $c(i) = k$ and OFF otherwise.
- Fill tasks with a time window of length three are scheduled at their release date if the overlapping color switch is ON and at their deadline minus one if the color switch is OFF.
- For $i \in [0, n)$ bridge $B_{i,0}$ is OFF if $c(i) = 0$ and ON otherwise. Bridge $B_{i,2}$ is ON if $c(i) = 2$ and OFF otherwise.
- For $j \in [0, m), e_j = \{i_1, i_2\}, i_1 < i_2$ and $k \in \{0, 1, 2\}$ bridge $B_{n+j,k}$ is ON if $c(i_1) = k$ and OFF otherwise.

We show that schedule s is feasible. All switches and bridges are either ON or OFF so all delays are met and there is no conflict with any mechanism taken in isolation. Plus the time windows used by the color switches are disjoint from each other so two color switches cannot interfere with each other. The same holds for color choice tasks, bridges and fill tasks. So a conflict could only come from the interaction between two mechanisms of different kinds.

First consider fill tasks. Their time windows do not overlap color choice tasks. Plus these time windows do not overlap with any release date or deadline minus one of any bridge task while all bridges are either ON or OFF. Finally only fill tasks with a time window of length three might interfere with a color switch. When the overlapping color switch is ON the fill task is at its release date, so no conflict. And when it is OFF then the fill task is at its deadline minus one, so no conflict either.

Now consider color choice tasks. Let $i \in [0, n)$. If $c(i) = 1$ then color switch $C_{i,1}$ is ON and bridge $B_{i,2}$ is OFF, so no conflict there. If $c(i) = 2$ then bridge $B_{i,2}$ is ON, so no conflict there either. And if $c(i) = 0$ then bridge $B_{i,0}$ is OFF, so no conflict there either.

Finally consider the interactions between color switches and bridges. Let $i \in [0, n)$. In the $(i + 1)^{th}$ color choice zone bridge $B_{i,0}$ is always ON except when $c(i) = 0$, in which case it is OFF and color switch $C_{i,0}$ is ON. So no conflict here. And bridge $B_{i,2}$ is always OFF except when $c(i) = 2$, in which case it is ON and color switch $C_{i,2}$ is also ON. So no conflict in the color choice zones. Let $j \in [0, n), e_j = \{i_1, i_2\}, i_1 < i_2$. Let $k \in \{0, 1, 2\}$. In the $(j + 1)^{th}$ edge check zone bridge $B_{n+j,k}$ is either ON or OFF, so it never interferes with color switches $C_{i',k'}$ between subsections (i_1, k) and (i_2, k). And it is always OFF except when $c(i_1) = k$, in which case it is ON and color switch $C_{i_1,k}$ is also ON.

So the only conflict would be between bridge $B_{n+j,k}$ and color switch $C_{i_2,k}$, in particular only when they are both ON. However $B_{n+j,k}$ is ON only when $C_{i_1,k}$ is ON too. So we would have $c(i_1) = c(i_2) = k$ which would contradict that $(c(i))_{0\leq i<n}$ is a valid 3-coloring. So there is no conflict involving bridge $B_{n+j,k}$.

Thus s is a feasible schedule for I_{min}. □

5.3.4 Reduction with Maximum Delays

Similar mechanisms are used to build I_{max} a scheduling instance of $\mathcal{P}'_{max}$ where all coupled tasks have the same maximum delay $\ell = \beta$ with $\beta = 18n$. We define instance I_{max}.

Unlike the minimum delay case, color choice information is propagated from right to left. As such edge check zones are set within time interval $[0, (3+6m+1)\beta)$ while color choice zones are set within time interval $[(3+6m+1)\beta, (3+6m+4)\beta)$. Given j in $[0, m)$ and k in $\{0, 1, 2\}$ we define $b_{j,k} = 1 + 6j + 2k$ in order to describe concisely the part of the $(j+1)^{th}$ edge check zone where color k is checked for edge e_j. Then this segment can be written as $[b_{j,k}\cdot\beta, (b_{j,k}+2)\cdot\beta)$. The time windows of all tasks in I_{max} are given below:

Definition 94 (I_{max})**.**

For each i in $[0, n)$:

- *Color choice task $\mathcal{C}_i$ with $i \in [0, n)$. It has deadline $(3+6m+2)\cdot\beta+18i+10$ and a time window of length three.*
- *Color switch $C_{i,k}$ with $i \in [0, n), k \in \{0, 1, 2\}$. It contains $3 + 6m - k$ coupled tasks. Given $b \in [k, 3 + 6m)$ the right half of the $(b - k + 1)^{th}$ coupled task has deadline $(3 + 6m + 3 - b) \cdot \beta + 18i + 6k + 5$ and a time window of length two. The left half has the same time window with an offset $-(\beta + 1)$.*

  At the beginning of $C_{i,0}$ we also add an isolated task with deadline $(3 + 6m + 3 - b) \cdot \beta + 18i + 4$ and a time window of length two.
- *Other tasks in the $(i + 1)^{th}$ color choice zone - see Figure 5.8*
  - *Bridge $B_{i,0}$ (resp. $B_{i,2}$). The right half of its redirect has deadline $(3+6m+3)\cdot\beta+18i+7$ (resp. $(3+6m+2)\cdot\beta+18i+13$) and a time window of length five. The left half has the same time window with an offset $-(\beta+1)$. The bridge switch has a single task with deadline $(3 + 6m + 2) \cdot \beta + 18i + 8$ (resp. $(3 + 6m + 1) \cdot \beta + 18i + 14$) and a time window of length three (resp. five).*
  - *Six fill tasks: two with a time window of length three to set the redirects (deadline $(3+6m+2)\cdot\beta+18i+5$ and $(3+6m+1)\cdot\beta+18i+11$) and four with a time window of length one to set the bridge switches (deadline $(3 + 6m + 2) \cdot \beta + 18i + 7$ and $(3 + 6m + 1) \cdot \beta + 18i + \alpha$ with $\alpha \in \{13, 14, 15\}$).*

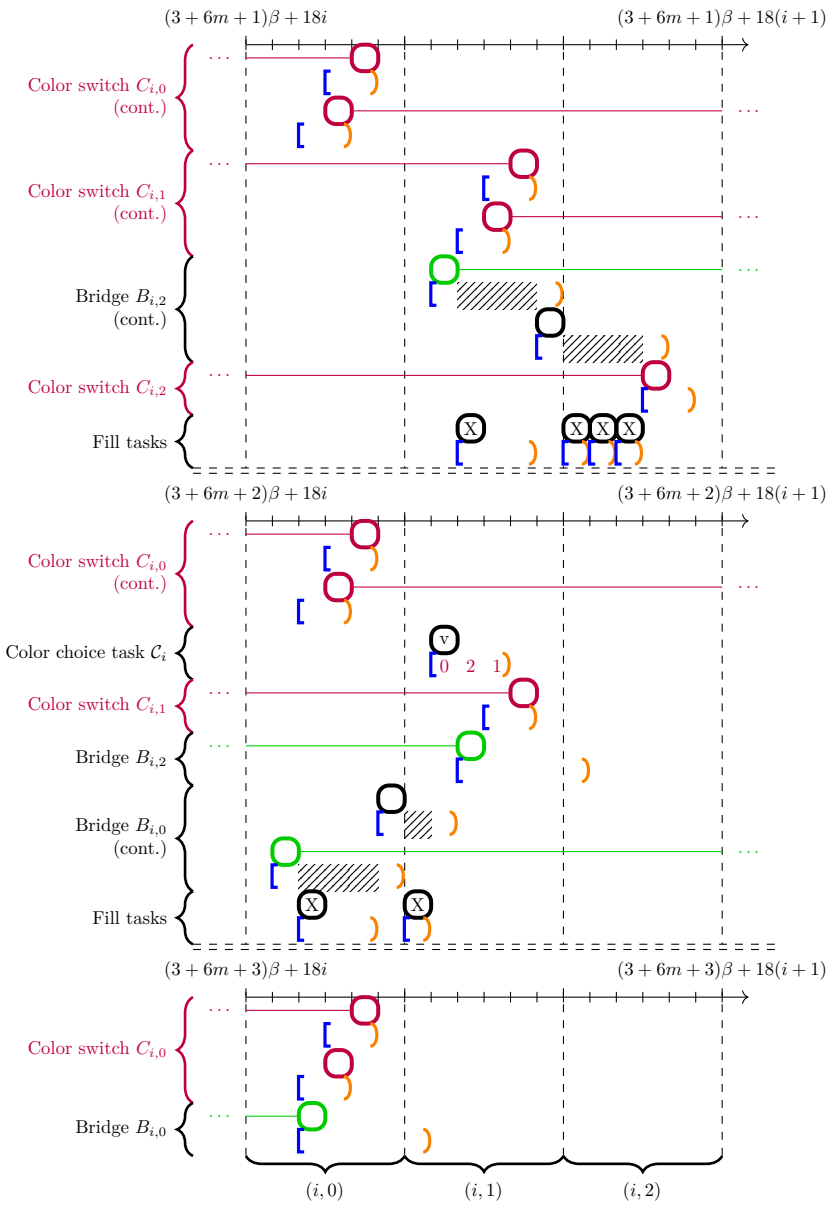


Figure 5.8: The $(i+1)^{th}$ color choice zone in the reduction to problem $\mathcal{P}'_{max}$.

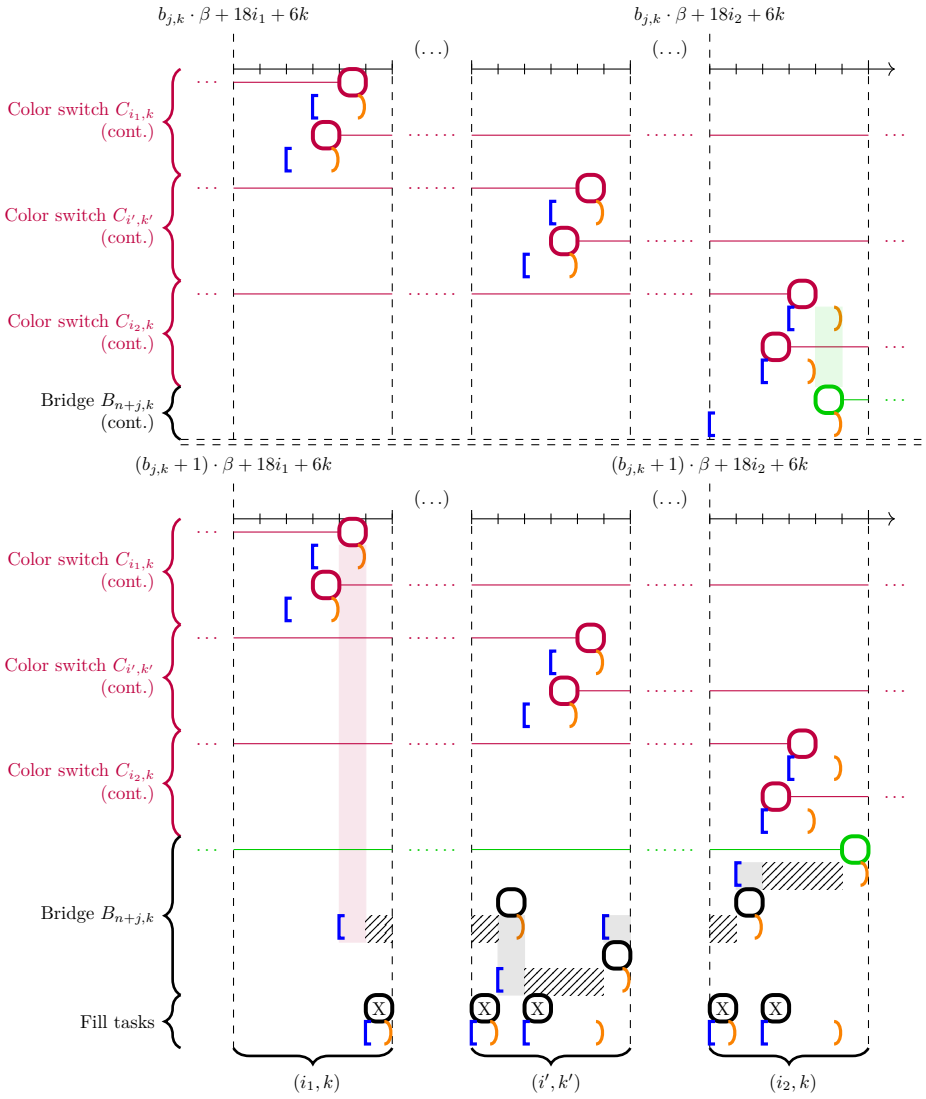


Figure 5.9: The $(j+1)^{th}$ edge check zone corresponding to (edge, checked color) couple (e_j, k) in the reduction to $\mathcal{P}'_{max}$. The middle part describes the tasks in all subsections (i', k') in between subsections (i_1, k) and (i_2, k) in the same section of length β.

For each couple (e_j, k) *in* $E \times \{0,1,2\}$ *with* $e_j = \{i_1, i_2\}, i_1 < i_2$*:*

- *Tasks in the* $(j+1)^{th}$ *edge check zone - see Figure 5.9*

  Bridge $B_{n+j,k}$ *and* $6(i_2 - i_1) + 1$ *fill tasks to set the bridge.*

  In subsection (i_2, k)*:*

  - *the bridge redirect: its right half with deadline* $(b_{j,k}+1)\cdot\beta+18i_2+6k+6$ *and a time window of length five. Its left half has the same time window with an offset* $-(\beta+1)$*.*
  - *one fill task with a time window of length three to set the redirect (deadline* $(b_{j,k}+1)\cdot\beta + 18i_2 + 6k + 5$*).*
  - *one task from the bridge switch with deadline* $(b_{j,k}+1)\cdot\beta+18i_2+6k+2$ *and a time window of length four.*
  - *one fill task with a time window of length one with deadline* $(b_{j,k} + 1)\cdot\beta + 18i_2 + 6k + 1$ *to set this switch task.*

  In the $3(i_2 - i_1) - 1$ *subsections* (i', k') *between* (i_1, k) *and* (i_2, k)*:*

  - *two bridge switch tasks: the first (resp. second) has deadline* $(b_{j,k} + 1)\cdot\beta + 18i' + 6k' + 6$ *(resp.* $(b_{j,k}+1)\cdot\beta + 18i' + 6k' + 2$*) and a time window of length five (resp. three).*
  - *two fill tasks : the first (resp. second) has a time window of length three (resp. one) and deadline* $(b_{j,k} + 1)\cdot\beta + 18i' + 6k' + 5$ *(resp.* $(b_{j,k} + 1)\cdot\beta + 18i' + 6k' + 1$*) to set the first (resp. second) bridge switch task in this subsection.*

  In subsection (i_1, k)*:*

  - *one fill task with a time window of length one with deadline* $(b_{j,k} + 1)\cdot\beta + 18i_1 + 6k + 6$ *to set the last bridge switch task.*

5.3.5 Reduction with Exact Delays

Similarly we build I_{ex} a scheduling instance of $\mathcal{P}_{ex}$ where all coupled tasks have the same minimum delay $\ell = \beta - 2$ with $\beta = 18n$. We define instance I_{ex} then prove that it is feasible if and only if G is 3-colorable.

This time no problem relaxation was proved, so all tasks must be coupled. As such we need a bit more room in each zone to set extra tasks without hindering the base mechanisms. Color choice zones are set within time interval $[0, 4\beta)$ while edge check zones are set within time interval $[4\beta, (4 + 9m + 1)\beta)$. Given j in $[0, m)$ and k in $\{0,1,2\}$ we define $b_{j,k} = 4 + 9j + 2k$ in order to describe concisely the part of the $(j+1)^{th}$ edge check zone where color k is checked for edge e_j. Then this segment can be written as $[b_{j,k}\cdot\beta, (b_{j,k}+3)\cdot\beta)$.

Definition 95 (I_{ex})**.** *All the tasks are coupled tasks with a delay* $\ell = \beta - 2$ *and an offset* $\beta - 1$ *between both halves. We describe the left half of each coupled task below:*

For each i *in* $[0, n)$*:*

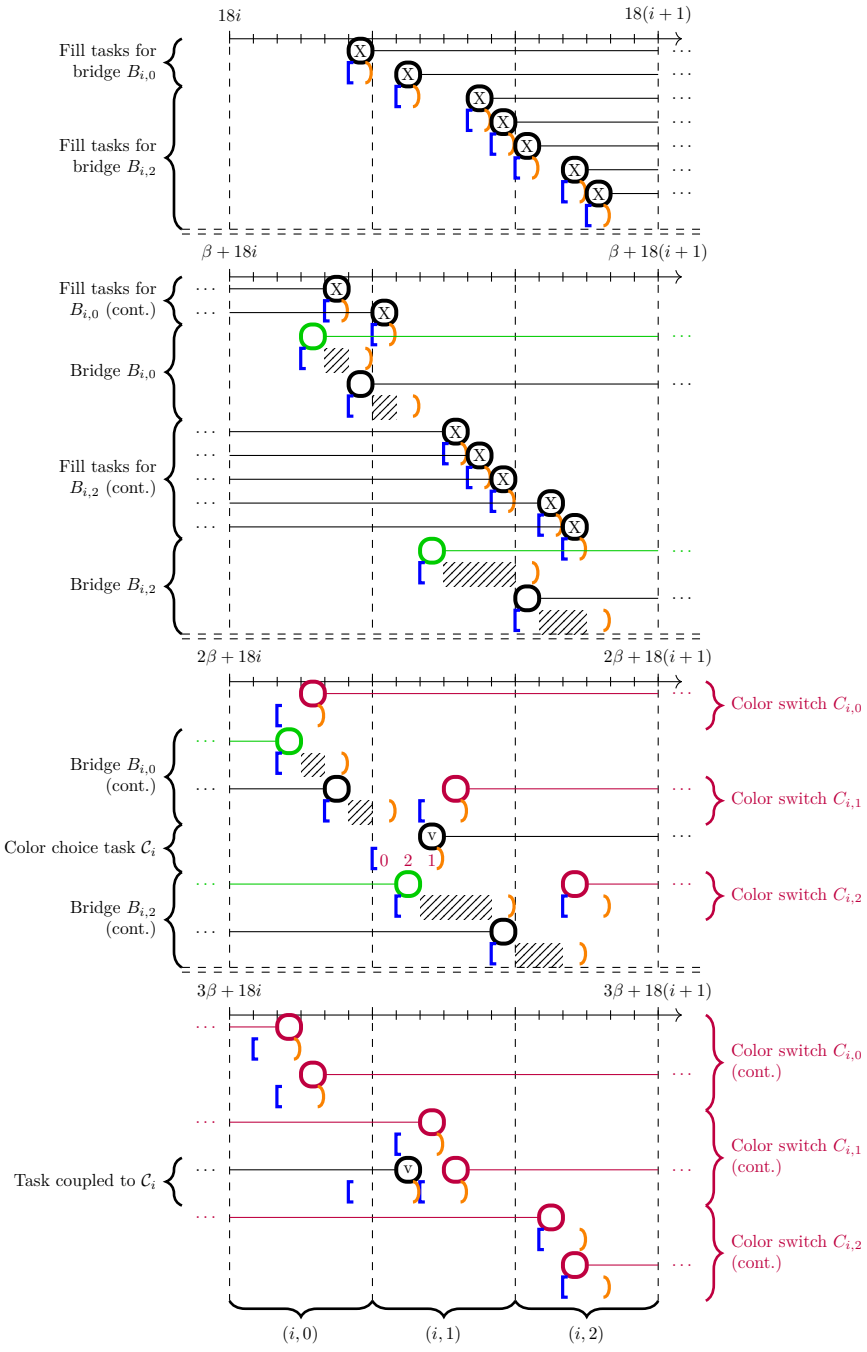


Figure 5.10: The $(i+1)^{th}$ color choice zone in the reduction to problem $\mathcal{P}_{ex}$.

- *Color choice task* $\mathcal{C}_i$ *with* $i \in [0, n)$. *The left half has release date* $2\beta+18i+6$ *and a time window of length three.*
- *Color switch* $C_{i,k}$ *with* $i \in [0, n), k \in \{0, 1, 2\}$. *It contains* $2 + 9m$ *coupled tasks. Given* $b \in [2, 4 + 9m)$ *the left half of the* $(b-1)^{th}$ *coupled task has release date* $b \cdot \beta + 18i + 6k + 2$ *and a time window of length two.*
- *Other tasks in the* $(i+1)^{th}$ *color choice zone - see Figure 5.10*
  - *Bridge* $B_{i,0}$ *(resp.* $B_{i,2}$*). The left half of its redirect has release date* $\beta + 18i + 3$ *(resp.* $\beta + 18i + 8$*) and a time window of length three (resp. five). The bridge switch has a single coupled task with left half release date* $\beta + 18i + 4$ *(resp.* $2\beta + 18i + 10$*) and a time window of length three (resp. four).*
  - *Seven fill coupled tasks with time windows of length one (left half with release date* $\beta + 18i + \alpha$ *with* $\alpha \in \{5, 7, 10, 11, 12, 14, 15\}$.

For each couple (e_j, k) *in* $E \times \{0, 1, 2\}$ *with* $e_j = \{i_1, i_2\}, i_1 < i_2$*:*

- *Other tasks in the* $(j+1)^{th}$ *edge check zone - see Figure 5.11*

  Bridge $B_{n+j,k}$ *and* $3(i_2 - i_1) + 1$ *fill coupled tasks to set the bridge.*

  In subsection (i_1, k)*:*
  - *one coupled task from the bridge switch with left half release date* $(b_{j,k} + 1) \cdot \beta + 18i_1 + 6k + 4$ *and a time window of length three.*
  - *one fill coupled task with a time window of length one (left half release date* $(b_{j,k} + 1) \cdot \beta + 18i_1 + 6k + 5$*) to set this switch task.*

  In the $3(i_2 - i_1) - 1$ *subsections* (i', k') *between* (i_1, k) *and* (i_2, k)*:*
  - *two bridge switch coupled tasks: the first (resp. second) has left half release date* $(b_{j,k}+1)\cdot\beta+18i'+6k'$ *(resp.* $(b_{j,k}+1)\cdot\beta+18i'+6k'+5$*) and a time window of length six (resp. two). Except the subsection right before* (i_2, k) *which only has the first coupled task.*
  - *one fill coupled task with a time window of length four and left half release date* $(b_{j,k} + 1) \cdot \beta + 18i' + 6k' + 1$ *to set the first bridge switch task in this subsection.*

  In subsection (i_2, k)*:*
  - *the bridge redirect: its left half with release date* $(b_{j,k} + 1) \cdot \beta + 18i_2 + 6k - 1$ *and a time window of length six. Its right half has the same time window with an offset* $\beta - 1$.
  - *two fill coupled tasks with a time window of length four (resp. two) with left half release date* $b_{j,k} \cdot \beta + 18i_2 + 6k + 1$ *(resp.* $(b_{j,k} + 1) \cdot \beta + 18i_2 + 6k$*) to set the redirect.*

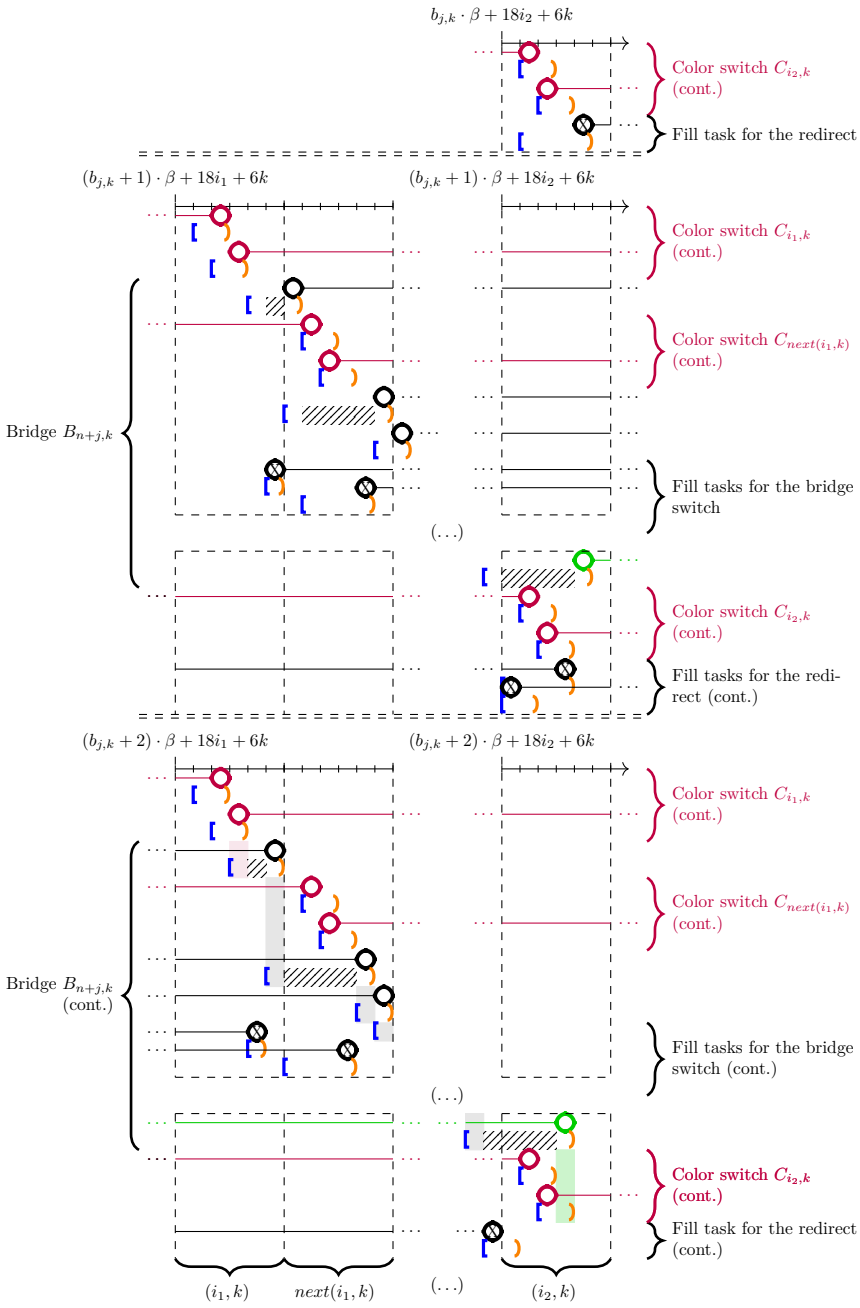


Figure 5.11: The $(j+1)^{th}$ edge check zone corresponding to (edge, checked color) couple (e_j, k) in the reduction to $\mathcal{P}_{ex}$.

Remark 96. *Note that all tasks have a time window of length six of less, so we have slack $\sigma = 5$.*

Compared to the previous reductions the most notable change is the introduction of two new gadgets to set bridges $B_{n+j,k}$ in every edge check zone. First the bridge redirect is set by blocking all intermediate time slots of its left half by setting four tasks to be scheduled within an interval of length four. Second the intermediate time slots of every bridge switch task with time window length six are blocked with a fill coupled task. When the associated color switch is OFF then it is scheduled at its deadline minus one and all four time slots are occupied. When the color switch is ON then it is scheduled at its release date and these four time slots are blocked again. Thus $B_{n+j,k}$ is indeed a bridge and the rest of the proofs unfolds similarly to the minimum delays case.

5.4 Combining with Maximum Delay Value ℓ_{max}

In the previous section problem $1|prec(\ell_{i,j}), r_j, \bar{d}_j|\star$ was shown to be para-NP-hard with parameter σ alone, even with unit jobs and a precedence graph only composed of isolated edges. Plus in [BvdW20] the authors noted that with exact (or maximum) delays and chain precedence this problem is also NP-hard when $\ell_{max} = 0$ - i.e para-NP-hard parameterized by ℓ_{max} - using a straightforward reduction from 3-PARTITION. Here we show that this problem becomes FPT when both parameters are combined.

Theorem 97. *$1|prec(\ell_{i,j}), r_j, \bar{d}_j|\star$ with all three precedence delay types combined is FPT parameterized by $\sigma + \ell_{max}$.*

We consider here an instance I of the decision problem $1|prec(\ell_{i,j}), r_j, \bar{d}_j|\star$ assuming a general precedence graph, precedence delays of all three types in the instance and any processing time of jobs. Stage α of the dynamic programming scheme corresponds to the α^{th} task to be scheduled in order from left to right. In a state u of stage α the following items are stored:

(a) the α^{th} task to be scheduled, denoted by $j(u)$;

(b) its completion time $c(u) = C_{j(u)}$;

(c) the set $A(u)$ of scheduled tasks i for which $\bar{d}_i > c(u)$;

(d) a schedule $S(u)$ of time interval $[c(u) - \ell_{max}, c(u))$, describing all jobs that complete in this interval and their completion times.

Definition 98. *A state u is valid if there exists a feasible schedule of all jobs of $A(u) \cup \{j(u)\} \cup \{i, \bar{d}_i \leq c(u)\}$ that coincides in its last time interval with $S(u)$. We denote by $\mathcal{V}_\alpha$ the set of valid states of stage α and by N_α the subset of jobs $j(u)$ for which there exists a valid state u of stage α.*

We now define the transitions between consecutive stages, which correspond to the edges of a multi-stage graph G_I. Let u be a valid state of stage $\alpha - 1$. To build a valid successor v of stage α we:

- choose $j(v)$ such that $j(v)$ is unscheduled, has no predecessor, or any predecessor k of $j(v)$ is completed before $c(u)$ in u: $k \in A(u)$ or $\bar{d}_k \leq c(u)$;
- choose a completion time $c(v)$ for $j(v)$ within the time interval of $j(v)$ satisfying precedence constraints and resource requirements;
- check that a job k with $\bar{d}_k \leq c(v)$ does not remain unscheduled (otherwise v is discarded);
- then $A(v)$ and $S(v)$ can be deduced from $j(v)$, $A(u)$ and $S(u)$.

Definition 99. *Our* **dynamic programming algorithm** *builds the graph stage by stage, starting from an initial state with an empty schedule. At each stage $\alpha - 1$ we consider each state u and generate only valid successors of u in stage α. Once a successor v is generated by appending a task, the algorithm checks whether the state has already been created, and if not it appends v to the list of states of stage α. The instance is feasible if and only if there is a valid state of stage n.*

Let us now analyze the complexity of this algorithm, by bounding the number of nodes and arcs and the construction of successors of a state. We first assume that a preprocessing is done in $\mathcal{O}(n \cdot log(n))$ that sorts the release times and deadlines by nondecreasing order. Then for each date the set of available tasks is computed. There are $\mathcal{O}(\sigma \cdot n)$ of them with a nonempty set of tasks. All these sets can be computed in time $\mathcal{O}(\sigma \cdot n)$ by going through the previously computed list of ordered release date and deadline values. Finally we also assume that for each deadline t, the set of the at most $\mu + 1$ tasks that crosses time unit $t - 1$ has been preprocessed in time $\mathcal{O}(\mu \cdot n)$.

5.4.1 Number of Tasks Crossing a Time Interval

We first prove the following lemma which bounds the number of tasks that are relevant to a time interval in a feasible instance.

Lemma 100. *Consider a feasible instance of $1|prec(\ell_{i,j}), r_j, \bar{d}_j|\star$. Given a time interval of length T, the number of tasks whose time window intersect with this interval is bounded by $2\mu + T$.*

Proof. Let $[t, t + T)$ be this interval. If $T = 1$ then the result holds by the definition of pathwidth μ. Now suppose $T \geq 2$. Given a task j whose time window $[r_j, \bar{d}_j)$ intersects $[t, t + T)$, at least one time unit in $[t, t + T)$ must be included in $[r_j, \bar{d}_j)$.

- If time t or time $t + T - 1$is included in $[r_j, \bar{d}_j)$:

  by the definition of pathwidth μ the number of such tasks is bounded by $\mu + 1$.

- Otherwise, if a time $t' \in (t, t+T-1)$ is included in $[r_j, \bar{d}_j)$:

  the whole time window of task j must be included in time interval $[t+1, t+T-2]$. By the pigeonhole principle the number of such tasks is bounded by the length of the interval, which is $T-2$, if the instance is feasible.

In total the number of such tasks is bounded by $(\mu+1)+(\mu+1)+(T-2) = 2\mu+T$. □

We now show that in a feasible instance the number of valid couples (α, j) is in $\mathcal{O}((\mu+\sigma)\cdot n)$.

Lemma 101. $\sum_{1\leq\alpha\leq n} |N_\alpha| \leq [2\mu+\sigma+1]\cdot n$.

Proof. We show that each task appears in at most $2\mu+1+\sigma$ stages. We do so by bounding the indices α where a task j can appear.

- Lower bound: let Y_j be the set of tasks with a deadline not greater than r_j. Then in any feasible schedule all these tasks must be scheduled before j. This means that α must be greater than $|Y_j|$.
- Upper bound: let i be a task scheduled before j in some feasible schedule. Then i must be completed before the latest starting time of j, which is bounded by $r_j+\sigma$. Now, if $i \notin Y_j$ then $\bar{d}_i$ must be greater than r_j. This means that the time window of task i intersects with time interval $[r_j, r_j+\sigma)$ of length σ. By Lemma 100 the number of such task i not included in Y_j is bounded by $2\mu+\sigma$. Thus in any feasible schedule task j must appear in a stage α not greater than $|Y_j|+2\mu+\sigma+1$.

Hence a task j can appear in at most $2\mu+\sigma+1$ different stages α. This means that each task is included in at most $2\mu+\sigma+1$ different sets N_α. This bounds the sum of their sizes by the wanted value. □

Then we use Lemma 101 to bound the total number of states:

Proposition 102. *The number of valid states of stage α satisfies:*

$$|\mathcal{V}_\alpha| \leq |N_\alpha| \cdot f(\sigma, \ell_{max})$$

where $f(\sigma,\ell_{max}) = (\sigma+1)\cdot 2^{2\sigma+1}\cdot(4\sigma+\ell_{max})^{\ell_{max}+1}$. The total number of states is bounded by $(5\sigma+1)\cdot f(\sigma,\ell_{max})\cdot n$.

Proof. First we prove the first part of the proposition part by part:

(a) By definition of N_α the number of possible tasks $j(u)$ of a state u of stage α is bounded by $|N_\alpha|$.

(b) For a given $j(u)$, by definition of slack σ there are at most $\sigma+1$ possible completion times for $j(u)$.

(c) If a job i is in $A(u)$, it was scheduled before $j(u)$ but time slot $c(u)$ is part of $[r_i, \bar{d}_i)$.By the definition of pathwidth μ there can only be at most $\mu+1$ of such jobs. This bounds the number of possible sets $A(u)$ by $2^{\mu+1}$.

(d) A partial schedule of length ℓ_{max} is stored. Thus once $c(u)$ is set (from parts (a) and (b)), by Lemma 100 there are at most $2\mu + \ell_{max}$ different tasks which can appear in these partial schedules. For each of these tasks there are at most $\ell_{max}+1$ choices: either not be part of the partial schedule or have its completion time at one of the ℓ_{max} times in this interval. Thus the number of such partial schedules is bounded by $(2\mu + \ell_{max})^{\ell_{max}+1}$.

Now if we add them up with all stages α, we get the same expression, except with $(\sum_{1\leq\alpha\leq n} |N_\alpha|)$ as a factor instead of $|N_\alpha|$. By Lemma 101 this sum is bounded by $(2\mu + \sigma + 1) \cdot n$. Thus the total number of states is bounded by $[2\mu + \sigma + 1] \cdot (\sigma + 1) \cdot 2^{\mu+1} \cdot (2\mu + \ell_{max})^{\ell_{max}+1} \cdot n$. Finally applying Result 78 to the three occurrences of μ in this expression achieves the proof. □

5.4.2 Successor Generation

We show that the outdegree of any node in G_I is bounded and establish the complexity of generating successors of a state.

Lemma 103. *If u is a valid state of stage $\alpha - 1$.*

- *There are at most $\mu+1$ jobs that can be chosen as $j(v)$ for a valid successor v of u. The set of these jobs can be computed in $\mathcal{O}(\mu \cdot log(\mu))$.*
- *There are at most $\sigma + 1$ possible values of $c(v)$ once $j(v)$ is chosen.*
- *Checking whether the choice $(j(v), c(v))$ is consistent with time windows and precedence constraints induced by u and that v is a valid state can be done in $\mathcal{O}((\ell_{max} + \sigma)\log(\sigma))$.*
- *Comparing v to already created states can be done in $\mathcal{O}(log(|\mathcal{V}_\alpha|))$.*

So, the generation of all valid successors of a valid state u of stage $\alpha - 1$ is in $\mathcal{O}\left(log(|\mathcal{V}_\alpha|) \cdot \sigma^2(\ell_{max} + \sigma) \cdot \log(\sigma)\right)$.

Proof. Let us consider the least deadline $\bar{d}_{\hat{j}}$ of an unscheduled job $\hat{j}$. To compute $\bar{d}_{\hat{j}}$ it is sufficient to search within the sorted deadlines of jobs starting from the one next to $c(u)$ (which can be stored with u). We can search sequentially the next deadlines from $c(u)$ and for each such task k decide whether it is unscheduled (checking if $r_k \geq c(u)$ (it is unscheduled) or if it is in $A(u)$). Checking if a job is in $A(u)$ can be done in $\mathcal{O}(log(\mu))$ with appropriate data structure. At most $\mu + 1$ jobs will be checked before finding an unscheduled job if any (since all scheduled jobs have a time window crossing $c(u)$).

Once $\hat{j}$ is found, if another candidate i was chosen instead to extend u, then it would be completed before $\hat{j}$ in a valid schedule. Thus $r_i < \bar{d}_{\hat{j}}$. Plus by the definition of $\hat{j}$ we also have $\bar{d}_{\hat{j}} \leq \bar{d}_i$. This means that time $\bar{d}_{\hat{j}} - 1$ is included

in the time window of all candidates i. By the definition of pathwidth μ this bounds the number of candidates to extend u by $\mu+1$. Our pre-processing then gives the set of candidates.

Assume we have chosen a $j(v)$ among these candidates There are then at most $\sigma+1$ possible completion times for $j(v)$. Let us choose $c(v)$ among these possibilities. Notice that we can scan the possible interval by storing at each step the next and previous deadline of $c(v)$. We now have to check the remaining wanted properties.

1. $j(v)$ is unscheduled. As mentioned previously this costs $\mathcal{O}(\log(\mu))$.
2. $\bar{d}_{j(v)} - p_{j(v)} \geq \max(r_{j(v)}, c(v))$.
3. All jobs i with $\bar{d}_i \leq c(v)$ have been scheduled: we just have to check the deadlines from $c(v)$ by decreasing order to $c(u)$, and check whether all such jobs i are in $A(u)$. At most $\mu+1$ such jobs have been already scheduled since then $c(u)$ is in their time window, so the check will take at most $\mathcal{O}(\mu log(\mu))$.
4. We must check that all predecessors of $j(v)$ have been scheduled. If a predecessor k of $j(v)$ was unscheduled then according to the previous check it would satisfy $\bar{d}_k > c(v)$. We can assume that deadlines and release times are consistent with the precedence constraints with delays. So we would have $r_k \leq r_{j(v)} < c(v) < \bar{d}_k \leq \bar{d}_{j(v)}$. Scanning the deadlines from $c(v)$ to $\bar{d}_{j(v)}$ we can check whether the corresponding jobs are predecessors (with minimal and exact delays) of $j(v)$ and belong to $A(v)$. according to Lemma 100 there are at most $2(\mu+1)+\sigma$ such tasks. The time complexity of this check is $\mathcal{O}((2(\mu+1)+\sigma)log(\mu))$.
5. We must satisfy precedence constraints on $j(v)$. Notice that if k completes before $c(u)-\ell_{max}$, either the constraints cannot be satisfied for exact and maximum delay, or does not impact the starting time of $j(v)$ for minimum delay.

   So we can consider the list of jobs k in $S(u)$, check (in $\mathcal{O}(1)$) whether k is a predecessor of j and compute $t_k = C_k + p(k) + \ell_{k,j(v)}$. If the delay is exact we should check whether $c(v) = t_k$, if the delay is minimum we should have $c(v) \geq t_k$ and if the delay is maximum $c(v) \leq t_k$. In case of maximum or exact delay we check we considered all the predecessors of $j(v)$. All this can be checked in $\mathcal{O}(\ell_{max}+1)$.
6. Finally, checking whether a state v has already been created suppose that all states in $\mathcal{V}_\alpha$ are stored in an appropriate data structure, using an encoding of the state. It should be done in $\mathcal{O}(log(|\mathcal{V}_\alpha|))$.

Then bounding μ with 2σ using Result 78 achieves the proof. □

We now establish that the algorithm is fixed parameter tractable.

Proposition 104. *The proposed dynamic programming algorithm is correct and is FPT with respect to parameters σ and ℓ_{max}. Its time complexity is:*

$$\mathcal{O}^*((\sigma+\ell_{max})^6 \cdot log^2(\sigma+\ell_{max}) \cdot 2^{2\sigma+1} \cdot (4\sigma+\ell_{max})^{\ell_{max}+1}).$$

Proof. Correctness is ensured by the definition of valid states and by Lemma 103. Now considering the time complexity, from Proposition 102 we know that $|\mathcal{V}_\alpha| \leq |N_\alpha| f(\sigma, \ell_{max})$. And the algorithm generates successors of all sates of a stage. We can then, according to Lemma 103, state that the time complexity of the dynamic programming algorithm is in:

$$\mathcal{O}^*\left(\sigma^2 \cdot (\ell_{max}+\sigma) \cdot log(\sigma) \cdot (\sum_{\alpha=1}^{n} |\mathcal{V}_{\alpha-1}| log(|\mathcal{V}_\alpha|))\right).$$

But $log(|\mathcal{V}_\alpha|) \leq \log(|N_\alpha|) + log(f(\sigma, \ell_{max}))$. We deduce that:

$$\sum_{\alpha=1}^{n} |\mathcal{V}_{\alpha-1}| log(|\mathcal{V}_\alpha|) \leq$$
$$f(\sigma, \ell_{max}) \cdot \left((\sum_{\alpha=1}^{n} |N_{\alpha-1}| log(|N_\alpha|)) + log(f(\sigma, \ell_{max})) \cdot (\sum_{\alpha=1}^{n} |N_{\alpha-1}|)\right)$$

Now, $\sum_{\alpha=1}^{n} |N_{\alpha-1}| log(|N_\alpha|) \leq (\sum_{\alpha=1}^{n} |N_\alpha|)^2$ And we know from Lemma 101 that this is bounded by $(5\sigma+1)^2 \cdot n^2$. So we get the overall time complexity $\mathcal{O}(h(\sigma, \ell_{max}) \cdot n^2)$ by setting $h(\sigma, \ell_{max}) =$

$$\left(\sigma^2 \cdot (\ell_{max}+\sigma) \cdot log(\sigma)\right) \cdot f(\sigma, \ell_{max}) \cdot [(5\sigma+1)^2 + (5\sigma+1) \cdot log(f(\sigma, \ell_{max}))].$$

Now $log(f(\sigma, \ell_{max}))$ is in $\mathcal{O}((\sigma+\ell_{max}) \cdot log(\sigma+\ell_{max}))$, so that:

$$h(\sigma, \ell_{max}) \leq \Delta \cdot (\sigma+\ell_{max})^6 \cdot log^2(\sigma+\ell_{max}) \cdot 2^{2\sigma+1} \cdot (4\sigma+\ell_{max})^{\ell_{max}+1}$$

where Δ is a constant. Notice that the bounding done here is quite raw, since the aim is not to have the most accurate complexity but only to prove that the algorithm is FPT.

□

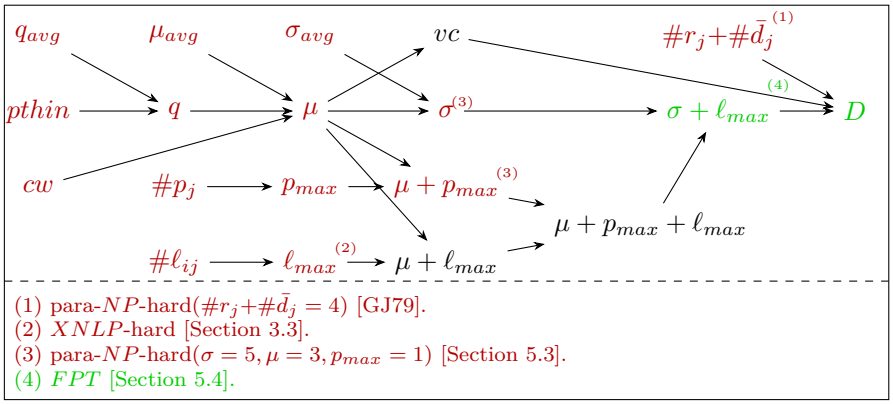


Figure 5.12: Parameterized landscape of $1|prec(\ell_{ij}), r_j, \bar{d}_j|\star$.

5.5 Summary & Concluding Remarks

Parameter	Section	Problem / Setting	Result
σ	5.2	Single machine	$\mu \leq 2\sigma$
		Pm	$\mu \leq 2(\sigma+1) \cdot m - 1$
		-	Several inferred FPT results.
	5.3	$1\|(1,\ell,1), r_j, \bar{d}_j\|\star$	para-NP-hard for all three delay types.
$\sigma + \ell_{max}$	5.4	$1\|prec(\ell_{i,j}), r_j, \bar{d}_j\|\star$	FPT with all three delay types combined.

Figure 5.13: Summary of the results obtained in this chapter.

In this chapter we considered slack σ as a parameter for scheduling problems featuring job time windows. We showed that pathwidth μ can be bounded by slack σ on a single machine or on the parallel machine setting with a fixed number of machines. This sets σ as a stronger parameter in these settings. As a consequence we infer several FPT results, notably on a fixed number of machines with arbitrary values. Despite this, we showed that this parameter cannot handle precedence delays on its own. Indeed for all three delay types we proved that scheduling coupled unit tasks with time windows on a single machine is para-NP-hard with respect to slack σ, even when all coupled tasks shared the same delay value. Fortunately when pairing σ with maximum delay value ℓ_{max} we were able to find a FPT algorithm, even with general precedence and all three delay types combined.

While this covers settings for which no FPT algorithm with respect to pathwidth μ was available, asking to bounding slack σ might significantly limit the range of scheduling instances for which these algorithms are applicable in practice. This suggests completing the landscape with weaker but more specialized parameters, i.e. which would only yield FPT results on a smaller selection of settings and in return perform substantially faster on them.

5.5.1 Problem Map with Parameter σ

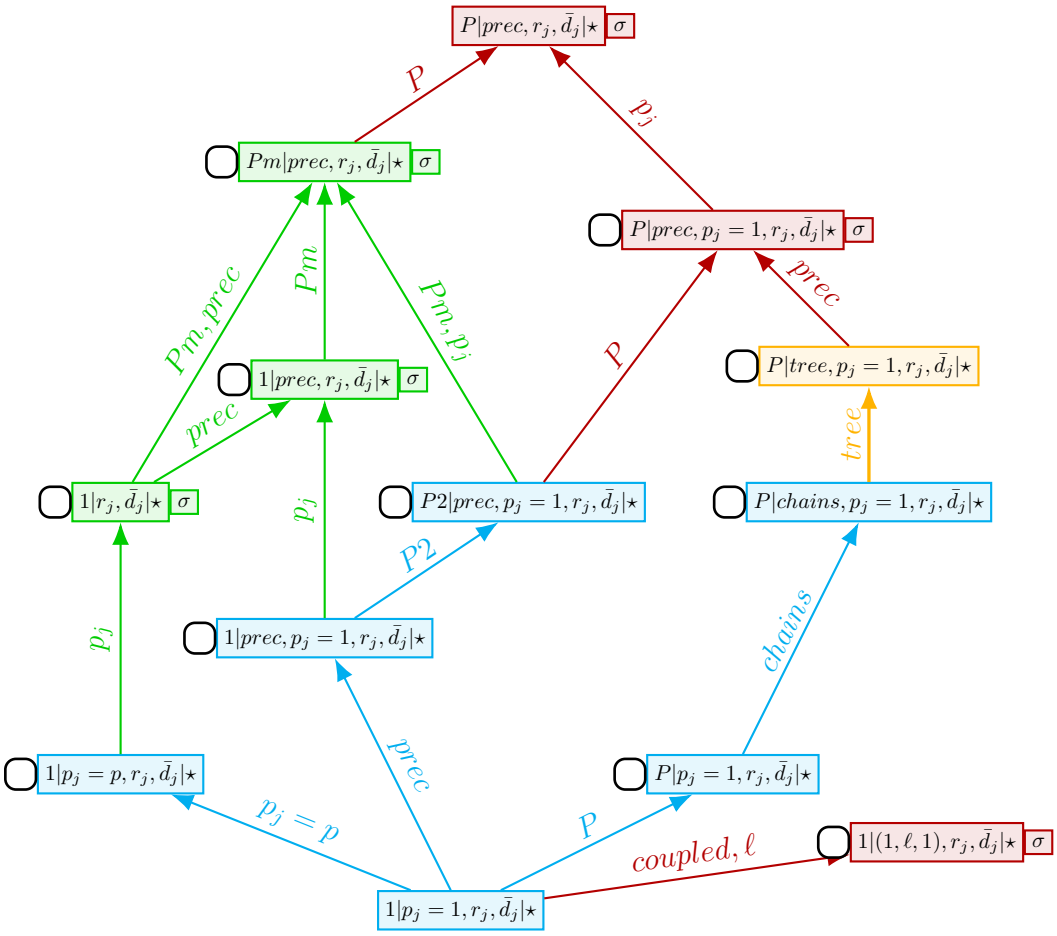


Figure 5.14: Problem map of parameter σ.

5.5.2 Problem Map with Parameter $\sigma + \ell_{max}$

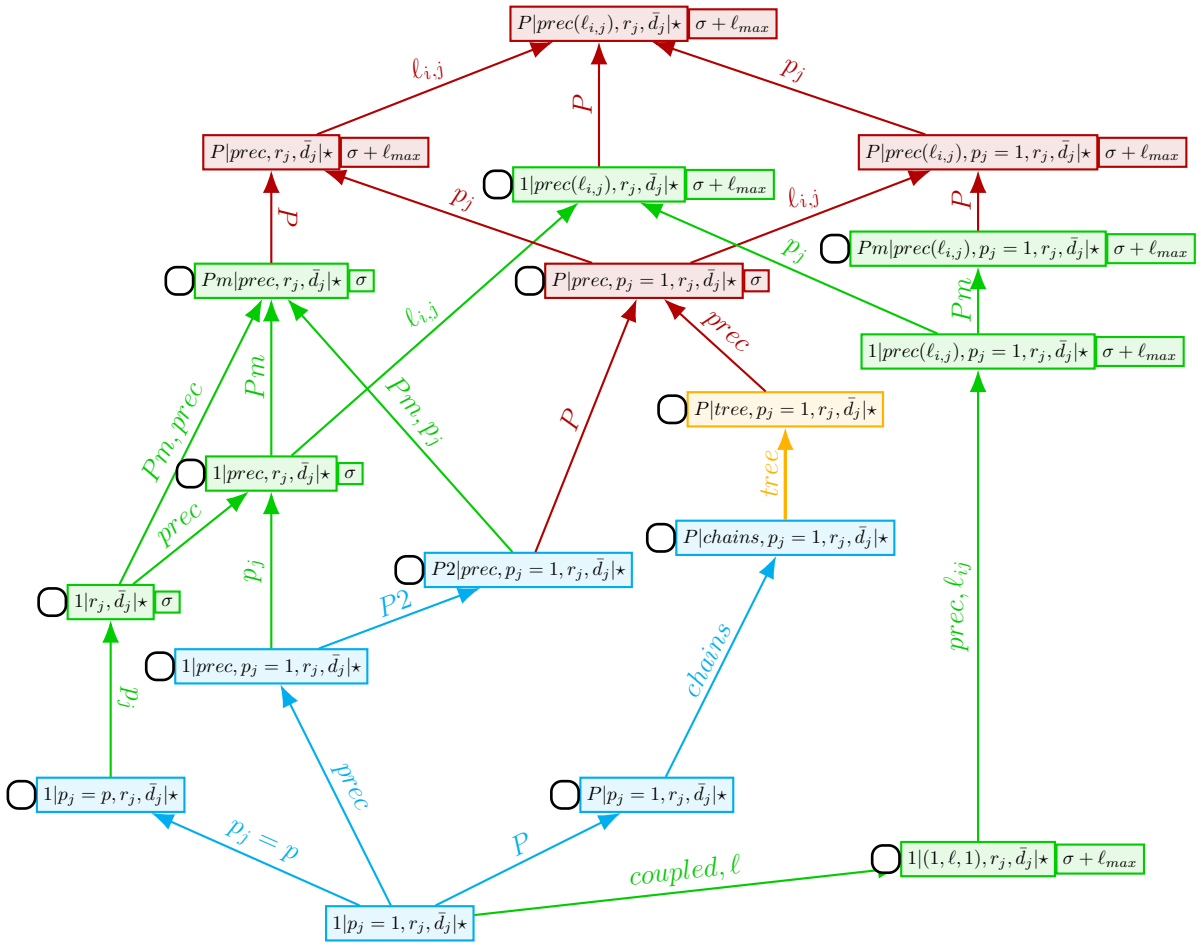


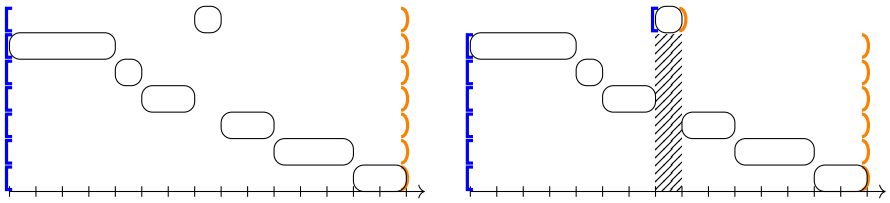
Figure 5.15: Problem map of parameter $\sigma + \ell_{max}$.

Chapter 6

Results with Proper Level q

6.1 Introduction

Despite the recent success of slack σ in single-machine scheduling [BdWH21] and pathwidth μ in parallel-machine scheduling [BdWH21, HMK23, KT21, MHMK22a] such parameters can overestimate the instance difficulty at times. Indeed a single job with large time window length is enough to get an unbounded slack. In the case of pathwidth μ we give an example in Figure 6.1 with two instances of decision problem $1|r_j, \bar{d}_j|\star$. In the left instance all jobs have the same release date and deadline, so this is an easy particular case which can be solved in linear time by adding up the processing times. Yet this instance has the same pathwidth as the right one, which simulates a PARTITION problem by using a fill job right in the middle. Note that both instances yield the same time window overlap graph - the complete graph K_n. So in order to distinguish them one would have to track time window interactions beyond their overlaps.



$\mu = n,\ \ q = 0$ $\qquad$ $\mu = n,\ \ q = n - 1$

Figure 6.1: Two instances of $1|r_j, \bar{d}_j|\star$ with the same pathwidth μ but different q-proper levels q.

Instead of time window overlaps like with pathwidth μ, we propose to track whenever a time window $[r_j, \bar{d}_j)$ strictly includes another time window $[r_i, \bar{d}_i)$ on both ends. This chapter is organized as follows. In Section 6.2 we define proper level q formally and show that it is weaker than previously considered

parameters. We then provide in Section 6.3 a FPT algorithm on single machine scheduling problem with job time windows and precedence relations. In contrast in Section 6.4 we show that the problem becomes para-NP-hard on parallel processors even with unit jobs. Ultimately in Section 6.5 we summarize the results obtained in this chapter and give our concluding remarks.

6.2 Definition & Direct Implications

The base idea is to only consider job time window overlaps which make the job ordering unclear. This typically happens when a time window $[r_j, \bar{d}_j)$ strictly includes another time window $[r_i, \bar{d}_i)$ on both ends. When this is the case we say that job j *surrounds* job i. Such time window interactions were considered in the past [EFM85, EFMR83, GWY01], albeit never in the context of parameterized complexity. We define parameter q as the maximum number of jobs j which can surround a job i.

Definition 105. *(Proper set Π_i) Let $i \in [1, n]$. We say that a job $j \in [1, n]$ surrounds job i when $r_j < r_i$ and $\bar{d}_i < \bar{d}_j$. In other words j surrounds i when time window $[r_i, \bar{d}_i)$ is strictly included by time window $[r_j, \bar{d}_j)$ on both ends. We define $\Pi_i = \{j \in [1, n], j \text{ surrounds } i\}$.*

(Proper level) We define parameter $q = max_{i \in [1,n]} |\Pi_i|$. In other words q is the maximum number of jobs j which can surround a job i.

This definition of q is inspired by Proskurowski and Telle in [PT99] where they defined interval graph classes based on their q-proper level. This is applied to the interval graph given by the job time windows.

Going back to Figure 6.1 the q-proper level effectively distinguishes between both instances: a low value for the easy instance on the left and a high value for the hard instance on the right. As the right instance suggests parameter q can be interpreted as the maximum size of any PARTITION subproblem which is encoded in the scheduling instance.

Finally note that parameter q is weaker than pathwidth μ:

Claim 106. *In any scheduling instance featuring job time windows we have:*

$$q \leq \mu.$$

Proof. Given a scheduling instance with q-proper level q, there is some job i such that q other jobs surround i. This means that time slot r_i is included in $[r_i, \bar{d}_i)$ as well as the time window of all the jobs which surround i. So we have: $q \leq \mu$. □

This implies the following negative results:

Corollary 107. *$P2|r_j, \bar{d}_j|\star$ and $1|(1, \ell, 1), r_j, \bar{d}_j|\star$ are para-NP-hard parameterized by q.*

Proof. The former is derived from [HMK23]. The latter is derived from Section 5.3. □

6.3 A FPT Algorithm on a Single Machine

We show that problem $1|prec, r_j, \bar{d}_j|C_{max}$ is FPT parameterized by q.

Theorem 108. *Any prec-consistent instance of* $1|prec, r_j, \bar{d}_j|C_{max}$ *can be solved in time* $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N + n^2)$ *and space* $\mathcal{O}((q + log(D)) \cdot 2^q \cdot N)$ *where* N *is the number of summits (i.e. jobs which surround no other job).*

Since we have both time windows constraints and precedence relations in our problem, we must ensure that the time constraints are compatible with the partial order $\rightarrow$. We reuse the *prec*-consistency which was introduced in Section 4.2 (see Definition 54). As a reminder we only remove time window parts which are made inaccessible according to precedence relations. So feasibility and the optimal makespan value are unchanged.

Given an instance of $1|prec, r_j, \bar{d}_j|C_{max}$ which is not *prec*-consistent, its release times and deadlines can be adjusted in time $\mathcal{O}(n^2)$ to fulfill this property by using path algorithms. While *prec*-consistency might look inconspicuous it is crucial to the dominance rule given in the next subsection. Without it one can build artificial counterexamples like the one in Figure 6.2.

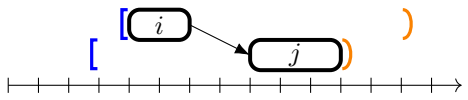


Figure 6.2: A non *prec*-consistent counterexample to the (wED) rule given in Definition 111.

Consider a *prec*-consistent instance. For the remainder of this section we suppose that the jobs have been renamed in $[1, n]$ in lexicographic nondecreasing order of (deadline, release date). So when we write "$i < j$" it means that: $(\bar{d}_i < \bar{d}_j) \vee [(\bar{d}_i = \bar{d}_j) \wedge (r_i \leq r_j)]$. Finally like in [EFM85] we highlight the following subset of jobs:

Definition 109. *We say that a job* $j \in [1, n]$ *is a summit when* j *surrounds no job. In other words: for all jobs* i *in* $[1, n]$ j *is not in* Π_i*. The summits are denoted* $s_1 < \ldots < s_N$ *with* N *the number of summits.*

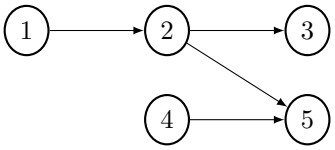
An example of a *prec*-consistent instance on $1|prec, r_j, \bar{d}_j|C_{max}$ is given in Figure 6.3. Jobs 1,2 and 3 surround no other job, so they are the summits of this instance. Summits fulfill the following properties:

Lemma 110. (i) *Every job either is a summit or surrounds a summit.*

(ii) *If* $s_i < s_k < j$ *and* $j \in \Pi_{s_i}$ *then* $j \in \Pi_{s_k}$*.*

Proof. (i) Let j be a job which is not a summit. Then j must surround at least one job j_1. Then either j_1 is a summit or it surrounds some job j_2. After k non-summit steps we would have $r_j < r_{j_1} < \ldots < r_{j_k}$ and $\bar{d}_{j_k} < \cdots < \bar{d}_{j_1} < \bar{d}_j$.

i	1	2	3	4	5
p_i	2	2	3	4	1
r_i	0	3	6	0	5
d_i	6	8	11	13	14
Π_i	$\emptyset$	$\{4\}$	$\{4,5\}$	$\emptyset$	$\emptyset$
$\lvert\Pi_i\rvert$	0	1	2	0	0



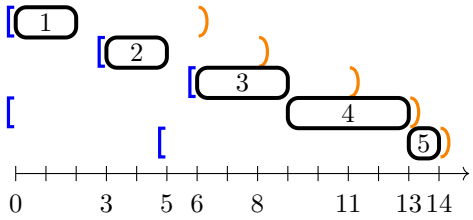


Figure 6.3: An example on $1|prec, r_j, \bar{d}_j|C_{max}$ with five jobs and q-proper level two.

Thus all these jobs are distinct from each other and it eventually ends with some job j_K which is a summit. Then we also have $r_j < r_{j_K}$ and $\bar{d}_{j_K} < \bar{d}_j$, so j surrounds summit j_K.

(ii) Let $s_i < s_k < j$ such that $j \in \Pi_{s_i}$. s_k is a summit so $s_k \notin \Pi_{s_i}$. And since $s_i < s_k$ we necessarily have $r_{s_i} \leq r_{s_k}$. So $r_j < r_{s_i} \leq r_{s_k}$. Now $s_k < j$ implies that $\bar{d}_{s_k} \leq \bar{d}_j$. If $\bar{d}_{s_k} = \bar{d}_j$ then we would have $j < s_k$ which leads to a contradiction. So $\bar{d}_{s_k} < \bar{d}_j$ and thus $j \in \Pi_{s_k}$. □

6.3.1 The Weak Earliest Deadline rule

In this subsection we define the weak earliest deadline (wED) rule and establish the dominance of schedules following this rule. Then we show that such optimal schedules can be further modified to get optimal schedules with a stronger structure based on the summits of the instance.

Motivation

It is worth noting that scheduling rules have already been successfully used in some subproblems of $1|prec, r_j, \bar{d}_j|C_{max}$ [Law73]. One of the most well-known examples is the *earliest deadline rule* [War59] - which will be denoted (ED). Given an instance of problem $1|\bar{d}_j|C_{max}$ the (ED) rule says the following: schedule jobs actively in nondecreasing order of their deadline. This gives the minimum makespan in time $\mathcal{O}(n \cdot log(n))$. The same can be achieved on problem $1|r_j|C_{max}$ with the *earliest release date rule* [Bak84] - which will be denoted (ERD).

However given an instance of $1|prec, r_j, \bar{d}_j|C_{max}$ such strategies become more difficult to pull off. Indeed both (ED) and (ERD) rules can still work but only when release dates and deadlines follow the same order. If there is at least one

job couple (i, j) such that $r_j < r_i$ and $\bar{d}_i < \bar{d}_j$ - i.e. job j *surrounds* job i - then both rules become wrong. Fittingly when this happens we have $q \geq 1$.

In response we propose a variant of the (ED) rule. We call it the *weak earliest deadline rule* and denote it (wED). It says the following: schedule jobs in nondecreasing order of their deadline, except when some job surrounds other jobs. Given each job $i \in [1, n]$ the only allowed exceptions are the jobs j which either surround i or surround a job k with a deadline $\bar{d}_k \leq \bar{d}_i$ and scheduled between j and i. For each job i we show that under the (wED) rule the number of such jobs j is bounded by our parameter q. This will be enough to define a dynamic programming algorithm with a number of states of the form $f(q) \cdot poly(n)$.

Definition and dominance

First we define the (wED) rule formally:

Definition 111. *A feasible schedule τ on an instance I follows the (wED) rule when: $\forall (i, j) \in [1, n]^2$, if $i < j$ then either i is scheduled before j (i.e. $\tau(i) < \tau(j)$) or j surrounds i (i.e. $j \in \Pi_i$) or there exists $h < i$ such that $\tau(j) < \tau(h) < \tau(i)$.*

On problem $1|prec, r_j, \bar{d}_j|C_{max}$ we show that the schedules following the (wED) rule are dominant.

Lemma 112. *Given a feasible schedule τ on a prec-consistent instance I of problem $1|prec, r_j, \bar{d}_j|C_{max}$, one can build a feasible schedule with a makespan lower than or equal to the makespan of τ and which follows the (wED) rule.*

Proof. Let τ be a feasible schedule. If τ follows the (wED) rule then we are done. If not then let (i, j) be a job couple such that $i < j$, $j \notin \Pi_i$ and all jobs between j and i in τ are higher than i. Then in τ we have "jSi" where j and all jobs in sequence S are higher than i. Within the same time interval we propose job reordering "ijS" the following way: push sequence "jS" to the right by p_i time units then insert i right before. Indeed $i < j$ and $j \notin \Pi_i$ so necessarily we have $r_i \leq r_j$. Plus j and all jobs in S are greater than i. Since our instance is *prec*-consistent it means that there is no precedence relation from any of these jobs to i. Thus i can be inserted as desired. Now again j and all jobs in sequence S are greater than i. So their deadlines are non lower than $\bar{d}_i$ and they can be pushed as desired. Note that job couple (i, j) and all couples (i, s) with s in sequence S are (wED)-valid, now that i is scheduled before them. Thus the resulting schedule is feasible, has the same makespan as τ, and has at least one less (wED)-invalidating job couple with (i, j) becoming (wED)-valid and no job couple becoming (wED)-invalidating. This procedure can be repeated a maximum of $\frac{n(n-1)}{2}$ times until we get a feasible schedule which follows the (wED) rule. □

A summit-based decomposition

Upon closer inspection the (wED)-following schedules have the following interesting structural properties, which will serve as the basis of our dynamic programming approach in the next section:

Lemma 113. *Any schedule on a prec-consistent instance I of $1|prec, r_j, \bar{d}_j|C_{max}$ which is feasible and follows the (wED) rule is of the form $T_1 s_1 \dots T_N s_N T_{N+1}$ where:*

- *(i)* $1 = s_1 < \dots < s_N$ *are the summits of the instance,*
- *(ii) for all k in $[1, N]$ and $i < s_k$ we have $\tau(i) < \tau(s_k)$,*
- *(iii) for all k in $[1, N]$ if $j > s_k$ and $\tau(j) < \tau(s_k)$ then $j \in \Pi_{s_k}$,*
- *(iv)*
  - *a. every job in sequence T_1 is in set Π_{s_1},*
  - *b. for all k in $[2, N]$ every job in sequence T_k is in set $\Pi_{s_{k-1}} \cup \Pi_{s_k}$, and*
  - *c. every job in sequence T_{N+1} is in set Π_{s_N}.*

Proof. Let τ be a feasible schedule which follows the (wED) rule. We start by proving point (ii). Let $k \in [1, N]$ and $i < s_k$. We show that $\tau(i) < \tau(s_k)$. By the (wED) rule either $\tau(i) < \tau(s_k)$, or $s_k \in \Pi_i$, or there is some $h < i$ such that $\tau(s_k) < \tau(h) < \tau(i)$. s_k is a summit so by its definition the second case is not possible. Now by contradiction suppose that the third case is true. Take h as the lowest job such that $\tau(s_k) < \tau(h) < \tau(i)$. Then couple (h, s_k) must follow the (wED) rule while $\tau(s_k) < \tau(h)$ and there is no $h' < h$ such that $\tau(s_k) < \tau(h') < \tau(h)$ by minimality of h. Then it means that s_k is in Π_h, which contradicts that s_k is a summit. Thus by the (wED) rule this only leaves us with $\tau(i) < \tau(s_k)$.

Next we prove point (iii). Let $k \in [1, N]$ and $j > s_k$ such that $\tau(j) < \tau(s_k)$. Then by the (wED) rule either $j \in \Pi_{s_k}$ or there is $h < s_k$ such that $\tau(j) < \tau(h) < \tau(s_k)$. In the first case we are done, so suppose we are in the second phase. Take h minimum. Then by the (wED) rule $j \in \Pi_h$. But according to Lemma 110 (i) either h is a summit or it surrounds a summit. In the first case Lemma 110 (ii) would show that $j \in \Pi_{s_k}$ and we are done. In the second case we have a summit $s_i < h$ such that $r_j < r_h < r_{s_i}$ and $\bar{d}_{s_i} < \bar{d}_h < \bar{d}_j$. Then $j \in \Pi_{s_i}$ and again Lemma 110 (ii) can be used to show that $j \in \Pi_{s_k}$.

Now consider two consecutive summits s_k and s_{k+1}. s_k is lower than s_{k+1}. So by point (ii) we know that s_k is scheduled before s_{k+1} in τ. This means that τ is of the form $T_1 s_1 \dots T_N s_N T_{N+1}$ where all the jobs in sequences T_k are non-summits. Finally note job 1 has the lowest deadline and the lowest release date among the jobs with the same deadline. So job 1 is always a summit and $s_1 = 1$. This proves point (i).

Finally we prove point (iv). Let $k \in [1, N+1]$ and j be a job in sequence T_k.

a. If $k = 1$ then $s_1 = 1$, so $j > s_1$ and $\tau(j) < \tau(s_1)$. Thus by point (iii) $j \in \Pi_{s_1}$.

c. If $k = N+1$: job j is not a summit so by Lemma 110 (i) it surrounds one summit s_ℓ. But necessarily $s_\ell \leq s_N$. If $\ell = N$ then we are done. Else we know that $\tau(s_N) < \tau(j)$ so by point (ii) $j > s_N$. Then $s_\ell < s_N < j$ with j surrounding s_ℓ. By Lemma 110 (ii) j also surrounds s_N.

b. If $k \in [2, N]$: again job j is not a summit so by Lemma 110 (i) it surrounds one summit s_ℓ. If $\ell \leq k-1$ then the same reasoning as in point *c.* can be applied to show that $j \in \Pi_{s_{k-1}}$. Now suppose $\ell \geq k$. Then we have $s_k \leq s_\ell < j$ and $\tau(j) < \tau(s_k)$. Thus by point (iii) $j \in \Pi_{s_k}$.

□

Then in any (wED)-following schedule every job lower than a summit s_k must be scheduled before s_k. And among the jobs higher than s_k only those in Π_{s_k} can be scheduled before s_k. With the next section methodology it would already lead to an FPT dynamic programming algorithm with $\mathcal{O}((2q)! \cdot 4^q \cdot N)$ states. However this number can be decreased to $\mathcal{O}(2^q \cdot N)$ by restricting further the set of dominant schedules. We do so by fixing the job order in each T_k. As a result we get the dominance proposed in [EFMR83] for problem $1|r_j, \bar{d}_j|C_{max}$, except we must deal with precedence constraints and restrict ourselves to *prec*-consistent instances. Such schedules will be called *summit-ordered*.

Definition 114. *A schedule on a prec-consistent instance I of $1|prec, r_j, \bar{d}_j|C_{max}$ is called summit-ordered when it is feasible and of the form $T_1 s_1 \dots T_N s_N T_{N+1}$ where properties $(i), (ii), (iii)$ of Lemma 113 are satisfied and moreover the following property holds:*

(iv)
- *a. all jobs in sequence T_1 are in Π_{s_1} and scheduled in nondecreasing order of their release date.*
- *b. for all k in $[1, N-1]$ $T_{k+1} = C_k A_{k+1} B_{k+1}$ where:*
  - *all jobs in sequence C_k are in Π_{s_k}, not in $\Pi_{s_{k+1}}$ and scheduled in nondecreasing order of their deadline,*
  - *all jobs in sequence A_{k+1} are in $\Pi_{s_k} \cap \Pi_{s_{k+1}}$ and scheduled in any order (say in their lexicographic order),*
  - *all jobs in sequence B_{k+1} are in $\Pi_{s_{k+1}}$, not in Π_{s_k} and scheduled in nondecreasing order of their release date.*
- *c. all jobs in sequence T_{N+1} are in Π_{s_N} and scheduled in nondecreasing order of their deadline.*

For example consider the instance given in Figure 6.3. A feasible schedule with job order "12453" (if it exists) would be *summit-ordered* with $C_2 = \emptyset$, $A_3 = \{4\}$ and $B_3 = \{5\}$, while one with job order "12543" would not be. We conclude this section with one final dominance result:

Proposition 115. *On problem $1|prec, r_j, \bar{d}_j|C_{max}$ the summit-ordered schedules are dominant.*

Proof. Let τ be a feasible schedule. From Lemmas 112 and 113 one can build $\bar{\tau}$ of the form $T_1 s_1 \dots T_N s_N T_{N+1}$ with a makespan lower or equal to τ which follows the (wED) rule and verify points (i), (ii) and (iii). In order to prove point (iv) we propose a reordering of each sequence T_k with $k \in [1, N+1]$.

Let $k \in [1, N-1]$. Consider sequence "$s_k T_{k+1} s_{k+1}$". First we want to reorder each sequence T_{k+1} into a form $C_k A_{k+1} B_{k+1}$. Given the jobs in T_{k+1}, C_k groups those in Π_{s_k} and not in $\Pi_{s_{k+1}}$, A_{k+1} groups those common to both proper sets, and B_{k+1} groups those in $\Pi_{s_{k+1}}$ and not in Π_{s_k}. If T_{k+1} is not of this form then we first check the jobs from C_k. If there is a job from C_k scheduled after a job of $A_{k+1} \cup B_{k+1}$, let c be the leftmost of them. Then we have "$s_k C S c$" in schedule $\bar{\tau}$ where the jobs in C are from C_k and the jobs in S are from $A_{k+1} \cup B_{k+1}$. We propose to reorder into "$s_k C c S$" the following way: push sequence S to the right by p_c units then insert c right before S. Indeed all jobs from S are in $\Pi_{s_{k+1}}$ so their deadlines are higher than $\tau(s_{k+1})$ and they could be pushed to the right as desired. Plus c is in Π_{s_k} so its release date is lower than $\tau(s_k)$ and it could be inserted as desired. So the only obstacle would be a precedence relation between some job α in S to job c. On the one hand α is in $\Pi_{s_{k+1}}$ so it is greater than s_{k+1}. On the other hand c is not in $\Pi_{s_{k+1}}$ and is scheduled before s_{k+1} in $\bar{\tau}$. By Lemma 113 (iii) this means that c is lower than s_{k+1} and thus $c < \alpha$. Recall that our instance is prec-consistent, so it means that there cannot be a precedence relation from α to c. Thus the proposed reordering "$s_k C c S$" is allowed and does not increase the makespan of our schedule. Repeating this at most $|C_k|$ times allows us to get all jobs from C_k on the left side of interval $[\tau(s_k) + p_{s_k}, \tau(s_{k+1}))$. Then one can show that all jobs from B_{k+1} can be grouped on the right side of interval $[\tau(s_k) + p_{s_k}, \tau(s_{k+1}))$ with symmetric arguments.

Now let $k \in [1, N]$. We show that the jobs in C_k can be reordered in nondecreasing order of their deadline. Notice that jobs in $C_k \subset \Pi_{s_k}$ have release time lower than r_{s_k}. So they can be scheduled without idle time after s_k. If two consecutive jobs i, j of C_k satisfy $\bar{d}_i > \bar{d}_j$, we cannot have $i \to j$ as our instance is prec-consistent. So i, j can be swapped without modifying the feasibility nor the makespan of the schedule. Iterating such swaps leads to the desired order. Similarly one can show that all jobs from B_k can be reordered in nondecreasing order of their release dates with symmetric arguments. □

6.3.2 The Algorithm

In this subsection we propose a dynamic programming algorithm which explores active *summit-ordered* schedules - "active" meaning that there is no unnecessary waiting time.

Core concept

Assume that we have built a partial *summit-ordered* schedule until s_k. To extend this schedule to other jobs, we need to keep track of the information which ensures that each job is scheduled exactly once. By the definition of

summit-ordered schedules we only need to recall the jobs greater than each summit s_k and scheduled before them. To this purpose we define the following set:

Definition 116. *(Lingering set Λ_k) We define* $\Lambda_k = (\bigcup_{1\leq h\leq k} \Pi_{s_h}) \cap [s_k+1, n]$.

Then a *summit-ordered* schedule $T_1 s_1 \dots T_N s_N T_{N+1}$ can be represented as a state path $(s_1, L_1) \to (\dots) \to (s_N, L_N)$ where every L_k is a subset of Λ_k. Such a path decomposition is motivated by the following lemma:

Lemma 117. *Let $\tau = T_1 s_1 \dots T_N s_N T_{N+1}$ be an active summit-ordered schedule on an instance I of $1|prec, r_j, \bar{d}_j|C_{max}$. Then for every k in $[1, N]$ schedule $T_1 s_1 \dots T_k s_k$ is a feasible active schedule on job subset $[1, s_k] \cup L_k$ where $L_k = (\bigcup_{1\leq h\leq k} T_h) \cap [s_k + 1, n]$.*

Proof. Feasibility and activeness are stable properties by prefix operation. What is left to show is that the set of jobs featured in partial schedule $T_1 s_1 \dots T_k s_k$ is indeed $[1, s_k] \cup L_k$.

($\subseteq$) Since we have a *summit-ordered* schedule, by Definition 114 (i) $s_1, \dots, s_k$ are all lower or equal to s_k, so they are all included in $[1, s_k]$. Now given $\ell \in [1, k]$ and a job $j \in T_\ell$: if $j \leq s_k$ then $j \in [1, s_k]$, else we have $j \in [s_k + 1, n]$ and $j \in \bigcup_{1\leq h\leq k} T_h$, so $j \in L_k$.

($\supseteq$) By definition L_k is included in $\bigcup_{1\leq h\leq k} T_h$. Now let $j \in [1, s_k]$. Since we have a *summit-ordered* schedule, by Definition 114 (ii) all jobs lower than s_k are scheduled before s_k in τ. Thus either $j = s_\ell$ or $j \in T_\ell$ for some ℓ in $[1, k]$. □

Then the corresponding schedule can be retrieved solely from the sets L_k:

Lemma 118. *Let $\tau = T_1 s_1 \dots T_N s_N T_{N+1}$ be an active summit-ordered schedule on an instance I of $1|prec, r_j, \bar{d}_j|C_{max}$. Let $(s_1, L_1) \to (\dots) \to (s_N, L_N)$ be the corresponding state path. Then for every k in $[1, N]$:*

a. $B_1 = L_1$,

b. for every k in $[2, N-1]$:

- $C_k = \{j \in \Pi_{s_k}, j \notin L_k \cup \Pi_{s_{k+1}}\}$,
- $A_{k+1} = \{j \in \Pi_{s_k} \cap \Pi_{s_{k+1}} \cap L_{k+1}, j \notin L_k\}$,
- $B_{k+1} = \{j \in \Pi_{s_{k+1}} \cap L_{k+1}, j \notin \Pi_{s_k} \cup L_k\}$,

c. $C_N = \{j \in \Pi_{s_N}, j \notin L_N\}$.

Proof. This directly comes from Definition 114 (iv) and Lemma 117. □

Finally in order to find the optimal makespan of our instance, we show that we only need to consider *summit-ordered* schedules which are optimal on every prefix $T_1 s_1 \dots T_k s_k$:

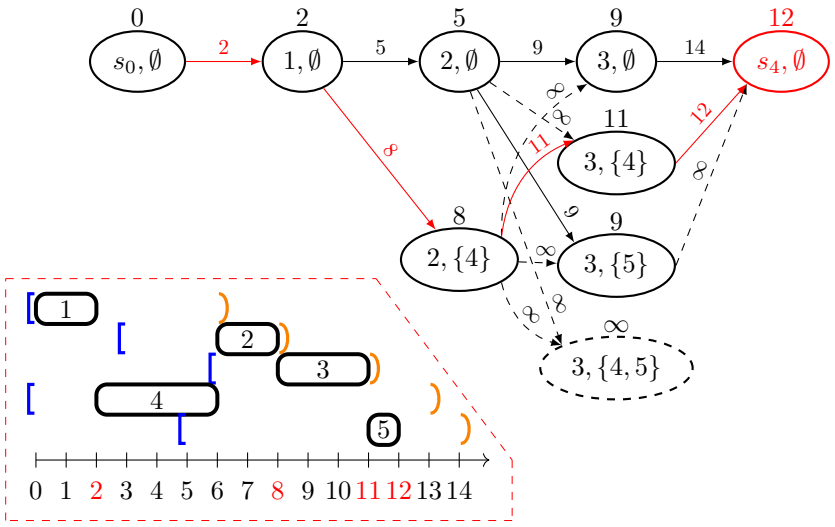


Figure 6.4: State graph associated to the example given in Figure 6.3. The value of each arc is the makespan of the scheduling candidate computed by extend(k, L_k, L_{k+1}). The red path corresponds to active schedule "14235", which gives the optimal makespan.

Lemma 119. *Let $\tau = T_1 s_1 \dots T_N s_N T_{N+1}$ be an active summit-ordered schedule with optimal makespan on an instance I of $1|prec, r_j, \bar{d}_j|C_{max}$. Then one can build an active summit-ordered schedule $\tau' = T'_1 s_1 \dots T'_N s_N T'_{N+1}$ also with optimal makespan and such that for all $k \in [1, N]$ schedule $T'_1 s_1 \dots T'_k s_k$ has optimal makespan among the schedules of state $(s_k, \bigcup_{1 \leq h \leq k} T'_h \cap [s_k + 1, n])$.*

Proof. This is proved by downward induction on k. Suppose that for all $j > k$ schedule $T_1 s_1 \dots T_j s_j$ is optimal among the schedules associated to the same state - which is true in base case $k = N$. Let τ_k be a schedule associated to the same state as schedule $T_1 s_1 \dots T_k s_k$ and with optimal makespan. Then by Proposition 115 one can build $\tau'_k = T'_1 s_1 \dots T'_k s_k$ with the corresponding properties and the same makespan as τ_k. We propose $\tau' = \tau'_k \cdot T_{k+1} s_{k+1} \dots T_N s_N T_{N+1}$, for which the result holds for all $j \geq k$. This concludes the induction. □

This motivates the procedure given in Algorithm 3. Set dummy summits s_0, s_{N+1} with processing time 0, an initial state $(s_0, \emptyset)$ and a final state $(s_{N+1}, \emptyset)$. For $k = 0$ to N use the optimal makespan from states (s_k, L_k) and extend them to the states (s_{k+1}, L_{k+1}). Each of these state pairs is treated by Algorithm 4, which operates accordingly to Lemma 118. Once the unique set of jobs to be scheduled between summits s_k and s_{k+1} is retrieved, the corresponding dominating job ordering is determined. Then these jobs are appended in order and in an active manner (i.e. as soon as possible). Eventually either the schedule computed by Algorithm 4 is invalid, or it is an optimal makespan candidate for the successor state. At the end of the procedure the optimal makespan is given by the value of state $(s_{N+1}, \emptyset)$.

Algorithm 3 main()

▷ Solving $1|prec, r_j\bar{d}_j|C_{max}$.
preprocessing() ▷ Check *prec*-consistency. Compute proper sets and lingering sets.
$r_{s_0}, p_{s_0}, \bar{d}_{s_0}, p_{s_{N+1}} \leftarrow 0$; $r_{s_{N+1}} \leftarrow max_{i\in[1,n]}(r_i)$; $\bar{d}_{s_{N+1}} \leftarrow \infty$
$\Pi_{s_0}, \Lambda_0, \Pi_{s_{N+1}}, \Lambda_{N+1} \leftarrow \emptyset$
Create table T with $N+2$ columns and $2^{|\Lambda_k|}$ slots in each column k.
Initialize T with ∞ everywhere.
$T[0][0] \leftarrow 0$
for $k = 0$ to N **do**
  for each $(L_k, L_{k+1}) \subseteq \Lambda_k \times \Lambda_{k+1}$ **do**
    $T[k+1, L_{k+1}] \leftarrow \min(T[k+1, L_{k+1}], \text{extend}(k, L_k, L_{k+1}))$
return $T[N+1, \emptyset]$

The state graph corresponding to the Figure 6.3 example is given in Figure 6.4. We have $\Lambda_1 = \emptyset, \Lambda_2 = \{4\}$ and $\Lambda_3 = \{4, 5\}$. The optimal makespan is obtained with *summit-ordered* active schedule "14235", which corresponds to state path $(s_0, \emptyset) \to (1, \emptyset) \to (2, \{4\}) \to (3, \{4\}) \to (s_4, \emptyset)$.

State bounding

Now we bound the size of lingering sets Λ_k. It turns out that only the proper set of summit s_k is needed to obtain Λ_k:

Lemma 120. $\forall k \in [1, N]$, $\Lambda_k = \Pi_{s_k}$.

Proof. Let $k \in [\![n]\!]$. By definition every job j in Π_{s_k} is greater than s_k. So j is in Λ_k. Now let $\ell \in \Lambda_k$. We show that $\ell \in \Pi_{s_k}$. By the definition of Λ_k there is $j \in [1, k]$ such that $\ell \in \Pi_{s_j}$. If $j = k$ then we are done. Else we have $s_j < s_k < \ell$ with $\ell \in \Pi_{s_j}$. Then by Lemma 110 (ii) $\ell \in \Pi_{s_k}$. □

This bounds the size of Λ_k and the number of successors (s_{k+1}, L_{k+1}) for each state by parameter q.

Corollary 121. $\forall k \in [1, N]$, $|\Lambda_k| \leq q$. *Plus every state has at most* 2^q *successors.*

Proof. Let $k \in [1, N]$. By the definition of proper level q a proper set has size at most q. By Lemma 120 so does lingering set Λ_k. Now consider some state (s_k, L_k). A successor of this state is of the form (s_{k+1}, L_{k+1}) where L_{k+1} is a subset of Λ_{k+1}. So the bound on the size of Λ_{k+1} bounds the number of successors of state (s_k, L_k) by 2^q. □

Complexity

Now that the number of states and successors have been bounded we compute the time and space complexity of the proposed dynamic programming algorithm. In the preprocessing phase of Algorithm 3 *prec*-consistency is checked then proper sets and lingering sets are computed. This takes time $\mathcal{O}(n^2)$ and space

Algorithm 4 extend(k, L_k, L_{k+1})

▷ Input: $k \in [0, N], L_k \subseteq \Lambda_k, L_{k+1} \subseteq \Lambda_{k+1}$.
▷ Goal: start from an optimal schedule on job subset $[1, s_k] \cup L_k$ and attempt to extend it to job subset $[1, s_{k+1}] \cup L_{k+1}$.
1: $Cmax \leftarrow T[k, L_k]$
2: **if** $[1, s_k] \cup L_k \not\subseteq [1, s_{k+1}] \cup L_{k+1}$ or $Cmax = \infty$ **then return** ∞
▷ Add jobs between summits s_k and s_{k+1} accordingly to Lemma 118.
3: $C \leftarrow list(\{j \in \Pi_{s_k}, j \notin L_k \cup \Pi_{s_{k+1}}\})$ ▷ C_k
4: sort(C, nondecreasing $\bar{d}_j$)
5: $A \leftarrow list(\{j \in \Pi_{s_k} \cap \Pi_{s_{k+1}} \cap L_{k+1}, j \notin L_k\})$ ▷ A_{k+1}
6: $B \leftarrow list(\{j \in \Pi_{s_{k+1}} \cap L_{k+1}, j \notin \Pi_{s_k} \cup L_k\})$ ▷ B_{k+1}
7: sort(B, nondecreasing r_j)
8: $add \leftarrow C \cdot A \cdot B \cdot [s_{k+1}]$
9: **if** prec_invalid(L_k, add) **then return** ∞
10: **for** j in add (in order) **do**
11: $Cmax \leftarrow \max(Cmax, r_j) + p_j$
12: **if** $Cmax > \bar{d}_j$ **then return** ∞
13: **return** $Cmax$

$\mathcal{O}(q \cdot N)$. Now by Corollary 121 the number of states associated to each summit is bounded by 2^q, and for each of these states at most 2^q candidate successors are explored. For each couple $((s_k, L_k), (s_{k+1}, L_{k+1}))$ considered in Algorithm 3 line 8, Algorithm 4 adds at most $2q + 1$ jobs to the schedule represented by state (s_k, L_k). By Lemma 118 these jobs can be retrieved from sets L_k and L_{k+1} (lines 3, 5 and 6) and appended accordingly to Definition 114 (lines 4, 7, 8 and 10 to 12) in time $\mathcal{O}(q \cdot log(q))$.

Now only the precedence checks related to the added jobs in $T_{k+1} \cup \{s_{k+1}\}$ are left (Algorithm 4 line 9). Since we build a *summit-ordered* schedule, following Definition 114 (iv), there is no invalidating precedence constraint from one job in $T_{k+1} \cup \{s_{k+1}\}$ to another. So an invalidating one would necessarily go from a job i in $T_{k+1} \cup \{s_{k+1}\}$ to a job j in $\bigcup_{1 \leq h \leq k} T_h \cup \{s_h\}$. Since our instance is *prec*-consistent j must be greater than i, which is itself greater than or equal to s_{k+1} and thus greater than s_k. So $j \in L_k$. By Corollary 121 this leaves at most $2q + 1$ possible jobs at the start of the potentially invalidating precedence constraint and at most q possible jobs at the end of it. So the precedence checks take time $\mathcal{O}(q^2)$ for any couple $((s_k, L_k), (s_{k+1}, L_{k+1}))$.

Then either some deadline/precedence constraint is invalidated or the jobs are successfully added and a new candidate makespan value is proposed to the successor. So the main loop of Algorithm 3 takes time $\mathcal{O}(q^2 \cdot 4^q \cdot N)$. Now we consider space complexity. For each state (i, L) we must remember index i, some encoding of set L and the minimum makespan currently found. By Corollary 121 this requires space $\mathcal{O}(q + log(D))$ where $D = max_{i \in [1,n]}(\bar{d}_i)$. Finally once the whole state graph has been computed, a feasible schedule with optimal makespan can be retrieved by starting from the final state and going backwards in the state graph. This leads to the time and space complexity given in Theorem 108.

Note that precedence relations only intervene during the precedence checks

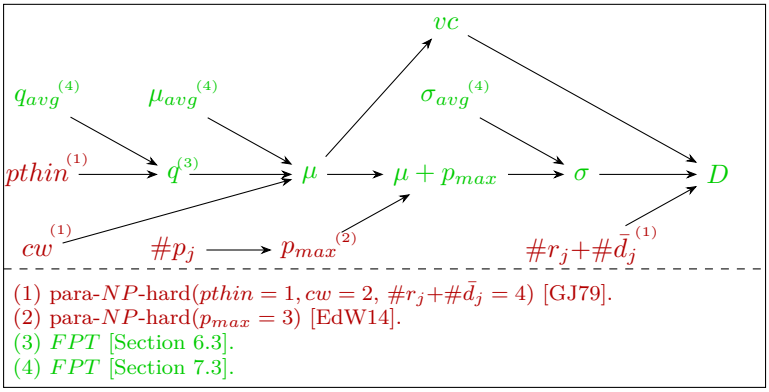


Figure 6.5: Parameterized landscape of $1|prec, r_j, \bar{d}_j|\star$.

between each couple $((s_k, L_k), (s_{k+1}, L_{k+1}))$ in line 9 of Algorithm 4. So if our instance has no precedence relations then each of these couples take time $\mathcal{O}(q \cdot log(q))$ to process instead of time $\mathcal{O}(q^2)$.

Corollary 122. *Any instance of problem $1|r_j, \bar{d}_j|C_{max}$ can be solved in time $\mathcal{O}(max(1, q \cdot log(q) \cdot 4^q) \cdot N + n^2)$ and space $\mathcal{O}((q + log(D)) \cdot 2^q \cdot N)$.*

Finally note that if only the optimal value is wanted and not a schedule, it is possible to reduce the space complexity to $\mathcal{O}((q + log(D)) \cdot 2^q)$. Indeed only the optimal makespan of the states associated to summit s_h must be remembered at any time in order to find the optimal makespan of all states associated to summit s_{h+1}. And when all the states associated to s_h have interacted with their successors, the space they take can be freed and used for the states associated to s_{h+2}. Finally instead of a preprocessing phase proper sets and lingering sets are only computed whenever needed and forgotten once all the states associated to the corresponding job have interacted with all their successors.

Minimizing L_{max}

We conclude this subsection by extending the method to instances of problem $1|prec, r_j|L_{max}$. Deadlines are replaced with due dates and we aim to minimize the maximum lateness of the instance. We run the dynamic programming algorithm for the makespan in order to get an initial value for L_{max}. This could be as far as $D' = [max_{i \in [1,n]}(r_i) + \sum_{i \in [1,n]} p_i]$ from the optimal value. Then we perform a binary search by solving a series of decision problems. We use deadlines instead of due dates and update them accordingly to the current step of the binary search. Each of these steps is solved by one run of our makespan algorithm.

Note that the value of parameter q is the same in every step of the search. Indeed when changing the L_{max} threshold all deadlines are shifted by the same amount of the time, so no relation of the form "$\bar{d}_i < \bar{d}_j$" is ever changed.

This also means that *prec*-consistency, proper sets and lingering sets remain unchanged and thus do not need to be computed again at every step.

Corollary 123. *Any prec-consistent instance of* $1|prec, r_j|L_{max}$ *can be solved in time* $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N \cdot log(D') + n^2)$ *and space* $\mathcal{O}((q + log(D')) \cdot 2^q \cdot N)$.

6.4 Negative Parallel Machine Results

In this section we show that $P|prec, p_j = 1, r_j, \bar{d_j}|\star$ is para-NP-hard parameterized by q in two ways: first when paired with makespan threshold D (which is the maximum deadline here), then when combined with precedence graph width w. Both proofs adapt previous parameterized reductions from Lenstra and Rinnooy-Kan [LRK78] and Bodlaender and Fellows [BF95] respectively.

6.4.1 With Parameter $q + D$

We show that $P|prec, p_j = 1, r_j, \bar{d_j}|\star$ is para-NP-hard parameterized by $q + D$.

Theorem 124. $P|prec, p_j = 1, r_j, \bar{d_j}|\star$ *is* NP*-hard when* $q + D = 3$.

We adapt the reduction from CLIQUE proposed by Lenstra and Rinnooy-Kan in [LRK78]. Let $G = (V, E)$ be a graph with $v = |V|$ and $e = |E|$. Let k be the clique size objective. Then Lenstra and Rinnooy-Kan defined an instance of problem $P|prec, p_j = 1|C_{max} < D$ the following way:

Definition 125. *[LRK78] Set* $a = \frac{k(k-1)}{2}$, $a' = e - a$ *and* $k' = v - k$. *Then scheduling instance* I *has* $m = (1 + max(k, a + k', a'))$ *machines,* $n = 3m$ *jobs and makespan threshold* $D = 3$. *Jobs are split into* v *vertex jobs* $J_i, i \in V$, e *edge jobs* $J_{\{i,j\}}, \{i, j\} \in E$ *and* $3m - v - e$ *fill jobs:* $m - k$ *of them at time* 0, $m-(a+k')$ *of them at time* 1 *and* $m-a'$ *of them at time* 2. *Precedence relations are used to set the desired number of fill jobs in each time unit. Plus for each edge* $\{i, j\} \in E$ *we have precedence from vertex jobs* J_i, J_j *to edge job* $J_{\{i,j\}}$.

Then if there is a clique of size k in G we schedule the corresponding k vertex jobs at time 0, the a edge jobs associated to this clique and the k' remaining vertex jobs at time 1, and the l' remaining edge jobs at time 2. The resulting schedule is feasible. Conversely if there is no clique of size k in G at most $a - 1$ edge jobs can be scheduled before time 2. So at least $a' + 1$ edge jobs have to be scheduled at time 2 which, combined with the $m - a'$ fill jobs set at that time, exceeds the number of machines. Thus there is no feasible schedule for I. This concludes the reduction and the proof that $P|prec, p_j = 1|C_{max} < D$ is strongly NP-hard when $D = 3$.

Note that in any feasible schedule considered in the proof, vertex jobs are scheduled at time 0 or 1 while edge jobs are scheduled at time 1 or 2. So job time windows can be added accordingly without hindering feasibility.

Definition 126. *Instance* I' *of problem* $P|prec, p_j = 1, r_j, \bar{d_j}|\star$ *is defined with help from the instance* I *of Definition 125. We have the set of jobs, we keep*

the precedence relations between vertex jobs and vertex jobs, but we remove the those between fill jobs. Instead we use time windows to set the fill jobs: $m-k$ *of them with time window* $[0,1)$*,* $m-(a+k')$ *of them with time window* $[1,2)$ *and* $m-a'$ *of them with time window* $[2,3)$*. All vertex jobs have time window* $[0,2)$ *and all edge jobs have time window* $[1,3)$*.*

The resulting instance I' is *prec*-consistent and equivalent to base instance I. Note that all time windows have length 1 or 2, so no time window can strictly include another on both ends. Thus in instance I' we have $q=0$. This concludes the reduction and the proof that $P|prec, p_j=1, r_j, \bar{d}_j|\star$ is strongly NP-hard when $q+D=3$.

6.4.2 With Parameter $q+w$

Here we consider parameter $q+w$ and we show that $P|prec, p_j=1, r_j, \bar{d}_j|\star$ is $W[2]$-hard.

Theorem 127. *$P|prec, p_j=1, r_j, \bar{d}_j|\star$ is $W[2]$-hard parameterized by $q+w$.*

We adapt the reduction from k-DOMINATING SET proposed by Bodlaender and Fellows in [BF95]. Let $G=(V,E)$ be a graph with $V=\{u_0,,\ldots,u_{v-1}\}$, $v=|V|$ and $e=|E|$. Let k be the dominating set size objective.

Before introducing the reduction, we first define some values and state their properties that will be used later. We define $c=2v^2+1$. For any $0\leq\alpha\leq(k+1)v$ and $0\leq l_1,l_2\leq v-1$ we define:

$$\lambda(\alpha,l_1,l_2)=(v-1+\alpha\cdot c+l_1\cdot(2v)-l_2) \tag{6.1}$$

Then the following properties can be stated:

Lemma 128. $\forall l_1,l_2,l_1',l_2'\in\{0,\ldots,v-1\}$:

- $\lambda(\alpha+1,l_1,l_2)-\lambda(\alpha,l_1',l_2')\geq v+2$ *for any* $\alpha\in\{0,\ldots,(k+1)v-1\}$,
- $\lambda(\alpha,l_1+1,l_2')-\lambda(\alpha,l_1,l_2)\geq v+1$ *for any* $\alpha\in\{0,\ldots,(k+1)v\}$.

Proof. For a fixed α, the minimum value of $\lambda(\alpha+1,l_1,l_2)$ is obtained with $l_1=0$ and $l_2=v-1$, and the maximum value of $\lambda(\alpha,l_1',l_2')$ is obtained with $l_1'=v-1$ and $l_2'=0$. So we get:

$$\begin{aligned}\lambda(\alpha+1,l_1,l_2)-\lambda(\alpha,l_1',l_2') &\geq [v-1+(\alpha+1)\cdot c-(v-1)]\\ &\quad -[v-1+\alpha\cdot c+(v-1)\cdot(2v)]\\ &= c-(2v^2-v-1)\\ &= v+2\end{aligned}$$

Similarly for fixed α,l_1:

$$\begin{aligned}\lambda(\alpha,l_1+1,l_2)-\lambda(\alpha,l_1,l_2') &\geq [v-1+\alpha\cdot c+(l_1+1)(2v)-(v-1)]\\ &\quad -[v-1+\alpha\cdot c+l_1(2v)]\\ &= 2v-(v-1)\\ &= v+1\end{aligned}$$

□

Then we define an instance I of $P|prec, p_j = 1, r_j, \bar{d}_j|\star$ the following way:

Definition 129. *Instance $I = \langle m, prec, r_j, \bar{d}_j, D\rangle$ has $m = 2k+1$ machines and makespan threshold $D = \lambda((k+1)\cdot v, 1, v-1) = (k+1)\cdot v\cdot c + 2v$. Let $D' = D - v$.*

We have $k+1$ job chains of length D': one floor chain $(f_j)_j$ and k selector chains $(s_{i,j})_{i,j}$ with $1 \leq i \leq k$ and $0 \leq j \leq D'-1$. The floor chain aims at ticking time with not much flexibility. The starting time of the first job in each selector chain will be used to select a node in the dominating set. The jobs constituting these chains are respectively called floor jobs f_j and selector jobs $s_{i,j}$, and they have time window $[j, j+v)$.

On top of these chains we add the following jobs:

- ***floor gadgets:*** *one job f'_j "parallel" to each floor job f_j with $j = \lambda(\alpha, l, 0)$ or $j = \lambda(\alpha, l, 0) + v$ for some $0 \leq l \leq v-1$ and $0 \leq \alpha \leq (k+1)v - 1$. Precedence relations (f_{j-1}, f'_j) and (f'_j, f_{j+1}) are in prec. Plus floor gadget f'_j has the same time window as floor job f_j.*
- ***selector gadgets:*** *for vertex indices $0 \leq l_1, l_2 \leq v-1$ such that $l_1 \neq l_2$ and $(u_{l_1}, u_{l_2}) \notin E$: two jobs $s'_{i,j}, s'_{i,j+v}$ associated to each selector job $s'_{i,j}$ with $1 \leq i \leq k$ and j of the form $\lambda(\alpha, l_1, l_2)$ where $1 \leq \alpha \leq (k+1)v$. Precedence relations $(s_{i,j-1}, s'_{i,j})$, $(s'_{i,j}, s_{i,j+1})$, $(s_{i,j+v-1}, s'_{i,j+v})$ and $(s'_{i,j+v}, s_{i,j+v+1})$ are in prec. Plus selector gadget $s'_{i,j}$ has the same time window $[j, j+v)$ as selector job $s_{i,j}$, while selector gadget $s'_{i,j+v}$ has the time window $[j+v, j+2v)$.*

Note that according to Lemma 128 the time windows of two different floor gadgets never intersect (i.e. $\lambda(\alpha, l+1, 0) > \lambda(\alpha, l, 0) + v$).

Lemma 130. *If graph G has a dominating set of size k then instance I has a feasible schedule.*

Proof. Given $\{u_{\gamma_1}, \ldots, u_{\gamma_k}\}$ a dominating set of graph G, we propose schedule τ where floor jobs/gadgets f_j, f'_j are scheduled at time j and for each $i \in [1, k]$ selector jobs/gadgets $s_{i,j}, s'_{i,j}$ are scheduled at time $\gamma_i + j$. Hence here the nodes in the dominating set define the starting time of the first job of the selector chains and the other jobs are performed in sequence without idle time.

We show that schedule τ is feasible. Clearly all precedence constraints are met in τ. Now we show that no more than $m = 2k+1$ jobs are scheduled at any time T. When T is *not* of the form $\lambda(\alpha, l, 0)$ nor $\lambda(\alpha, l, 0) + v$ where $0 \leq l \leq v-1$ and $0 \leq \alpha \leq (k+1)v - 1$, there is at most one floor job, no floor gadget, at most k selector jobs and at most k selector gadgets scheduled at the same time, due to precedence constraints on the chains and with gadgets, so no issue there.

Now consider any time $T = \lambda(\alpha, l, 0)$, or $T = \lambda(\alpha, l, 0) + v$ of this form. Then the floor gadget f'_T is scheduled at that time. Plus for each $i \in [1, k]$ at most one selector gadget associated to the i^{th} selector chain can be scheduled at time T and, if it exists, it is $s'_{i,T-\gamma_i}$. We show that at least one these k selector gadget possibilities does not exist.

- If u_l is in the dominating set then $l = \gamma_{i_0}$ for some $i_0 \in [1, k]$. If $T = \lambda(\alpha, l, 0)$, then by definition the selector gadget $s'_{i_0,\lambda(\alpha,l,\gamma_{i_0})}$, which can be rewritten as $s'_{i_0,T-\gamma_{i_0}}$, cannot exist (since $l = \gamma_{i_0}$). if $T = \lambda(\alpha, l, 0) + v$, similarly, selector gadget $s'_{i_0,\lambda(\alpha,l,\gamma_{i_0})+v}$ cannot exist.

- Otherwise vertex u_l is adjacent to some vertex u_{γ_i} from the dominating set, with $i \in [1, k]$. If $T = \lambda(\alpha, l, 0)$, then selector gadget $s'_{i,T-\gamma_i} = s'_{i,\lambda(\alpha,l,\gamma_i)}$ where $(u_l, u_{\gamma_i}) \in E$, and thus it cannot exist according to the definition. If $T = \lambda(\alpha, l, 0) + v$, then selector gadget $s'_{i,T-\gamma_i} = s'_{i,\lambda(\alpha,l,\gamma_i)+v}$ would exist although $(u_l, u_{\gamma_i}) \in E$, a contradiction. So no selector gadget from the i^{th} chain is scheduled at time T.

Thus in both cases at most $k - 1$ selector gadgets can be scheduled at the considered time T. So the number of machines is never exceeded anywhere and schedule τ is feasible. □

Conversely given a feasible schedule of instance I we show how to infer a dominating set of graph G. Again we use the position of the selector chains, albeit not from their start. We need to consider some time interval of length c where all chains are scheduled in an active manner. This was ensured by setting a sufficient number of such available disjoint ranges.

Lemma 131. *If instance I has a feasible schedule then graph G has a dominating set of size k.*

Proof. Let τ be a feasible schedule of instance I. Given $\alpha \in [1, (k+1)v - 1]$ the α^{th} range refers to time interval $[(-1+\alpha\cdot c), (-1+(\alpha+1)\cdot c))$, which corresponds to $[\lambda(\alpha, 0, 0) - v, \lambda(\alpha + 1, 0, 0) - v)$. We say that the α^{th} range is *polluted* by a chain when there exists a time unit in this range to which no job from this chain is scheduled. For instance the α^{th} range is polluted by i^{th} selector chain, when there exists a time unit T in this range such that for all $0 \leq j \leq D' - 1$ selector job $s_{i,j}$ is not scheduled at time T.

Since each of the $k + 1$ chains in I have length $D' = D - v$, they can only pollute at most $v - 1$ ranges each, and so at most $(k+1)v - (k+1)$ ranges total. can be polluted. Because the total number of ranges is $(k+1)v - 1$, at least one of them is not polluted, say the α_0^{th} range $[(-1 + \alpha_0 \cdot c), (-1 + (\alpha_0 + 1) \cdot c))$.

Due to precedence relations all floor gadgets f'_j (resp. selector gadgets $s'_{i,j}, s'_{i,j+v}$) executed in range α_0 are executed at the same time as their associated floor job f_j (resp. selector job $s_{i,j}$.

Then by the feasibility of τ and the precedence relations, exactly one job from each chain is scheduled at time $T_0 = (-1+\alpha_0 \cdot c)$. Let $j_0, \ldots, j_k$ be indices such that $f_{T_0-j_0}$ is the floor job scheduled at time T_0 and, given $i \in [1, k]$, s_{i,T_0-j_i} is the selector job of the i^{th} job which is scheduled at time T_0. Then we set $\gamma_i = (j_i - j_0)$ if $j_i \geq j_0$ and $(v + j_i - j_0)$ otherwise. Since j_i and j_0 can differ by at most $v - 1$, we have $\gamma_i \in [0, v - 1]$.

We claim that $\{u_{\gamma_1}, \ldots, u_{\gamma_k}\}$ is a dominating set of graph G. Let $l \in [0, v-1]$. We show that either vertex u_l is part of the proposed set, or it is adjacent to one

of the elements in the proposed set. Consider time units $(T_0 + l \cdot (2v) + j_0 + v) = \lambda(\alpha_0, l, 0) + j_0$ and $\lambda(\alpha_0, l, 0) + j_0 - v$ which are both in range α_0. Among the jobs scheduled at these times we have job floor $f_{\lambda(\alpha_0,l,0)}$, floor gadget $f'_{\lambda(\alpha_0,l,0)}$ (resp. job floor $f_{\lambda(\alpha_0,l-1,0)+v}$, floor gadget $f'_{\lambda(\alpha_0,l-1,0)+v}$) and k selector jobs, one per selector chain. So at most $k-1$ selector gadgets can be scheduled at each of these times due to the $2k+1$ machines. Consequently there is $i \in [1,k]$ such that selector gadget $s'_{i,\lambda(\alpha_0,l,0)+j_0-j_i}$ does not exist. And similarly there is $i' \in [1,k]$ such that $s'_{i',\lambda(\alpha_0,l,0)+j_0-j_{i'}-v}$ does not exist either.

- If $j_i \geq j_0$ then $s'_{i,\lambda(\alpha_0,l,0)+j_0-j_i} = s'_{i,\lambda(\alpha_0,l,\gamma_i)}$. This selector gadget does not exist, so either $l = \gamma_i$ or $(v_l, v_{\gamma_i}) \in E$.
- If $j_i < j_0$ then $s'_{i',\lambda(\alpha_0,l,0)+j_0-j_{i'}-v} = s'_{i',\lambda(\alpha_0,l,\gamma_{i'})}$. This selector gadget does not exist, so either $l = \gamma_{i'}$ or $(v_l, v_{\gamma_{i'}}) \in E$.

So in both cases we confirm that vertex v_l is either part of $\{u_{\gamma_1}, \ldots, u_{\gamma_k}\}$ or adjacent to one of these vertices. Therefore the proposed set is indeed a dominating set of size k in graph G. □

This concludes the reduction and the proof of Theorem 127. This confirms that contrary to pathwidth μ a FPT algorithm with respect to proper level q is unlikely on problem $P|prec, p_j = 1, r_j, \bar{d_j}|\star$.

6.5 Summary & Concluding Remarks

Parameter	Section	Problem / Setting	Result
q	6.2	-	$q \leq \mu$
		$P2\|r_j, \bar{d}_j\|\star$ $1\|(1,\ell,1), r_j, \bar{d}_j\|\star$	Inferred para-NP-hardness results.
	6.3	$1\|prec, r_j, \bar{d}_j\|C_{max}$	FPT in time $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N + n^2)$.
	6.4	$P\|prec, p_j = 1, r_j, \bar{d}_j\|\star$	para-NP-hard$(q + D)$.
			$W[2]$-hard$(q + w)$.

Figure 6.6: Summary of the results obtained in this chapter.

In this chapter we introduced a new scheduling parameter called the proper level q. It refined the count of overlapping time windows from pathwidth μ to only situations where such overlaps simulated a PARTITION subproblem and thus were believed to be truly problematic. As such we identified q as a weaker parameter than pathwidth μ (in general) and slack σ (on a single machine and on a fixed number of identical parallel machines). Nevertheless we gave a FPT algorithm on single machine scheduling with job time windows and precedence relations, which was a direct improvement of the FPT result with respect to pathwidth μ given in Chapter 4. When upgrading to the identical parallel setting we showed that, unlike with pathwidth μ, scheduling unit jobs with time windows and general precedence is para-NP-hard with respect to q.

Unlike the single machine case, it is clear that the nature of the precedence relations impacts the parameterized complexity of parallel machine scheduling with unit jobs with respect to proper level q. While we showed hardness with general precedence and Baptiste et al. solved the problem in polynomial time for chains [BBKT04], the case of tree-like precedence is left open. Such a result would definitely unveil deeper connections between precedence relations and job time windows in the parallel machine setting.

6.5.1 Problem Map with Parameter q

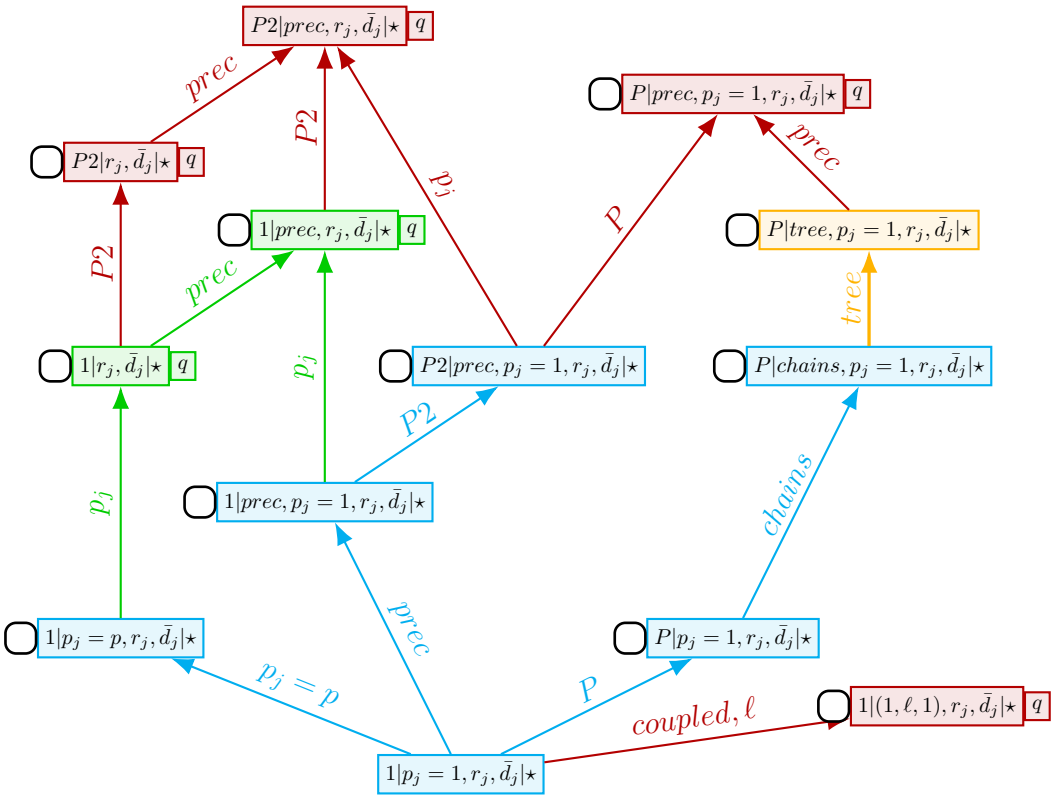


Figure 6.7: Problem map of parameter q.

Chapter 7

Miscellaneous Parameterized Results

In this chapter we present several other parameterized results obtained during this thesis and give input on their potential uses in scheduling problems with job time windows. Section 7.1 gives a first insight on the existence of polynomial kernels for the single machine scheduling problem with job time windows. We first establish that a polynomial kernel is unlikely to be found with pathwidth μ. In contrast we provide a polynomial kernel when the problem is parameterized by the vertex cover of the time windows comparability graph. Section 7.2 considers other width parameters based on job time window interval graph $G_{[r_j,d_j)}$, notably the twin-width tww. Although this parameter has led to multiple FPT results on graph problems, here we present some negative results on single machine scheduling and parallel machine scheduling with unit jobs. Finally in Section 7.3 we introduce a notion of average parameter. Since most of the considered parameters are a maximum value and can be made artificially high from a single outlier, the associated average parameter is expected to reflect better the actual difficulty of the given instance. We investigate conditions under which such average parameters can lead to FPT algorithms. We end by summarizing the results obtained in this chapter and giving our concluding remarks in Section 7.4.

7.1 Kernelization Algorithms

It is well known that whenever a problem $\mathcal{Q}$ is FPT with respect to some parameter k, there is a corresponding kernelization algorithm with respect to k [FG98]. Given a parameterized instance I of $\mathcal{Q}$ with parameter value $k(I)$, a kernelization is a polynomial-time algorithm which outputs an instance I' of $\mathcal{Q}$ with parameter value $k(I')$ such that I' is equivalent to I and the size of the parameterized instance $|I'| + k(I')$ is bounded by $f(k(I))$ for some computable function f. Such an instance I' is called a kernel. The kernel is called polynomial

if f is a polynomial function. There is a general way to infer a kernelization from any FPT algorithm. However, having a kernelization which actively builds the kernel from the input generally gives more information on the nature and shape of the set of kernels in problem $\mathcal{Q}$.

While such kernelizations have already been found for some scheduling problems [BJ22, JKZ24, KK22], they were done in the context of high-multiplicity scheduling with the number of job types among the parameters. This avoids dealing with an unbounded number of job types. Considering other structural parameters, we need to reduce the instance size - i.e. both the number of jobs and the data values - to a function of the parameter. To our knowledge, no such kernelization algorithm has been produced in the literature.

In this section we consider the single machine scheduling problem with job time windows and precedence constraints - denoted by $1|prec, r_j, \bar{d}_j|\star$. We will show that with pathwidth μ this problem is unlikely to have a polynomial kernel. As a result we propose a new structural based on job time window interval graph $G_{[r_j,d_j)}$ for which this will be the case:

Definition 132. *A vertex cover of a graph $G = (V, E)$ is a subset of V which covers all edges in E - i.e. given any edge in E at least one of its nodes is in the vertex cover. Parameter vc is defined as the minimum size of any vertex cover in $G_{[r_j,d_j)}$.*

Throughout this section we use the following example to illustrate our algorithms. We consider a set of 23 jobs indexed in $\{1, \ldots, 23\}$:

- For $i \in \{1, \ldots, 20\}$, $r_i = 10 \cdot (i-1)$ and $\bar{d}_i = 10 \cdot i$. We have $p_i = 2$ if $i = 1$ or $i = 20$ and $p_i = 5$ otherwise.
- Jobs $21, 22, 23$ have processing time 10, release times and deadlines $r_{21} = r_{22} = 0, \bar{d}_{21} = \bar{d}_{21} = 100$, and $r_{23} = 90, \bar{d}_{23} = 200$.
- We have a unique precedence constraint $(22, 23)$.

Now in this example the graph $G_{[r_j,d_j)}$ is composed of a triangle $21-22-23$, plus edges between $21, 22$ and all jobs $\{1, \ldots, 10\}$, and edges between 23 and all jobs $\{10, \ldots, 20\}$. Observe that jobs $\{1, \ldots, 20\}$ define an independent set. So in our example we have $vc = 3$, corresponding to the vertex cover defined by the three nodes $\{21, 22, 23\}$. Also note that the maximum clique size is 4 - and thus $\mu = 3$ - e.g. with jobs $21, 22, 23, 10$, the time windows of which overlap at time 90.

As with other common graph parameters like treewidth or pathwidth, it is NP-complete to compute $vc(G)$ for general graphs [GJ79]. However a vertex cover of size $vc(G)$ can be computed in linear time when restricted to interval graphs (see e.g. [MRR92] by Marathe et al.), so parameter vc can be computed in linear time. This opens up potential uses in FPT algorithms. Bessy and Giroudeau considered this parameter in the context of coupled-task scheduling with a compatibility graph $G_c = (\mathcal{J}, E)$ [BG19]. They showed that computing

the minimum makespan is FPT parameterized by $vc(G_c)$ plus the maximum length of a coupled task.

Parameter vc is closely related to pathwidth μ as shown by the following proposition:

Proposition 133. *In any scheduling instance featuring job time windows we have:* $\mu \leq vc$

Proof. In an instance with pathwidth μ the job time window interval graph admits a clique of size $\mu + 1$. This forces at μ jobs to be included in any vertex cover of this graph. □

This means that any problem FPT for parameter μ is also FPT for vc. We can thus derive the following result from one of our previous works [HMMK22], which provides two FPT algorithms for two problems parameterized by μ:

Corollary 134. $1|prec, r_j, \bar{d}_j|L_{max}$ *and* $P|prec, p_j = 1, r_j, \bar{d}_j|L_{max}$ *are* FPT *parameterized by* vc.

This section is organized as follows. First we prove that under usual parameterized complexity assumptions, the problem cannot have a polynomial kernel for parameter μ. In contrast in the second subsection we propose a first polynomial kernel for parameters vc and the maximum processing time of a job $p_{\max}$ combined. Then in the third subsection we provide an additional reduction rule allowing for a second polynomial kernel with parameter vc only. Eventually in the fourth subsection we prove that unlike the single machine case, the vertex cover parameter is not sufficient to derive any FPT algorithm - and thus a kernelization algorithm - in the case of parallel machines with processing set restrictions (a setting notably mentioned by Leung in [LL08]).

7.1.1 A Kernel Lower Bound with Parameter μ

In this subsection we use the cross-composition technique developed in [BJK14] to show that our single machine problem, although FPT parameterized by pathwidth μ, does not admit a polynomial kernel under usual parameterized complexity assumptions.

Theorem 135. *Unless* $NP \subseteq coNP/poly$ *there is no polynomial kernel for* $1|r_j, \bar{d}_j|\star$ *parameterized by pathwidth* μ.

We consider here a special case of cross-composition where the equivalence relation mentioned in the original definition of [BJK14] decides whether a string is a valid instance of a decision problem or not.

Definition 136. *Let* Σ *be an alphabet and let* $L \subset \Sigma^*$ *and* $C \subset \Sigma^*$ *with* $L \subset C$ *be two languages over* Σ. *Let* $Q \subset \Sigma^* \times \mathbb{N}$ *be a parameterized problem.* L *cross-composes into* Q *if there is an algorithm which, given* t *strings* $x_1, x_2, \ldots, x_t$ *in* C *computes an instance* $(x, k) \in \Sigma^* \times \mathbb{N}$ *in time polynomial in* $(\Sigma_{i=1}^{t}|x_i|)$ *such that:*

1. *deciding whether a given $x \in \Sigma^*$ is in C can be done in $\mathcal{O}(|x|^{\mathcal{O}(1)})$,*
2. *$(x,k) \in Q \iff x_i \in L$ for some $1 \leq i \leq t$,*
3. *k is bounded by a polynomial in $\max_{1\leq i\leq t} |x_i| + \log(t)$.*

We consider here that L will be the set of feasible instances of the PARTITION problem, whereas C will be the set of instances of the PARTITION problem, which is defined as follows:

Definition 137. *An instance of PARTITION is defined by a set A of n integers $a(1), \ldots, a(n)$ and an integer B such that $\sum_{i=1}^{n} a(i) = 2B$. The instance is feasible if there exists a subset $X \subset A$ such that $\sum_{a(i)\in X} a(i) = B$.*

Consider t instances of the PARTITION problem. We denote n_k the number of integers of the instance I_k, by $a_k(i)$ the i^{th} integer of instance I_k, and by B_k its threshold.

From these t instances we build an instance $\mathcal{J}(I_1, \ldots, I_t)$ of $1|r_i, \bar{d_i}|\star$ parameterized by μ - i.e. our single machine problem with an empty precedence relation. We first set $T = 2 \cdot (\max_{1\leq i\leq t} B_i)$, and for all $1 \leq k \leq t$: $D_k = (T+1)\cdot k + 2\cdot(\sum_{i=1}^{k}(B_i+1))$. We notably have: $D_k - D_{k-1} = 2B_k + T + 3$.

We define a job u with processing time T and time window $r_u = 0$, $\bar{d_u} = D_t$. Then all jobs associated to instance I_k will have their time window between D_{k-1} and D_k. To each instance I_k are associated three unit-time fill jobs. These jobs are delimiters for the other jobs schedule and must be performed at their release date in any feasible schedule:

	$f_{k,1}$	$f_{k,2}$	$f_{k,3}$
r_i	$D_{k-1} + B_k$	$D_{k-1} + 2B_k + 1$	$D_k - 1$
d_i	$D_{k-1} + B_k + 1$	$D_{k-1} + 2B_k + 2$	D_k

For each integer $a_k(i)$, a job of processing time $a_k(i)$, release time D_{k-1} and deadline $D_{k-1} + 4B_k + 2$ $(= D_k - T - 1 + 2B_k)$. Figure 7.1 shows the cross-composition for two instances.

Lemma 138. *The instance $\mathcal{J}(I_1, \ldots, I_t)$ is a cross-composition.*

Proof. The polynomiality of the construction is straightforward. Let us now compute the value of the parameter μ. As each instance I_k has all its associated jobs within the interval $[D_{k-1}, D_k)$, which interferes with at most one fill job and the job u at a given time, so that the pathwidth of the instance is $\mu = 1 + \max_{1\leq k\leq t} n_k$. As the size of I_k is a function of n_k our parameter satisfies the cross-composition condition.

We finally have to verify that the instance $\mathcal{J}(I_1, \ldots, I_t)$ is feasible if and only if one of the PARTITION instance is feasible.

$\Longleftarrow$ Assume that the PARTITION instance I_k is feasible. Let X_k be the subset of values such that $\sum_{a_k(i)\in X_k} a_k(i) = B_k$. We build a feasible schedule by scheduling job u at time $D_k - T - 1$, the tasks of X_k in sequence in the interval

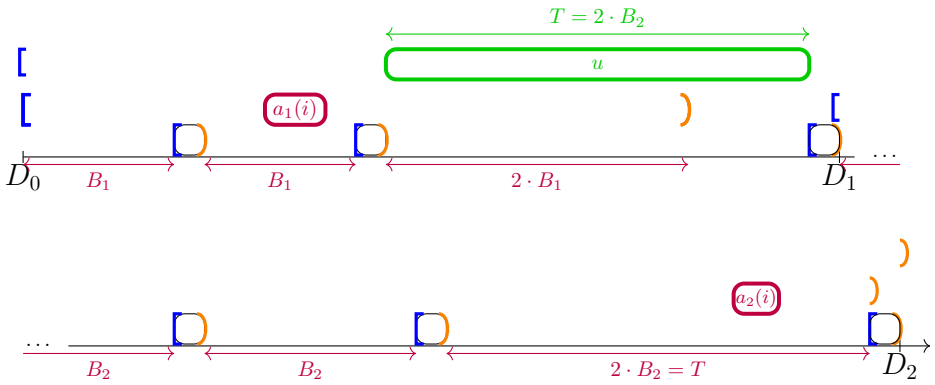


Figure 7.1: Cross-composition from two instances I_1, I_2 of PARTITION.

$[D_{k-1}, D_{k-1} + B_k)$, and the other tasks $a_k(j)$ are performed in sequence in interval $[D_{k-1} + B_k + 1, D_{k-1} + 2B_k + 1)$. The jobs $a_l(i)$ of the other instance are all performed in any order in the interval $[D_{l-1} + 2B_l + 2, D_{l-1} + 4B_l + 2)$. Job u does not interfere with any other job, so the schedule is feasible.

$\Longrightarrow$ Let us observe that in any feasible schedule, there is a k such that job u is performed at time $D_k - 1 - T$ between $f_{k,2}$ and $f_{k,3}$. Indeed, the size of the interval between two other fill jobs is always strictly less than T. Now the other jobs $a_k(i)$ cannot be scheduled in interval $[D_k - 1 - T, D_k)$ that perform u and a fill job. So they can only be executed before $f_{k,1}$ or between $f_{k,1}$ and $f_{k,2}$. If all jobs are executed this defines a partition of this instance: X_k is the set of jobs performed before $f_{k,1}$. □

It is well known that the PARTITION problem is NP-hard [GJ79]. From Theorem 9 of [BJK14], we immediately derive the following result:

Corollary 139. *Unless $NP \subseteq coNP/poly$ there is no polynomial kernel for $1|r_i, \bar{d}_i|\star$ parameterized by pathwidth μ.*

7.1.2 A Polynomial Kernel with Parameter $vc + p_{max}$

In this subsection we propose a polynomial kernel of $1|prec, r_j, \bar{d}_j|\star$ with respect to parameters vc plus $p_{\max} = \max_{i \in \mathcal{J}} p_i$ with size $\mathcal{O}(\log(vc^3 \cdot p_{\max}) \cdot vc^2)$.

Theorem 140. *With respect to parameter $vc + p_{\max}$ problem $1|prec, r_j, \bar{d}_j|\star$ has a polynomial kernel in time $\mathcal{O}(n^2 \cdot log(n))$. The resulting kernel has size $\mathcal{O}(log(vc^3 \cdot p_{\max}) \cdot vc^2)$.*

Let I be an instance of $1|prec, r_j, \bar{d}_j|\star$. Let $\mathcal{V} = (v_i)_{1 \leq i \leq vc}$ be a vertex cover of $G_{[r_j, d_j)}$ with minimum size vc. Let $(u_\alpha)_\alpha$ be the increasing sequence of release date/deadline values of the jobs from $\mathcal{V}$. We denote vc_α the number

of jobs of vertex cover $\mathcal{V}$ such that $[u_\alpha, u_{\alpha+1})$ is included in their time window : $vc_\alpha = |\{j \in \mathcal{V}, r_j \leq u_\alpha \text{ and } u_{\alpha+1} \leq \bar{d}_j\}|$

For our example, we get $\mathcal{V} = \{21, 22, 23\}$, $u_0 = 0, u_1 = 90, u_2 = 100, u_3 = 200$, and $vc_0 = 2, vc_1 = 3, vc_2 = 1$.

First note that the jobs outside the vertex cover have limited time window possibilities :

Proposition 141. *Given a vertex cover $\mathcal{V}$ of $G_{[r_j,d_j)}$ the jobs in $\mathcal{J} - \mathcal{V}$ form a sequence $(w_j)_{1\leq j\leq|\mathcal{J}-\mathcal{V}|}$ such that $\bar{d}_{w_j} \leq r_{w_{j+1}}$ for all $1 \leq j < |\mathcal{J} - \mathcal{V}|$.*

Proof. By the definition of $\mathcal{V}$ all job pairs in $\mathcal{J} - \mathcal{V}$ must have an empty time window intersection. So we get sequence $(w_j)_{1\leq j\leq|\mathcal{J}-\mathcal{V}|}$ by ordering the jobs in $\mathcal{J} - \mathcal{V}$ by increasing release dates. □

Definition 142. *Let w_j, w_{j+1} be two consecutive jobs of $\mathcal{J}-\mathcal{V}$. In any schedule of these two jobs we call gap the time interval $[s_{w_j} + p_{w_j}, s_{w_{j+1}})$. We call delimiters of the gap the two jobs w_j, w_{j+1}. By extension we call gap g_j the tuple w_j, w_{j+1}. The maximum capacity of the gap is obtained by left shifting w_j and right shifting w_{j+1} and thus equals $\bar{d}_{w_{j+1}} - p_{w_{j+1}} - r_{w_j} - p_{w_j}$. In some interval $[u_\alpha, u_{\alpha+1})$ a gap g_j is an inner gap if in any schedule it is included in the interval $[u_\alpha, u_{\alpha+1})$, so if $r_{w_j} \geq u_\alpha$ and $\bar{d}_{w_{j+1}} \leq u_{\alpha+1}$. Other gaps are called border gaps.*

Proposition 141 implies that the existence of a feasible schedule relies on the feasibility of inserting jobs of $\mathcal{V}$ into gaps defined by the sequence of jobs of $\mathcal{J} - \mathcal{V}$. When placing ourselves in some interval $[u_\alpha, u_{\alpha+1})$ - i.e. where no release date/deadline from the vertex cover is interfering - this is reminiscent of the multiple knapsack problem [GRR19] except the gap capacities can be adjusted from both ends with the schedule of their delimiters, and doing so impacts the capacity of the neighboring gaps. As the number of knapsacks can be bounded by the number of items in multiple knapsack, such analogy suggests that only a bounded number of gaps is enough to keep the equivalence with the original instance. We first prove that some gaps remain unused in dominant schedules in each interval $[u_\alpha, u_{\alpha+1})$.

Lemma 143. *If the instance I is feasible then there is a schedule where any job $i \in \mathcal{V}$ with $r_i \leq u_\alpha, \bar{d}_i \geq u_{\alpha+1}$ is scheduled either in a border gap or in one of the $3vc_\alpha$ highest maximum capacity inner gaps of the interval $[u_\alpha, u_{\alpha+1})$.*

Proof. If there are less than $3vc_\alpha$ inner gaps, the lemma is obvious. Assume there are more than $3vc_\alpha$ inner gaps. The key idea of the proof is that if a gap w_j, w_{j+1} is at maximum capacity, this may reduce its two neighbors gaps w_{j-1}, w_j and w_{j+1}, w_{j+2} but does not impact the size of the farther gaps. So by considering $3vc_\alpha$ gaps we can have vc_α gaps among them at maximum capacity.

Let $g_1, \ldots, g_{3\cdot vc_\alpha}$ be the $3\cdot vc_\alpha$ inner gaps with the highest maximum capacity. Let τ be a feasible schedule on instance I. Then in interval $[u_\alpha, u_{\alpha+1})$ at most vc_α gaps included in $[u_\alpha, u_{\alpha+1})$ are filled by jobs from vertex cover $\mathcal{V}$. Let $g'_1, \ldots, g'_k$ be these gaps ($k \leq vc_\alpha$). We show that the jobs scheduled in these

gaps can be scheduled in gaps $g_1, \ldots, g_{3 \cdot vc_\alpha}$ instead while keeping feasibility. We do so by mapping each gap g'_i to some gap g_j with larger or equal maximum capacity.

If a gap g'_i is already one of the $3 \cdot vc_\alpha$ gaps with the highest maximum capacity then we assign it to itself. Then for all other gaps g'_i we assign a gap g_j which is a neighbor of no other assigned gap $g_{j'}$. This allows us to use gap g_j at its maximum capacity, which is necessarily higher than or equal to equal to the capacity used by schedule τ in gap g'_i. Since each gap has at most two neighbors, at any iteration of this process there can only be at most $3(k-1) < 3 \cdot vc_\alpha$ gaps $g_1, \ldots, g_{3 \cdot vc_\alpha}$ which are either already assigned or a neighbor of an already assigned gap. So for each gap g'_i used in schedule τ a suitable gap g_j can be found in time $\mathcal{O}(vc_\alpha)$. Plus *prec*-consistency ensures that within every interval $[u_\alpha, u_{\alpha+1})$ there is no precedence relation between a job from the vertex cover and an inner job from $\mathcal{J} - \mathcal{V}$. So the jobs in τ can move to their assigned gaps without invalidating any precedence constraint. Thus the schedules which only use the (at most) $3 \cdot vc_\alpha$ inner gaps with the highest maximum capacity in each interval $[u_\alpha, u_{\alpha+1})$ are dominant. □

We now need to provide reduction rules that will reduce the set of jobs and the values of release times and deadlines by keeping only useful gaps.

To this purpose, we first define a basic transformation of the instance, called a Left Shift and Replace, that will be used in the next reduction rule.

Definition 144. *Let I be an instance of our problem,w_i, w_{i+1} and w_j, w_{j+1} two gaps with $i+1 < j$. We call Left Shift and Replace and denote $LSR(I, i, j)$ the following modified instance, illustrated by figure 7.2, in which the time interval between $\bar{d}_{w_{i+1}}$ and r_{w_j} is reduced to a single time unit occupied by a substitute job, by applying a negative offset to all release times and deadlines greater than or equal to r_{w_j}. Formally:*

1. *remove all jobs w_k with $i+1 < k < j$,*
2. *add a new substitute job w' with $p_{w'} = 1, r_{w'} = \bar{d}_{w_{i+1}}$ and $\bar{d}_{w'} = r_{w'} + 1$,*
3. *apply an offset of $-(r_{w_j} - \bar{d}_{w'})$ to all release times and deadlines of jobs w_k with $k \geq j$ and jobs $x \in \mathcal{V}$ such that $r_x \geq r_{w_j}$.*

A Left Shift and Replace is said to be valid if: I is feasible $\iff$ $LSR(I, i, j)$ is feasible.

Observe that the gaps introduced by the substitute jobs w' are gaps w_{i+1}, w' and w', w_j. Notice that $r'_w = \bar{d}_{w_{i+1}}$, so that the maximum capacity of the gap w_{i+1}, w' is less than the maximum capacity of the gap w_{i+1}, w_{i+2} in the original instance. Similarly, the maximum capacity of gap w', w_j is less than the maximum capacity of w_{j-1}, w_j in the original instance, since the time window of w_j is left shifted in instance $LSR(I, i, j)$ so that its release time equals the deadline of w'. Lemma 145 states conditions for which $LSR(I, i, j)$ is valid.

Lemma 145. *Let I be an instance such that there exists a feasible schedule for which no job of $\mathcal{V}$ is scheduled in the gaps whose delimiters are in $w_{i+1}, \ldots, w_j$ which are all inner gaps in an interval $[u_\alpha, u_{\alpha+1})$. Then $LSR(I, i, j)$ is valid.*

Proof. Let s be a feasible schedule of I with the property stated in the Lemma. We can easily build a schedule s' of $LSR(I, i, j)$ by applying an offset $-(r_{w_j} - \bar{d}_{w'})$ to all starting times after r_{w_j} (i.e. $s'(k) = s(k) - (r_{w_j} - \bar{d}_{w'})$ if $s(k) > r_{w_j}$). Then, if $s(k) \leq r_{w'}$, then $s'(k) = s(k)$. The schedule is feasible. As we assumed that the jobs of the vertex cover were not scheduled in the removed gaps, they are scheduled in the gaps to the left or to the right of w' in instance $LSR(I, i, j)$. These gaps have the same length after the offset.

Conversely if we have a feasible schedule s' of $LSR(I, i, j)$, then to build a schedule s of the original instance we just apply an offset $(r_{w_j} - \bar{d}_{w'})$ to all starting times $\geq \bar{d}_{w'}$. Then the removed jobs w_{i+2} to w_{j-1} are scheduled at their release times in sequence. Indeed, the gaps w_{i+1}, w_{i+2} has then the same length as w_{i+1}, w' in the modified instance, and the same argument hold for gap w_{j-1}, w_j. As we assumed *prec*-consistent deadlines, if the time window of w_k is included in $[u_\alpha, u_{\alpha+1})$ there cannot be a precedence constraint between w_k and a job x of $\mathcal{V}$ with $r_x \leq u_\alpha$ and $\bar{d}_x \geq u_{\alpha+1}$. So no precedence constraint is violated. □

We can now define our first reduction rule.

Reduction Rule 1. *For each α : if there are more than $3vc_\alpha$ inner gaps in $[u_\alpha, u_{\alpha+1})$, select the $3vc_\alpha$ ones with highest maximum capacity. We denote S the set of selected gaps to which we add the border gaps. While there are two non consecutive gaps w_i, w_{i+1}, w_j, w_{j+1} in S without a substitute job, update the instance: $I \leftarrow LSR(I, i, j)$.*

Figure 7.2 shows the reduction process.

Applying the reduction to our example would give the following result. In time interval $[u_2, u_3) = [100, 200)$, $vc_2 = 1$, and the three maximum capacity inner gaps are $(17, 18), (18, 19)$ with maximum capacity 10, and $(19, 20)$ with maximum capacity 13. The border gap is $(10, 11)$ So, jobs 12 to 16 can be removed and replaced by substitute job b with release date 110 and deadline 111. The release times and deadlines of jobs $17, 18, 19, 20, 23$ decrease by $160 - 111 = 49$.

In time interval $[u_1, u_2)$ no job can be removed, jobs 10, 11 are still in the instance. In time interval $[u_0, u_1) = [0, 90)$, $vc_0 = 2$, we have 10 jobs in $\mathcal{J} - \mathcal{V}$, delimiting 9 gaps fully in interval $[0, 90)$. Gap $(1, 2)$ has maximum capacity 13 (by left shifting job 1 and right shifting job 2) while the other gaps have maximum capacity 10. So we can select only the $6 = 3 \cdot vc_0$ first gaps, add the border gap $(9, 10)$ and remove job 8. Substitute job a replaces 8 with release time 70 and deadline 71. Then release times and deadlines greater than 80 decrease by $80 - 71 = 9$. This leads to the following reduced instance:

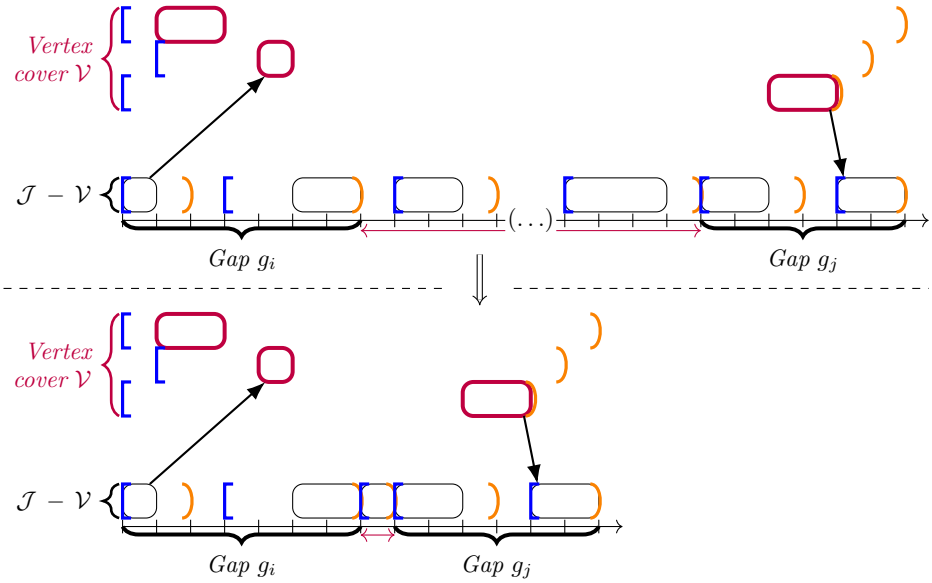


Figure 7.2: Job removal process between two consecutive non-neighboring chosen gaps g_i and g_j.

i	1	2	3	4	5	6	7	a	9	10	11	b	17	18	19	20	21	22	23
p_i	2	5	5	5	5	5	5	1	5	5	5	1	5	5	5	2	10	10	10
r_i	0	10	20	30	40	50	60	70	71	81	91	101	102	112	122	132	0	0	81
d_i	10	20	30	40	50	60	70	71	81	91	101	102	112	122	132	142	81	81	142

Lemma 146. *In each interval $[u_\alpha, u_{\alpha+1})$ of a prec-consistent instance reduction 1 is a valid. After its application, the number of jobs w_j from $\mathcal{J} - \mathcal{V}$ can be reduced down to $\mathcal{O}(vc^2)$.*

Proof. Lemma 143 establishes that if I is a feasible instance,there exists a schedule such that no job of $\mathcal{V}$ is scheduled in a non selected gap. And Lemma 145 indicates that if there are two non consecutive selected gaps, then the Left Shift and Replace rule is valid. So that iterating on α and on LSR reduction keeps the equivalence between instances, so that reduction rule is valid.

Let us now count the number of jobs. The number of substitute jobs introduced in interval $[u_\alpha, u_{\alpha+1})$ is at most $3vc_\alpha - 1$, plus two delimiters of the left and right non inner gap in the interval $[u_\alpha, u_{\alpha+1})$ By adding the two delimiters of the at most $3 \cdot vc_\alpha$ gaps, we successfully bound the number of jobs from $\mathcal{J} - \mathcal{V}$ in each interval $[u_\alpha, u_{\alpha+1})$ by $9 \cdot vc_\alpha + 1$. As there are at most $2vc$ possible values of α, the lemma holds. □

Now the values of the release times and deadlines are still not related to our parameters. We define reduction rule 2 that allows to further reduce release

times and deadlines so that their values depend only on vc and $p_{\max}$. Let I be an instance of the problem, and let $(v_\beta)_{\beta\in\{1,\dots,y\}}$ be the sorted sequence of release dates and deadlines of jobs in $\mathcal{J}$. Notice that in each interval $[v_\beta, v_{\beta+1})$ at most $vc+1$ jobs could be performed (the jobs of $\mathcal{V}$ and the only job w_j (if any) whose time window crosses this interval). We propose the following reduction rule:

Reduction Rule 2. *If $v_{\beta+1}-v_\beta > (vc+1)\cdot p_{\max}$, then set $t = v_\beta+(vc+1)\cdot p_{\max}$. Apply an offset of $-(v_{\beta+1}-t)$ to all release times and deadlines $\geq v_{\beta+1}$.*

Figure 7.3 shows the reduction principle. Notice that it cannot be applied to our example.

Lemma 147. *Reduction rule 2 is valid.*

Proof. If $v_{\beta+1} - v_\beta > (vc+1)\cdot p_{\max}$ then if there is a feasible schedule, we know that the load of interval $[v_\beta, v_{\beta+1})$ is at most $(vc+1)\cdot p_{\max}$. As no release time or deadline is strictly included in the interval we can assume that there is a feasible schedule where the jobs are left shifted in this interval, so that the interval $[v_\beta+(vc+1)\cdot p_{\max}, v_{\beta+1})$ is empty. So, by left shifting every start time release date and deadline above $v_{\beta+1}$ we build a feasible schedule of the new instance. And conversely we just have to apply an offset of $(v_{\beta+1}-t)$ to all dates above $v_{\beta+1}$ to get a feasible schedule of the original instance. □

This gives a polynomial kernel with respect to parameter $vc+p_{\max}$.

Proposition 148. *With respect to parameter $vc+p_{\max}$ problem $1|prec, r_j, \bar{d}_j|\star$ has a polynomial kernel in time $\mathcal{O}(n^2\cdot log(n))$. The resulting kernel has size $\mathcal{O}(log(vc^3\cdot p_{\max})\cdot vc^2)$ and $\mathcal{O}(vc^2)$ jobs.*

Proof. Consider the instance I' build from instance I after applying iteratively all our reduction rules. To build I', we begin by ensuring that the instance is *prec*-consistent in time $\mathcal{O}(n^2)$. Then a vertex cover of minimum size vc is computed in time $\mathcal{O}(|G_{[r_j,d_j)}|) = \mathcal{O}(vc^2)$ [MRR92]. After that the sequence $(u_\alpha)_\alpha$ of interval bounds is computed in time $\mathcal{O}(vc\cdot log(vc))$. Finally Lemma 146 can be applied to remove all but $\mathcal{O}(vc^2)$ jobs from $\mathcal{J}-\mathcal{V}$. This gives an equivalent instance $\hat{I}$ with $\mathcal{O}(vc^2)$ release times and deadlines. Now according to reduction rule 2, after iterative application of the rule the distance between two consecutive release dates or deadlines v_β is $\mathcal{O}(vc\cdot p_{\max})$. So the maximum deadline of any job is $\mathcal{O}(vc^3\cdot p_{\max})$. Thus the encoding of each release dates/deadline in I' is $\mathcal{O}(\log(vc^3\cdot p_{\max}))$. □

7.1.3 A Polynomial Kernel with Parameter vc

In this subsection we give an additional reduction rule which allows us to reduce the size of the kernel down to a polynomial in vc.

Theorem 149. *Problem $1|prec, r_j, \bar{d}_j|\star$ has a polynomial kernel with respect to parameter vc. The resulting kernel has size $\mathcal{O}(vc^8)$.*

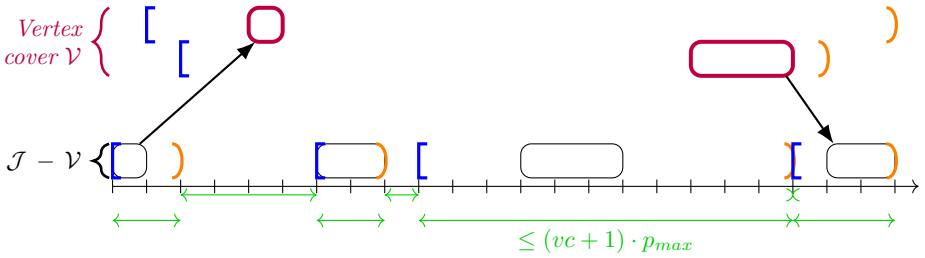


Figure 7.3: Illustration of the bounds between the release dates and deadlines of the jobs in $\mathcal{J} - \mathcal{V}$.

Note that in our previous kernel, $p_{\max}$ is only needed to bound the size of the time values. This dependency can be lifted by using the following theorem:

Theorem 150. *[FT87] Let $\mathbf{w} = (w_1, \ldots, w_d)$ be a vector of rational numbers and let N be a positive integer. Then in time $\mathcal{O}(d^8 \cdot (d + log(N)))$ one can build a vector $\mathbf{w}' = (w'_1, \ldots, w'_d)$ of positive integers such that:*

- *$w'_i \leq 2^{4d^3} \cdot N^{d(d+2)}$ for all $1 \leq i \leq d$,*
- *The sign of $\mathbf{w} \cdot \mathbf{x}$ equals the sign of $\mathbf{w}' \cdot \mathbf{x}$ for every $\mathbf{x} = (x_1, \ldots, x_d)$ vector of integers such that $\sum_{i=1}^{n} |x_i| \leq N - 1$.*

Given an instance I of $1|prec, r_j, \bar{d}_j|\star$ and let $\kappa(I)$ be the $(vc + p_{\max})$-kernel obtained with Proposition 148. Let $\tilde{n}$ be the number of its jobs. Then the number of time values we want to compress is $3\tilde{n}$, which is bounded by $\mathcal{O}(vc^2)$ by Proposition 148. So the idea is to formulate problem $1|prec, r_j, \bar{d}_j|\star$ into an integer program such that the value N needed to express all the constraints is bounded by a function of vc. Then we will set $d = 3\tilde{n}$, $\mathbf{w} = (p_1, r_1, \bar{d}_1, \ldots, p_{\tilde{n}}, r_{\tilde{n}}, \bar{d}_{\tilde{n}})$ and apply the theorem to get an equivalent instance with time values $\mathbf{w}' = (p'_1, r'_1, \bar{d}'_1, \ldots, p'_{\tilde{n}}, r'_{\tilde{n}}, \bar{d}'_{\tilde{n}})$ all bounded by a function of vc.

We consider the following integer program with positional variables:

$$x_{i,j} = \begin{cases} 1 & \text{if job } j \text{ is the } i^{th} \text{ job in the schedule,} \\ 0 & \text{otherwise.} \end{cases}$$

Linear program $BIP(I)$ is defined as follows:

$$\begin{cases} \text{One job per position} & \sum_{j=1}^{n} x_{i,j} = 1 \text{ for all } 1 \leq i \leq n. \\ \text{One position per job} & \sum_{i=1}^{n} x_{i,j} = 1 \text{ for all } 1 \leq j \leq n. \\ \text{Time windows} & \sum_{j=1}^{n} r_j \cdot x_{i_1,j} + \sum_{i=i_1}^{i_2} \sum_{j=1}^{n} p_j \cdot x_{i,j} \leq \sum_{j=1}^{n} \bar{d}_j \cdot x_{i_2,j} \\ & \text{for all } 1 \leq i_1 \leq i_2 \leq n. \\ \text{Precedence constraint} & \sum_{i=1}^{i_0} (x_{i,j_1} - x_{i,j_2}) \geq 0 \text{ for all } 1 \leq i_0 \leq n, (j_1, j_2) \in A. \end{cases}$$

This binary program was proved equivalent to the associated scheduling instance in [LQ92].

Lemma 151. *[LQ92] An instance I of $1|prec, r_j, \bar{d}_j|\star$ is feasible if and only if binary integer program $BIP(I)$ has a solution.*

We now intend to meet the conditions of Theorem 150 by providing bounds on the variables.

Lemma 152. *In any solution of $BIP(I)$ at most $n+2$ variables are equal to one in each constraint.*

Proof. At most n variables equal one for the two first set of constraints (one job per position and one position per job). Now, considering the time window constraint, it sums up with release time coefficient the variables of jobs in position i_1 (so only 1 variable equals one), similarly the third term of the sum considers the deadlines of jobs in position i_2, so only one of them equals one, whereas the intermediate sum considers all positions between i_1 and i_2, so at most n variables equal one. This gives our $(n+2)$ bound. Finally for the precedence constraints, only two variables might be equal to 1 in the sum. □

Then we get a kernel of size polynomial in vc by pairing Lemma 151 with Theorem 150. This gives the following reduction rule:

Reduction Rule 3. *Let I be an instance I and $\kappa(I)$ be the instance obtained after applying iteratively reductions 1 and 2. Let $\tilde{n}$ be the number of jobs in $\kappa(I)$. Then we set vector $\mathbf{w} = (p_1, r_1, \bar{d}_1, \ldots, p_{\tilde{n}}, r_{\tilde{n}}, \bar{d}_{\tilde{n}})$ from the time values of $\kappa(I)$ and apply Theorem 150 with $d = 3\tilde{n}$ and $N = \tilde{n} + 3$.*

Applying our three reduction rules leads to a polynomial kernel with respect to vc.

Proposition 153. *With respect to parameter vc problem $1|prec, r_j, \bar{d}_j|\star$ has a polynomial kernel in time $\mathcal{O}(n^2 \cdot log(n) + (vc^2)^8 \cdot (vc^2 + log(vc^2))) = \mathcal{O}(n^{18})$. The resulting kernel has size $\mathcal{O}(vc^8)$ and $\mathcal{O}(vc^2)$ jobs.*

Proof. Let I be an instance of $1|prec, r_j, \bar{d}_j|\star$. First we build the $(vc + p_{\max})$-kernel $\kappa(I) = \langle prec, p_j, r_j, \bar{d}_j \rangle$ obtained in time $\mathcal{O}(vc \cdot n \cdot log(n))$ by Proposition 148. Let I' be the instance build from $\kappa(I)$ by the reduction rule 3.

We know that the new coefficients are not greater than $2^{4d^3} \cdot N^{d(d+2)}$. Substituting d by $3\tilde{n}$ and N by $\tilde{n} + 3$ yields vector $\mathbf{w}' = (p'_1, r'_1, \bar{d}'_1, \ldots, p'_{\tilde{n}}, r'_{\tilde{n}}, \bar{d}'_{\tilde{n}})$ with components not greater than $2^{\mathcal{O}(\tilde{n}^3)}$. Since $\tilde{n} = \mathcal{O}(vc^2)$, the size of each encoded component is $\mathcal{O}(vc^6)$. And since we have $\mathcal{O}(vc^2)$ jobs in I' we get the announced space bound for it.

Similarly the complexity $\mathcal{O}(d^8 \cdot (d + \log(N)))$ yields complexity $\mathcal{O}((vc^2)^8 \cdot (vc^2 + log(vc^2)))$. Use these values to define $I' = \langle prec, p'_j, r'_j, \bar{d}'_j \rangle$ instance of $1|prec, r_j, \bar{d}_j|\star$. We show that $\kappa(I)$ is feasible if and only if I' is feasible. Lemma 152 ensures that bound $N = \tilde{n} + 3$ of all the positive values of variables in each inequality was enough to make integer programs $BIP(\kappa(I))$ and $BIP(I')$ equivalent according to Theorem 150. Plus by Lemma 151 we know that $\kappa(I)$ (resp. I') is feasible if and only if $BIP(\kappa(I))$ (resp. $BIP(I')$) has a solution. So I' is indeed equivalent to $\kappa(I)$, which is itself equivalent to I according to Proposition 148. □

7.1.4 A Negative Result on Identical Parallel Processors with Processing Set Restrictions

While we can have polynomial-size kernels on problem $1|prec, r_j, \bar{d}_j|\star$ with parameter vc, in this subsection we show that this becomes unlikely in the restricted assignment setting for parallel processors.

Theorem 154. *$P|\mathcal{M}_j, r_j, \bar{d}_j|\star$ is $W[1]$-hard parameterized by vc.*

Proof. We reduce from k-INDEPENDENT SET. Let $G = (V, E)$ be a graph and k be a positive integer. k-INDEPENDENT SET is feasible if there exists a subset or vertices $S \subseteq V$ called independent set with $|S| = k$ such that any two nodes of S are not linked by an edge. Let $n = |V|$ and $V = v_1, \ldots, v_n$. We order the edges $(e_j)_{0 \leq j < |E|}$ from E arbitrarily. We define instance I of our scheduling problem assuming n machines named $1, \ldots, n$ and two types of jobs:

- k vertex selectors with processing time $|E|$ and time window $[0, |E|)$. They can be processed in any of the n machines. They correspond to the k vertices chosen as an independent set candidate.
- $|E|$ edge check jobs, one per edge $e_j = \{v_{i_1}, v_{i_2}\}$ in E. It has processing time 1, time window $[j, j + 1)$ and can only be processed in machines i_1 and i_2. It ensures that the corresponding edge does invalidate that the chosen set of vertices is independent.

Notice that in the comparability graph of the time windows of this instance, the vertex selectors define a complete subgraph, whereas the edge check jobs define an independent set. Hence in this graph the minimal vertex cover is the set of vertex selectors, so that parameter $vc = k$.

We show that G has an independent set of size k if and only if I is feasible. Given an independent set $S = \{u_1, \ldots, u_k\}$ of G of size k we assign the k vertex selectors to the corresponding machines. Since it is an independent set all edge check jobs have at least one available machine to be scheduled at.

Now suppose we have a feasible schedule of I. Let $i_1, \ldots, i_k$ be the machines at which the k vertex selectors were scheduled. We claim that $\{v_{i_1}, \ldots, v_{i_k}\}$ is an independent set of G. By contradiction suppose there are two indices ℓ, ℓ' and some edge e_j such that $e_j = \{v_{i_\ell}, v_{i_{\ell'}}\}$. Then the corresponding edge check job would clash with either vertex selector and make the schedule invalid. This leads to a contradiction, which confirms that $(v_{i_1}, \ldots, v_{i_k})$ is an independent set of G. □

This shows that $P|\mathcal{M}_j, r_j, \bar{d}_j|\star$ is unlikely to be FPT parameterized by vc and thus unlikely to have kernelizations with respect to vc. It is open whether $P|r_j, \bar{d}_j|\star$ is FPT parameterized by vc for any number of machines greater than one. We are also looking for kernelization algorithms with other structural parameters like pathwidth μ, slack σ and proper level q.

7.2 Twin-Width tww and Other Width Parameters

In this section we consider other width parameters on job time window interval graph $G_{[r_j,d_j)}$. Note that in interval graphs several parameters are equivalent to the pathwidth:

Claim 155. *[BM93] Let G be an interval graph. Then: $pathwidth(G) = treewidth(G) = maximum\_clique\_size(G) - 1 = chromatic\_number(G) - 1$.*

In what follows we consider twin-width tww, proper thinness $pthin$ and clique-width cw. Twin-width is a newly introduced width parameter in graph theory [BKTW21]. It is defined by means of contraction sequences. Given a graph $G = (V, E)$ edges can be either black or in red in a contraction sequence, and they are all black initially. When contracting two nodes u and v into a node z, we have an edge between z and every neighbor w of either u or v. Edge $\{w, z\}$ is black if w was a neighbor of both u and v and both $\{w, u\}$ and $\{w, v\}$ were black. Otherwise this edge is red. The contraction sequence ends when the graph is reduced down to a single node. Then twin-width is defined the following way:

Definition 156. *Given a graph G twin-width $tww(G)$ is defined as the minimum value d such that there exists a contraction sequence of G with red degree at most d throughout the whole sequence.*

Note that contracting two twins - i.e. two nodes with the same neighborhood - does not create any extra red edge. This is the intuition behind the name twin-width. For example in the complete graph K_n all pairs of nodes are twins. So contracting any pair of nodes adds no red edges and gives complete graph K_{n-1} as a result. This means that complete graph has twin-width 0.

Despite being bounded by most commonly studied width parameters like pathwidth, treewidth or clique-width, FO model checking on graphs is FPT parameterized by $tww(G)$ - assuming that a corresponding contraction sequence was previously computed [BKTW21]. Unfortunately computing twin-width and a corresponding contraction sequence proves to be difficult in the first place. Indeed Bergé et al. showed that it is NP-complete to decide whether a graph G has twin-width at most 4 [BBD22].

It is worth noting that interval graphs can have unbounded twin-width while proper interval graphs have twin-width at most 2 [BGK$^+$24]. This suggests that twin-width could help classify interval graphs in a similar way as proper level q. In fact we show that both parameters are closely related in the context of job time window interval graphs. In this work we define parameter tww as the twin-width job time window interval graph $G_{[r_j,d_j)}$. We show that tww is bounded by a function of q. As an intermediate we use the notion of proper k-thinness:

Definition 157 (Proper thinness)**.** *[BdE19]*

Let $G = (V, E)$ be a graph. Let k be a positive integer. G is called (proper) k-thin if there is an ordering $v_1, \ldots, v_n$ of V and a partition of V into k classes $V^1, \ldots, V^k$ such that for each triple $(r, s, t) \in [1, n]^3$ with $r < s < t$:

(*i*) *(k-thin) if* v_r, v_s *belong to the same class and* $\{v_t, v_r\} \in E$ *then* $\{v_t, v_s\} \in E$ *[MORC07],*

(*ii*) *(proper) if* v_s, v_t *belong to the same class and* $\{v_r, v_t\} \in E$ *then* $\{v_r, v_s\} \in E$*.*

The proper thinness $pthin(G)$ *of a graph* G *is defined as the smallest integer* k *such that* G *is proper* k*-thin.*

Proper intervals graphs are proper 1-thin while the proper thinness of interval graphs can be unbounded. (Proper) k-thin graphs were recently shown to have twin-width at most $9k$ [BHJ24]. We show that q-proper graphs have bounded proper thinness.

Result 158. *q-proper graphs are proper $(q+1)$-thin.*

Proof. q-proper graphs are $(q+1)$-nested interval graphs - i.e. there are no chains of $(q+2)$ intervals nested in each other [KOŠ19]. Then by Proposition 11 in [BdE19] we conclude that q-proper graphs are proper $(q+1)$-thin. □

This infers that q-proper graphs have bounded twin-width.

Corollary 159. $tww \leq 9(q+1)$.

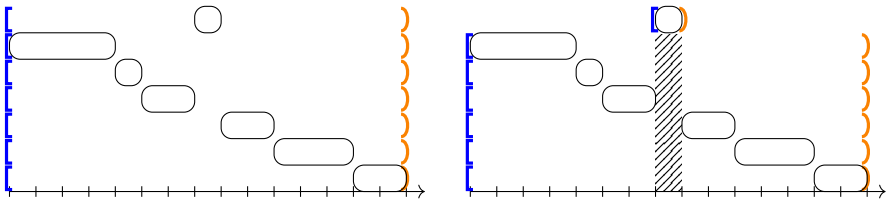
Thus this would set both tww and $pthin = pthin(G_{[r_j,d_j)})$ as legitimate weaker alternatives whenever a problem is FPT parameterized by q. Unfortunately the following result suggests that both parameters are too weak to yield fixed-parameter-tractable algorithms on our studied scheduling problems.

Result 160. *$1|r_j, \bar{d}_j|\star$ and $P|tree, p_j = 1, r_j, \bar{d}_j|\star$ are para-NP-hard parameterized by tww, pthin and cw (with $tww = 0, pthin = 1, cw = 2$).*

Proof. For $1|r_j, \bar{d}_j|\star$ a straightforward reduction from PARTITION yields a complete graph as the job time window interval graph $G_{[r_j,d_j)}$, which has twin-width 0, proper thinness 1 and clique-width 2. For $P|tree, p_j = 1, r_j, \bar{d}_j|\star$ we reduce from NP-hard problem $P|intree, p_j = 1, r_j|C_{max} < D$ [BGJ77] and set $\bar{d}_j = D$ for all jobs j. Then all job time windows overlap at time $D-1$, so $G_{[r_j,d_j)}$ is also a complete graph. □

A possible interpretation is given by Figure 7.4. While both proposed instances have $G_{[r_j,d_j)}$ be the complete graph, the left instance is easy to solve while the right instance simulates a PARTITION problem. So a parameter which only considers whether intervals overlap does not completely discriminate between easy and hard scheduling instances. For example pathwidth μ is too pessimistic while twin-width tww is too optimistic. Like proper level q a more accurate parameter based on $G_{[r_j,d_j)}$ must look deeper into the nature of interval overlaps.

Now it is worth noting that the twin-width of a directed acyclic graph is at most linear in its width w and a corresponding contraction sequence can



$tww = 0$, $q = 0$, $\mu = n$ $tww = 0$, $q = n - 1$, $\mu = n$

Figure 7.4: Two instances of $1|r_j, \bar{d}_j|\star$ with the same twin-width tww and pathwidth μ but different q-proper levels q.

be computed in time $\mathcal{O}(w \cdot n^2)$ [BH21]. So $tww(prec)$ could considered as a weaker parameter variant of w the width of precedence graph $prec$. One could consider replacing w with $tww(prec)$ in existing FPT results and see if such problems remain FPT. While most results with parameter w alone are already negative - $P|prec, p_j = 1|C_{max} < D$ is $XNLP$-hard [BGNS22] while $P2|prec, p_j \in \{1, 2\}|C_{max} < D$ and $P3|prec, p_j = 1, size_j \in \{1, 2\}|C_{max} < D$ are $W[2]$-hard [vBBB+16] - RCPSP was shown to be FPT when w is combined with λ the maximum allowed difference between the starting time of each job and their earliest possible starting time according to precedence relations. So one could consider whether RCPSP is FPT parameterized by $tww(prec) + \lambda$.

7.3 Average Parameters

In this section we introduce a notion of average parameters. All the mainly studied parameters - $\ell_{max}, \mu, \sigma, q$ - are defined as the maximum within a set of values. While this is enough to get an upper bound on the complexity of the associated algorithms, a single outlier value can increase the value of these parameters dramatically while having little impact on the complexity of the algorithms. In order to describe the difficulty of every instance more accurately, we propose a joint parameter which takes into account the whole set of values from which the parameter is computed.

Definition 161. *Let I be an instance of a parameterized problem $(\mathcal{P}, k)$. Suppose there is a subset $\mathcal{N}_I$ of elements such that $|\mathcal{N}_I| = poly(|I|)$, and a function $f : \mathcal{N}_I \mapsto \mathbb{N}_0$ such that $k = max_{j \in \mathcal{N}_I} f(j)$. Then we define:*

$$k_{avg} = log(\frac{1}{|\mathcal{N}_I|} \sum_{j \in \mathcal{N}_I} 2^{f(j)}).$$

For example consider the FPT algorithm of problem $1|prec, r_j, \bar{d}_j|C_{max}$ parameterized by q given in Section 6.3. We set $\mathcal{N}_I$ as the set of summits in instance I. Then $|\mathcal{N}_I| \leq n$. And if function f returns the size of the corresponding proper set then we have $q = max_{j \in \mathcal{N}_I} f(j)$. In the proposed dynamic program-

ming algorithm the number of states is equal to $2+\sum_{j\in\mathcal{N}_I} 2^{f(j)}$. While it can be bounded by $\mathcal{O}(2^q \cdot |\mathcal{N}_I|)$ parameter q_{avg} gives a better estimate $\Theta(2^{q_{avg}} \cdot |\mathcal{N}_I|)$.

Note that, as expected, the average parameter is always smaller than the original one. However choosing this notion of "average parameter" guarantees the following lower bound on k_{avg}:

Lemma 162. *$k - log(|\mathcal{N}_I|) \leq k_{avg} \leq k$.*

Proof. The lower bound is obtained by only keeping one j with maximum value $f(j)$ in the sum defining k_{avg}. The upper bound is obtained by bounding all elements in the sum by 2^k. □

This lower bound allows us to get FPT results with the average parameter when a FPT algorithm has a single-exponential dependency on the original parameter.

Proposition 163. *If a problem $\mathcal{P}$ can be solved in time $a^k \cdot poly(|I|)$ with $a \in \mathbb{R}^{+*}$ then $\mathcal{P}$ is FPT parameterized by k_{avg}.*

Proof. By Lemma 162: $k - log(|\mathcal{N}_I|) \leq k_{avg}$. So:

$$\begin{aligned} a^k &\leq a^{k_{avg}+log(|\mathcal{N}_I|)} \\ &\leq a^{k_{avg}} \cdot |\mathcal{N}_I|^{log(a)} \\ &\leq a^{k_{avg}} \cdot poly(|I|). \end{aligned}$$

Thus $\mathcal{P}$ can be solved in time $[a^{k_{avg}} \cdot poly(|I|)] \cdot poly(|I|) = a^{k_{avg}} \cdot poly(|I|)$. □

In the side of theory it shows that problem $\mathcal{P}$ remains FPT with a slightly smaller parameter. In practice given some real-world instance computing k_{avg} instead of k gives a better guess at whether the FPT algorithm can be efficient. We propose examples of applications with three of the mainly studied parameters in this work: slack σ, pathwidth μ and proper level q.

With slack σ we simply set $\mathcal{N}_I$ as the set of jobs and function f mapping every job j to value $\bar{d}_j - r_j - p_j$. Then we have $|\mathcal{N}_I| = n$ and $\sigma = max_{j\in\mathcal{N}_I} f(j)$. In [BdWH21] Baart et al. showed that $1|r_j, s_{jk}, reject, \bar{d}_j| \sum_{j\notin R}(w_j T_j - v_j)$ can be solved in time $\mathcal{O}(\sigma^3 \cdot 4^\sigma \cdot n^2)$. So Proposition 163 implies the following result:

Corollary 164. *$1|r_j, s_{jk}, reject, \bar{d}_j| \sum_{j\notin R}(w_j T_j - v_j)$ is FPT parameterized by σ_{avg}.*

With pathwidth μ we set $\mathcal{N}_I$ as the set of intervals delimited by the non-decreasing sequence of release date and deadline values. Then $|\mathcal{N}_I| \leq 2n$. And if function f gives the number of time windows intersecting each of these intervals, we indeed have $\mu = max_{j\in\mathcal{N}_I} f(j)$. Since Munier showed that problem $P|prec, p_j = 1, r_j, \bar{d}_j|L_{max}$ has a FPT algorithm in time $\mathcal{O}(16^\mu \cdot n^4)$ [MK21], Proposition 163 implies the following result:

Corollary 165. *$P|prec, p_j = 1, r_j, \bar{d}_j|L_{max}$ is FPT parameterized by μ_{avg}.*

Finally with proper level q we set $\mathcal{N}_I$ as the set of summits. Then $|\mathcal{N}_I| \leq n$. And if function f returns the size of the corresponding proper set then we do have $q = max_{j \in \mathcal{N}_I} f(j)$. In [MHMK24a] we showed that $1|prec, r_j|L_{max}$ has a FPT algorithm with a single-exponential dependency in q. So Proposition 163 implies the following result:

Corollary 166. $1|prec, r_j|L_{max}$ *is* FPT *parameterized by* q_{avg}.

7.4 Summary & Concluding Remarks

Parameter	Section	Problem / Setting	Result
μ	7.1	$1\|r_j, \bar{d}_j\|\star$	No polynomial kernel (unless $NP \subseteq coNP/poly$).
vc		$1\|prec, r_j, \bar{d}_j\|\star$	Polynomial kernel with both $(vc + p_{max})$ and vc.
		$P\|\mathcal{M}_j, r_j, \bar{d}_j\|\star$	$W[1]$-hard.
tww, $pthin, cw$	7.2	$1\|r_j, \bar{d}_j\|\star$ $P\|tree, p_j = 1, r_j, \bar{d}_j\|\star$	para-NP-hard.
μ_{avg}, σ_{avg}, q_{avg}	7.3	-	Several inferred FPT results.

Figure 7.5: Summary of the results obtained in this chapter.

In this chapter we presented the remaining results obtained in this thesis. First we discussed kernelization algorithms and showed that even on a single machine a polynomial kernel is unlikely with pathwidth μ as a parameter. In response we introduced a stronger parameter called the vertex cover vc, for which we give a polynomial kernel. Next we considered other structural parameters based on interval graph $G_{[r_j,d_j)}$. We argued that most of them are not best suited to deal with scheduling problems featuring job time windows. Finally we introduced a notion of average parameter which can preserve FPT results with a single-exponential time dependency on the original parameter.

Further research would primarily look for new kernelization algorithms with other structural parameters like pathwidth μ or proper level q. The main challenge remains to find suitable reduction rules which would allow us to delete a significant number of jobs from the original scheduling instance. Despite the recent progress with vertex cover vc, we expect the wanted reduction rules to be more intricate and lead to kernels of exponential size at best. Still pursuing new parameter ideas based on interval graph $G_{[r_j,d_j)}$ could certainly foster this search for reduction rules. Such parameters would take inspiration from proper level q and use information beyond the base graph structure in their definition.

Chapter 8

Conclusion

8.1 Summary of our Results

In this work we explored the parameterized complexity of RCPSP and its subproblems enhanced with job time windows and/or precedence delays. We summarize the obtained results in Figure 8.1.

In Chapter 3 we considered bounding the values of the precedence delays which can be set on the edges of the precedence graph given in the input. This corresponds to parameter ℓ_{max}, with which we proved a number of (mainly negative) results. We started with the exact and maximum delay types on single machine scheduling with chains of unit jobs. In the presence of job time windows we showed that the problem was para-NP-hard with either delay type, even with a single delay value. Without time windows we noted that the problem was still $W[1]$-hard with exact delays while it could be solved in linear time with arbitrary maximum delays.

Next with minimum delays we gave several more negative results. On single machine scheduling with unit jobs and general precedence we proved $XNLP$-hardness with respect to ℓ_{max} and precedence graph width w combined. Then we showed that the problem was still $W[2]$-hard with chains of precedence if extra delays of length zero were allowed. With parameter ℓ_{max} alone this was strengthened to para-NP-hardness on parallel machines with job time windows. This last result was part of our IPEC 2022 publication [MHMK22a]. In short our results show that bounding the maximum delay value ℓ_{max} is not enough to obtain FPT algorithms on most scheduling subproblems of RCPSP enhanced with precedence delays.

In Chapter 4 we examined pathwidth μ which had been a successful parameter in the recent literature. We confirmed this by finding a new FPT algorithm on single machine scheduling with job time windows and precedence constraints. This result was presented at MAPSP 2022 [HMMK22]. However when precedence delays were added, the problem became para-NP-hard no matter the delay type even with unit jobs and chains of precedence. This was

Parameter	Sect.	Problem / Setting	Result
ℓ_{max}	3.2	$1\|chains(\ell), p_j = 1, r_j, \bar{d}_j\|\star$	para-NP-hard with exact or maximum delays.
		$1\|chains(\ell), p_j = 1\|C_{max} < D$	$W[1]$-hard with exact delays.
		$1\|chains(\ell_{i,j}), p_j = 1\|C_{max} < D$	$\mathcal{O}(n)$ with maximum delays.
		$1\|chains(0, \ell), p_j = 1, r_j\|C_{max} < D$ $1\|chains(0, \ell), p_j = 1, \bar{d}_j\|\star$	NP-hard with maximum delays.
	3.3	$1\|prec(\ell), p_j = 1\|C_{max} < D$	$XNLP$-hard($\ell_{max} + w$) with minimum delays.
		$1\|chains(0, \ell), p_j = 1\|C_{max} < D$	$W[2]$-hard(ℓ_{max}+#chains) with minimum delays.
		$P\|chains(0, \ell), p_j = 1, r_j, d_j\|\star$	para-NP-hard($\ell = 1$) with minimum delays.
μ	4.2	$1\|prec, r_j, \bar{d}_j\|C_{max}$	FPT in time $\mathcal{O}(\mu^2 \cdot 4^\mu \cdot n + n^2)$.
	4.3	$1\|chains(\ell_{ij}), p_j = 1, r_j, \bar{d}_j\|\star$	para-NP-hard for all three delay types.
$\mu + \ell_{max}$	4.4	$P\|prec(\ell_{i,j}), p_j = 1, r_j, \bar{d}_j\|\star$	FPT with minimum delays.
σ	5.2	Single machine	$\mu \leq 2\sigma$
		Pm	$\mu \leq 2(\sigma + 1) \cdot m - 1$
		-	Several inferred FPT results.
	5.3	$1\|(1, \ell, 1), r_j, \bar{d}_j\|\star$	para-NP-hard for all three delay types.
$\sigma + \ell_{max}$	5.4	$1\|prec(\ell_{i,j}), r_j, \bar{d}_j\|\star$	FPT with all three delay types combined.
q	6.2	-	$q \leq \mu$
		$P2\|r_j, \bar{d}_j\|\star$ $1\|(1, \ell, 1), r_j, \bar{d}_j\|\star$	Inferred para-NP-hardness results.
	6.3	$1\|prec, r_j, \bar{d}_j\|C_{max}$	FPT in time $\mathcal{O}(max(1, q^2 \cdot 4^q) \cdot N + n^2)$.
	6.4	$P\|prec, p_j = 1, r_j, \bar{d}_j\|\star$	para-NP-hard($q + D$).
			$W[2]$-hard($q + w$).
μ	7.1	$1\|r_j, \bar{d}_j\|\star$	No polynomial kernel (unless $NP \subseteq coNP/poly$).
vc		$1\|prec, r_j, \bar{d}_j\|\star$	Polynomial kernel with both ($vc + p_{max}$) and vc.
		$P\|\mathcal{M}_j, r_j, \bar{d}_j\|\star$	$W[1]$-hard.
tww, $pthin, cw$	7.2	$1\|r_j, \bar{d}_j\|\star$ $P\|tree, p_j = 1, r_j, \bar{d}_j\|\star$	para-NP-hard.
μ_{avg}, σ_{avg}, q_{avg}	7.3	-	Several inferred FPT results.

Figure 8.1: Summary of the results obtained in this thesis.

first presented at ROADEF 2022 and it established that pathwidth μ alone as a parameter was not suited to deal with precedence delays. However when combined with maximum delay value ℓ_{max} we proposed a FPT algorithm on parallel machines with general precedence - as long as we have unit processing times. Both the negative result and the FPT algorithm were included in our IPEC 2022 publication [MHMK22a]. This motivated the integration of job time windows to problems featuring precedence delays in order to broaden available parameter choices.

Chapter 5 was dedicated to slack σ, another parameter based on job time windows. On a single machine or a fixed number of identical parallel processors we showed that σ was stronger than pathwidth μ. This inferred several FPT results with respect to σ from the recent advances made with pathwidth μ. In the rest of the chapter we focused on single machine scheduling with both time windows and precedence delays. We drew similar conclusions to pathwidth μ. With slack σ alone we showed that single machine scheduling was para-NP-hard for all three precedence delay types even when restricted to coupled unit tasks with the same intermediate delay value. Then when combined with ℓ_{max} the problem became FPT even with general precedence and arbitrary processing times - whereas this is open with respect to parameter $\mu + \ell_{max}$. The contents of this chapter are intended to be submitted to Discrete Applied Mathematics in the near future.

In Chapter 6 we proposed a new parameter based on job time windows. We called it the proper level q and we showed that it was weaker than pathwidth μ. This implied that most of our studied problems were para-NP-hard with respect to q. Nevertheless we managed to obtain a FPT algorithm on single machine scheduling with job time windows and precedence constraints. This led to our publication in ISCO 2024 [MHMK24a]. We closed this chapter with a couple additional negative results with respect to q on parallel machine scheduling with unit jobs - a problem known to be FPT with respect to μ [MK21].

Finally in Chapter 7 we presented several other parameterized results obtained during this thesis. We began by discussing the possibility of kernelization algorithms for scheduling problems with job time windows. We showed that a polynomial kernel with respect to pathwidth μ was unlikely to exist even on the single machine case. In response we proposed vertex cover vc, a stronger parameter with which we managed to secure a polynomial kernel - combined with maximum processing time p_{max} to begin with, then alone. When upgrading to identical parallel machines subject to the restricted assignment setting we showed that the problem became $W[1]$-hard parameterized by vc. This made it unlikely to find any kernelization algorithm on this problem with respect to vc. These kernel results are subject to a submission to IPEC 2024.

Next we considered other notorious structural parameters in graph theory like treewidth or twin-width and gave input on their potential uses in scheduling problems with job time windows. We argued that most of the parameters defined on our time window interval graph failed to distinguish between easy and hard scheduling instances. As a consequence they were either too 'optimistic' like the twin-width or too pessimistic like pathwidth μ. Lastly we proposed a notion

of average parameter which could capture better the effective time and space complexity of many existing FPT algorithms. We showed that when the time complexity of such an algorithm only had a single-exponential dependency on a given parameter, the FPT result could be transferred to our corresponding average parameter.

8.2 Further Work

In this final section we give several ideas to utilize and extend the work done in this thesis.

First one can take inspiration from the MIMO framework developed by Knop et al. in the high-multiplicity setting [KKL$^+$19] and set more metatheorems for scheduling problems. For each parameter the goal would be to characterize which job properties are pivotal in getting a FPT result. Recall that any scheduling problem expressible in the MIMO framework has a FPT algorithm with respect to the number of job types d either combined with the number of machines m or the maximum processing times m. Coincidentally with respect to pathwidth μ we have a FPT algorithm either on a single machine with arbitrary processing times or on multiple machines with unit jobs. In contrast it is believed that adding precedence constraints and renewable resources with non-unit amount requirements has no impact on any FPT result involving parameter μ. As such we conjecture that RCPSP with job time windows and unit jobs is FPT parameterized by μ.

In the long run the best case scenario would be to find an analogue to Courcelle's theorem on scheduling problems, for example with respect to pathwidth μ on problems featuring job time windows. This could prove tricky, knowing that scheduling problems can be notoriously heterogeneous in their featured machine environment and job properties. Indeed this prevents us from reusing a lot of the work done in parameterized graph theory in a straightforward way, notably when attempting to define a suitable logic in which scheduling problems can be expressed.

Another theoretical matter concerns kernelization algorithms on scheduling problems with structural parameters, as we did with vertex cover vc in Chapter 7. For instance numerous FPT algorithms were found with pathwidth μ, slack σ and proper level q in the last five years, whereas to our knowledge no kernelization algorithm has been designed with respect to any of these parameters. The main challenge remains to find suitable reduction rules which would allow us to delete a significant number of jobs from the original scheduling instance. Even though our recent progress with vertex cover vc is encouraging, we expect the wanted reduction rules to be more intricate and lead to kernels of exponential size at best.

On another note we observe that the parameterized results in scheduling have been mostly theoretical. While the first objective is to identify the FPT problems without much attention on the time complexity, some of the proposed algorithms already have a chance to compete with the state of the art. In

this thesis our FPT algorithms with respect to pathwidth μ or proper level q could be considered right away in some settings, as they have a rather low time dependency on both the input size and the parameter - respectively quadratic and single-exponential. In the literature Tarhan et al. showed that an existing branch-and-bound method could be adapted to a FPT algorithm with respect to parameter $\mu + p_{max}$ on parallel-machine scheduling with precedence relations and job time windows [TCH+23]. They then initiated some computational experiments on random instances. One catch is that the values of parameters μ and q tend to be high on general instances. For example with job time windows drawn uniformly both parameters have an expected value at least proportional to the input size, which is less than ideal in a parameterized setting. Nevertheless one can identify specific settings in which our parameters could perform well. For instance pathwidth μ often has a low value when the time window lengths are small compared to makespan threshold D, while proper level q is small typically when all time windows have nearly the same length.

Similarly we note that parameterized studies on scheduling problems have aimed at exact resolution significantly more often than approximations. In the case where FPT algorithms are either unlikely, difficult to find or simply too time consuming, such parameterized approximations would provide a compromise between optimality and performance. For instance while Baart found a FPT algorithm with respect to slack σ on a single machine problem with sequence-dependent setup times and rejection, they also proposed a parameterized FPTAS with respect to weaker parameter μ [BdWH21]. This area of research is largely open on scheduling problems and looks promising.

Lastly with the recent rapid development of parameterized complexity in scheduling, it has become increasingly difficult to stay up to date and track all progress in the field. As such, newcomers would greatly benefit from a publicly available archive gathering all parameterized complexity results in scheduling. With this goal in mind, we plan to contribute to the Scheduling Zoo [BKD10] in the near future. The project consists of a searchable bibliography in the area of scheduling problems. It is publicly available via a webpage hosted by LIP6. It notoriously features reduction rules, which defines a partial order on the problems and makes it able to find results for generalizations or particular cases. Right now the Scheduling Zoo only features a limited number of isolated parameterized results. We intend to update the bibliography with all recent parameterized complexity results. In collaboration with C. Dürr we also plan to update the reduction rule system in order to make it capable of inferring results from parameterized problems defined on different parameters.

Appendix A

Extra

A.1 Associated Publications

In this section we list the publications produced as part of this thesis.

International Conferences with Program committees and Proceedings

- [**MHMK24a**] **ISCO 2024**
  - Presented in Tenerife, Canary Islands, Spain in May 2024.
  - Title: *A new structural parameter on single machine scheduling with release dates and deadlines*
  - Co-authors: Claire Hanen, Alix Munier-Kordon (LIP6)
  - These results correspond to Section 6.3.
- [**MHMK22a**] **IPEC 2022**
  - Presented in Potsdam, Germany in September 2022.
  - Title: *Parameterized complexity of a parallel machine scheduling problem*
  - Co-authors: Claire Hanen, Alix Munier-Kordon (LIP6)
  - These results correspond to Subsection 3.3.3 and Section 4.4.

Other International Conferences

- [**HMMK22**] **MAPSP 2022**
  - Presented in Oropa (Biella), Italy in June 2022.
  - Title: *Parameterized complexity of a single machine scheduling problem with precedence, release dates and deadlines*
  - Co-authors: Claire Hanen, Alix Munier-Kordon (LIP6)
  - These results correspond to Section 4.2.

National Conferences

- [**MHMK24b**] **ROADEF 2024**
  - Presented in Amiens, France in March 2024.
  - Title: *Parameterized complexity: a two-dimensional approach to study scheduling problems*
  - Co-authors: Claire Hanen, Alix Munier-Kordon (LIP6)
- [**Mal22**] **ROADEF 2022**
  - Presented in Lyon, France in February 2022.
  - Title: *Parameterized complexity of a single machine scheduling problem*
  - These results correspond to Section 4.3.

Pending Submissions

- **INFORMS Journal on Computing** Reviewing in progress
  - Title: *Kernelization algorithms on single machine scheduling with time windows*
  - Co-authors: Claire Hanen, Alix Munier-Kordon (LIP6)
  - These results correspond to Section 7.1.

In Preparation

- **Discrete Applied Mathematics** Writing in progress
  - Title: *Scheduling with precedence constraints, time windows and bounded proper level*
  - Co-authors: Claire Hanen, Alix Munier-Kordon (LIP6)
  - These results correspond to Chapter 6.
- **Theoretical Computer Science** Writing in progress
  - Title: *Parameterized analysis of single machine scheduling with time windows and precedence delays*
  - Co-authors: Claire Hanen, Alix Munier-Kordon (LIP6)
  - These results correspond to Section 5.3.
- **Journal of Combinatorial Optimization** Writing in progress
  - Title: *Single machine scheduling with precedence delays, time windows and bounded maximum delay*
  - Co-authors: Claire Hanen, Alix Munier-Kordon (LIP6)
  - These results correspond to Subsection 3.3.1, Subsection 3.3.2 and Section 5.4.

A.2 Notations and Definitions

In this section we give a non-exhaustive list of the notations used throughout this work. We also give concise definitions of the complexity classes and the parameters mentioned in this thesis.

A.2.1 Scheduling

General

$\mathcal{P}$	Generic notation for the currently studied scheduling problem.
I	Generic notation for the currently studied scheduling instance.
$\mathcal{J}$	Generic notation for the set of jobs in the current instance.
τ	Generic notation for a schedule $\tau : \mathcal{J} \mapsto \mathbb{N}_0$.
n	Generic notation for the number of jobs.
m	Generic notation for the number of machines.
p_j	Processing time of job j.
C_j	Completion time of job j in a given schedule τ. Denoted by $C_j^{(\tau)}$ if schedule τ must be specified. *Formula:* $\tau(j) + p_j$.
U_j	Throughput associated to job j. $U_j = 1$ if j is completed by its due date (i.e. $C_j \leq d_j$), otherwise $U_j = 0$. Denoted by $U_j^{(\tau)}$ if schedule τ must be specified. *Setting:* due dates d_j.
T_j	Tardiness of job j. Denoted by $T_j^{(\tau)}$ if schedule τ must be specified. *Setting:* due dates d_j. *Formula:* $max(0, C_j - d_j)$.
F_j	Flow time of job j (i.e. the difference between its completion time and its release date). Denoted by $F_j^{(\tau)}$ if schedule τ must be specified. *Setting:* release dates r_j. *Formula:* $(C_j - r_j)$.
$G_{[r_j,d_j)}$	Interval graph of the job time windows. The vertices are the intervals and there is an edge between two intervals if they overlap. *Setting:* job time windows $[r_j, \bar{d}_j)$.
active	Property of a schedule τ. Jobs are scheduled without any unnecessary downtime anywhere (i.e. as early as possible).

Scheduling problems are denoted as an extended version of the three field notation $\alpha|\beta|\gamma$ proposed by Graham et al. in [GLLK79]. Field α gives the machine environment, field β describes the job properties and field γ is the objective function to be minimized among valid schedules.

Field α: Machine Environment

1	Single machine.
P	Identical parallel machines.
Pm	Identical parallel machines with a fixed number of machines.
$\bar{P}$	Identical parallel machines with an unlimited number of machines.
Q	Uniform parallel machines. Each machine i has a speed value s_i. Job j is processed by machine i in time $\frac{p_j}{s_i}$.
R	Unrelated parallel machines. Jobs can have machine dependent processing times. $p_j^{(i)}$ denotes the processing time of job j on machine i.

Field β: Job Properties

$\mathcal{M}_j$	Restricted assignment setting. This corresponds to a parallel machine setting where each job j is assigned a set $\mathcal{M}_j$ of machines on which it is allowed to be scheduled. *Setting:* machine environment P, Q or R.
prec	General precedence graph on the set of jobs. If $(i,j) \in prec$ then job j cannot start before job i is completed.
chains	The precedence graph is a collection of disjoint chains.
intree	The precedence graph is a tree with maximum outdegree 1.
outtree	The precedence graph is a tree with maximum indegree 1.
tree	The precedence graph is either an intree or an outtree.
opposing forest	The precedence graph is a collection of intrees and outtrees.
$prec(\ell_{i,j})$	General precedence graph with a delay value on each relation. If $i \xrightarrow{\geq \ell_{i,j}} j$ (resp. $i \xrightarrow{=\ell_{i,j}} j$, $i \xrightarrow{\leq \ell_{i,j}} j$) then job j must start at least (resp. exactly, at most) $\ell_{i,j}$ time units after job i is completed.
$prec(\ell)$	All precedence delays in *prec* have the same value $\ell \in \mathbb{N}$.
$prec(0,\ell)$	All precedence delays in *prec* have value either 0 or ℓ.
(a_j, ℓ_j, b_j)	Coupled task scheduling. Jobs are grouped into pairs. In the j^{th} coupled task the first (resp. second) job has processing time a_j (resp. b_j) and there is a precedence delay ℓ_j between both jobs.

(a_j, ℓ, b_j)	All coupled tasks have the same precedence delay value ℓ.
(a, ℓ, b)	All coupled tasks have identical precedence delay value ℓ and task processing times a, b.
(p, ℓ, p)	All coupled task delays have value ℓ and all tasks have processing time p.
$(1, \ell_j, 1)$	Coupled task scheduling with unit-time jobs.
G_c	Comparability graph. The nodes are the coupled tasks of the instance. There is an edge between two coupled tasks if they are allowed to interleave each other. *Setting:* Coupled task scheduling.
$HM(m, n)$	High-multiplicity setting. Instead of describing the properties of the n jobs (resp. m machines) individually in the input, d job types (resp. κ machine types) are characterized and the number of elements associated to each type is given.
$HM(m)$	High-multiplicity setting but only on the set of machines.
$HM(n)$	High-multiplicity setting but only on the set of jobs.
$pmtn$	Jobs can be interrupted and resumed at a later date.
$p_j = 1$	All jobs have a unit-time processing time.
$p_j = p$	All jobs have the same processing time $p \in \mathbb{N}$.
$p_j \in \{p, q\}$	All jobs have processing time either p or q.
$p_j^{(i)} \in \{p_j, \infty\}$	Alternative notation of restricted assignment setting $\mathcal{M}_j$.
r_j	Release date. Job j only becomes available at this date.
d_j	Due date. *Setting:* due date dependent objectives like L_{max} and $\sum T_j$.
$\bar{d}_j$	Deadline. Job j must be completed by this date.
r_C	Chain release date. The first job of the chain only becomes available at this date. *Setting: chains*
$\bar{d}_C$	Chain deadline. The last job of the chain must be completed by this date. *Setting: chains*
$size_j$	Job size. Job j requires $size_j$ machines/resource amount to be processed. *Setting:* machine environment P or a problem with a renewable resource like RCPSP restricted to a single resource.
c_{ij}	Communication delay. If $(i, j) \in prec$ and i, j are scheduled on different machines, then job j must start at least c_{ij} time units after job i is completed.

$s_{i,j}$	Sequence-dependent setup time. If job j is scheduled right after job i on some machine, then j can only be started at least $s_{i,j}$ time units after i is completed.
$reject$	Jobs can be rejected - i.e. removed from the schedule - usually with some penalty on the objective.

Field γ: Objective Function

$\star$	Decision problem.
C_{max}	Makespan (i.e. the maximum completion time). *Formula:* $min_{\tau \text{valid schedule}} [max_{j \in \mathcal{J}} C_j^{(\tau)}]$.
$C_{max} < D$	Decision problem associated to the makespan. Its threshold D is given in the input.
C_{min}	Minimum completion time. *Formula:* $max_{\tau \text{valid schedule}} [min_{j \in \mathcal{J}} C_j^{(\tau)}]$.
C_{envy}	Envy (i.e. the difference between the maximum and minimum completion times). *Formula:* $min_{\tau \text{valid schedule}} [(max_{j \in \mathcal{J}} C_j^{(\tau)}) - (min_{j \in \mathcal{J}} C_j^{(\tau)})]$.
L_{max}	Maximum lateness. *Setting:* due dates d_j. *Formula:* $min_{\tau \text{valid schedule}} [max_{j \in \mathcal{J}} (C_j^{(\tau)} - d_j)]$.
$\sum U_j$	Throughput (i.e. the number of jobs on time). *Setting:* due dates d_j. *Formula:* $max_{\tau \text{valid schedule}} [\sum_{j \in \mathcal{J}} U_j^{(\tau)}]$.
$\sum C_j$	Total completion time. *Formula:* $min_{\tau \text{valid schedule}} [\sum_{j \in \mathcal{J}} C_j^{(\tau)}]$.
$\sum w_j C_j$	Weighted total completion time. *Formula:* $min_{\tau \text{valid schedule}} [\sum_{j \in \mathcal{J}} (w_j C_j^{(\tau)})]$.
$\sum w_j T_j$	Weighted total tardiness. *Setting:* due dates d_j. *Formula:* $min_{\tau \text{valid schedule}} [\sum_{j \in \mathcal{J}} (w_j T_j^{(\tau)})]$.
$\sum w_j F_j$	Weighted total flow time. *Setting:* release dates r_j. *Formula:* $min_{\tau \text{valid schedule}} [\sum_{j \in \mathcal{J}} (w_j F_j^{(\tau)})]$.
ℓ_2	ℓ_2-norm of the load vector. The load $\mathcal{L}_i$ of machine i is the total amount of time this machine spends processing jobs. *Setting:* machine environment P, Q or R. *Formula:* $min_{\tau \text{valid schedule}} \sqrt{\sum_{i \text{ machine}} (\mathcal{L}_i^{(\tau)})^2}$.

ℓ_p^C	ℓ_p-norm of the completion time vector ($p \in \mathbb{N}$). *Formula:* $min_{\tau \text{valid schedule}} [(\sum_{j \in \mathcal{J}} (C_j^{(\tau)})^p)^{\frac{1}{p}}]$.

A.2.2 Parameterized Complexity

$poly(\|I\|)$	Some polynomial in the instance size.
FPT	Parameterized complexity class which stands for *fixed-parameter tractable.* A problem is FPT parameterized by k if it can be solved deterministically in time $f(k) \cdot poly(\|I\|)$ for some computable function f.
$FPT(k)$	Shorthand notation to say that a problem is fixed-parameter tractable parameterized by k.
XP	Parameterized complexity class. A problem is XP parameterized by k if it can be solved deterministically in time $f(k) \cdot \|I\|^{g(k)}$ for some computable functions f, g.
$XP(k)$	Shorthand notation to say that a problem is XP parameterized by k.
para-NP	Parameterized complexity class. A problem is para-NP parameterized by k if it can be solved nondeterministically in time $f(k) \cdot poly(\|I\|)$ for some computable function f. A problem is para-NP-hard with respect to k if and only if it is NP-hard for some fixed value of k. If so, then unless $P = NP$ this problem is not FPT parameterized by k.
para-NP-hard($k = a$)	Shorthand notation to say that a problem is NP-hard when k is fixed with a value a.
$W[t]$	t^{th} level of the W-hierarchy ($t \in \mathbb{N}$). See Section 2.2.
$W[t]$-hard(k)	Shorthand notation to say that a problem is $W[t]$-hard parameterized by k.
$XNLP$	Parameterized complexity class. A problem is $XNLP$ parameterized by k if it can be solved nondeterministically in time $f(k) \cdot poly(\|I\|)$ and space $g(k) \cdot log(\|I\|)$ for some computable functions f, g.

A.2.3 Parameters

m	Number of machines. *Setting:* machine environment P, Q or R.
D	Makespan threshold. It can also be interpreted as a global deadline on the instance.
p_{max}	Maximum processing time.
ℓ_{max}	Maximum precedence delay value.

s_{max}	Maximum machine speed. *Setting:* machine environment Q.
s_{max}/s_{min}	Ratio between the maximum and minimum machine speeds. *Setting:* machine environment Q.
w_{max}	Maximum job weight. *Setting:* Any weighted objective function like $\sum w_j C_j$.
$\#p_j$	Number of distinct processing times.
$\#r_j$	Number of distinct release dates.
$\#\bar{d}_j$	Number of distinct deadlines.
$\#d_j$	Number of distinct due dates.
$\#\ell_{i,j}$	Number of distinct precedence delay values.
$\#w_j$	Number of distinct job weights. *Setting:* Any weighted objective function like $\sum w_j C_j$.
#rejected	Number of rejected jobs. *Setting: reject*
#selected	Number of selected jobs. *Setting: reject*
d	Number of job types.
κ	Number of machine types.
ρ	Rank of the matrix $(p_j^{(i)})_{i,j}$ defined by the processing time values on unrelated parallel machines. *Setting:* machine environment R.
w	Width of precedence graph *prec* (i.e. the maximum size of any antichain in *prec*). *Setting: prec* *Formula:* $max_{S \subseteq \mathcal{A}} \lvert S \rvert$ where $\mathcal{A} = \{S \subseteq \mathcal{J} \mid \forall i,j \in S, (i,j) \notin prec\}$.
λ	Allowed lag (i.e. the maximum difference between the starting time of a job and its earliest possible starting time according to precedence constraints. The latter is obtained with the length of the longest path in *prec* ending with the job.) *Setting: prec*
th	Thickness (i.e. the maximum number of chain time windows which can include a time unit). *Setting: chains*, chain release dates r_C and chain deadlines $\bar{d}_C$. *Formula:* $max_{0 \leq t \leq D} \lvert \{C \text{ chain in } prec, r_C \leq t < \bar{d}_C\} \rvert$. where $D = max_{C \text{ chain in } prec}(\bar{d}_C)$.

μ	Pathwidth of interval graph $G_{[r_j,d_j)}$ (i.e. the maximum number of overlapping job time windows). *Setting:* release dates r_j and deadlines $\bar{d}_j$. *Formula:* $max_{0\leq t\leq D}\lvert\{j \in \mathcal{J}, r_j \leq t < \bar{d}_j\}\rvert$. where $D = max_{j\in\mathcal{J}}(\bar{d}_j)$.
σ	Slack (i.e. the maximum difference between the time window length of a job and its processing time). *Setting:* release dates r_j and deadlines $\bar{d}_j$. *Formula:* $max_{j\in\mathcal{J}}(\bar{d}_j - r_j - p_j)$.
q	Proper level of interval graph $G_{[r_j,d_j)}$ (i.e. the maximum number of job time windows which can strictly include a time window on both ends). *Setting:* release dates r_j and deadlines $\bar{d}_j$. *Formula:* $max_{j\in\mathcal{J}}\lvert\{i \in \mathcal{J}, (r_i < r_j) \wedge (\bar{d}_j < \bar{d}_i)\}\rvert$.
vc	Minimum size of any vertex cover in interval graph $G_{[r_j,d_j)}$. See Section 7.1. *Setting:* release dates r_j and deadlines $\bar{d}_j$.
tww	Twin-width of interval graph $G_{[r_j,d_j)}$. See Section 7.2. *Setting:* release dates r_j and deadlines $\bar{d}_j$.
$pthin$	Proper thinness of interval graph $G_{[r_j,d_j)}$. See Section 7.2. *Setting:* release dates r_j and deadlines $\bar{d}_j$.
cw	Clique-width of interval graph $G_{[r_j,d_j)}$. See Section 7.2. *Setting:* release dates r_j and deadlines $\bar{d}_j$.
μ_{avg}	Average parameter associated to pathwidth μ. See Section 7.3.
σ_{avg}	Average parameter associated to slack σ. See Section 7.3.
q_{avg}	Average parameter associated to proper level q. See Section 7.3.

A.2.4 Non-Scheduling Problems Mentioned

Classical Problems

3-COLORING
Input: A graph $G = (V, E)$.
Question: Is there a valid 3-coloring of G?
I.e. can you color the nodes in V with three colors so that no two adjacent vertices are of the same color?

DIRECTED BANDWIDTH
Input: A directed acyclic graph $G = (V, E)$, a positive integer b.
Question: Is the directed bandwidth of G at most b?
I.e. is there an injection $f : V \mapsto \mathbb{N}$ such that for every edge $(u, v) \in E$ we have $f(u) < f(v)$ and $f(v) - f(u) \leq b$?

PARTITION
Input: A positive integer T, $2n$ positive integers a_j which sum up to $2T$.
Question: Is there a partition in two parts of n integers such that every part sums up to T?

3-PARTITION
Input: Two positive integers B and T, $3B$ positive integers a_j such that $\frac{T}{4} < a_j < \frac{T}{2}$ and which sum up to BT.
Question: Is there a partition in B parts of 3 integers such that every part sums up to T?

CUTTING STOCK
Input: d item types of sizes $\mathbf{s} = (s_1, \ldots, s_d) \in \mathbb{N}^d$ and multiplicities $\mathbf{n} = (n_1, \ldots, n_k) \in \mathbb{N}^d$,
κ bin types of capacities $\mathbf{a} = (a_1, \ldots, a_\kappa) \in \mathbb{N}^\kappa$ and costs $\mathbf{c} = (c_1, \ldots, c_\kappa) \in \mathbb{N}^\kappa$.
Question: A vector $\mathbf{x} = (x_1, \ldots, x_\kappa) \in \mathbb{N}^\kappa$ of how many bins to buy of each size, and a packing of items to those bins, such that the total cost $\mathbf{c} \cdot \mathbf{x}$ is minimized.

Parameterized Problems

k-BIN PACKING
Input: A positive integer T, n positive integers a_j, a positive integer k.
Parameter: k
Question: Can the n integers a_j be put into at most k bins of capacity T?

k-COLORING
Input: A graph $G = (V, E)$, a positive integer k.
Parameter: k
Question: Is there a valid k-coloring of G?
I.e. can you color the nodes in V with k colors so that no two adjacent vertices are of the same color?

k-CLIQUE
Input: A graph $G = (V, E)$, a positive integer k.
Parameter: k
Question: Is there a clique of size at least k in G?

k-INDEPENDENT SET
Input: A graph $G = (V, E)$, a positive integer k.
Parameter: k
Question: Is there an independent set of size at least k in G?
I.e. are there at least k nodes in V such that there is no edge between them?

k-DOMINATING SET
Input: A graph $G = (V, E)$, a positive integer k.
Parameter: k
Question: Is there a dominating set of size at most k in G?
I.e. is there a subset S of at most k nodes in V such that every node in V is either in S or adjacent to a node in S?

k-BANDWIDTH
Input: A graph $G = (V, E)$, a positive integer k.
Parameter: k
Question: Is the bandwidth of G at most k?
I.e. is there an injection $f : V \mapsto \mathbb{N}$ such that for every arc (u, v) in E we have $|f(v) - f(u)| \leq k$?

k-DIRECTED BANDWIDTH
Input: A directed acyclic graph $G = (V, E)$, a positive integer k.
Parameter: k
Question: Is the directed bandwidth of G at most k?
I.e. is there an injection $f : V \mapsto \mathbb{N}$ such that for every edge $(u, v) \in E$ we have $f(u) < f(v)$ and $f(v) - f(u) \leq k$?

A.3 Figure Index

Problem Maps of Studied Parameters

Figure A.1: Problem maps of the studied parameters.

Parameter Maps of Studied Scheduling Problems

Figure A.2: Parameter maps of some studied scheduling problems.

Bibliography

[ADN13] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. John Wiley & Sons, 2013.

[Bak84] Kenneth R Baker. The effects of input control in a simple scheduling model. *Journal of Operations Management*, 4(2):99–112, 1984.

[Bap10] Philippe Baptiste. A note on scheduling identical coupled tasks in logarithmic time. *Discrete Applied Mathematics*, 158(5):583–587, 2010.

[BBD22] Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022.

[BBKT04] P. Baptiste, P. Brucker, S. Knust, and Vadim Timkovsky. Ten notes on equal-execution-time scheduling. *4OR*, 2:111–127, January 2004.

[BdE19] Flavia Bonomo and Diego de Estrada. On the thinness and proper thinness of a graph. *Discrete Applied Mathematics*, 261:78–92, 2019.

[BDW86] J. Błażewicz, M. Drabowski, and J. We glarz. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Trans. Comput.*, 35(5):389–393, 1986.

[BdWH21] Robert Baart, Mathijs de Weerdt, and Lei He. Single-machine scheduling with release times, deadlines, setup times, and rejection. *European Journal of Operational Research*, 291(2):629–639, 2021. Number: 2 Publisher: Elsevier.

[BF95] Hans L Bodlaender and Michael R Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.

[BFLL+09] Nadia Brauner, Gerd Finke, Vassilissa Lehoux-Lebacque, Chris Potts, and Jonathan Whitehead. Scheduling of coupled tasks and one-machine no-wait robotic cells. *Computers & Operations Research*, 36(2):301–307, February 2009.

[BG19] S. Bessy and R. Giroudeau. Parameterized complexity of a coupled-task scheduling problem. *Journal of Scheduling*, 22(3):305–313, June 2019.

[BGJ77] P. Brucker, M.R. Garey, and D.S. Johnson. Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness. *Math. Oper. Res.*, 2(3):275–284, 1977.

[BGK+24] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width iii: max independent set, min dominating set, and coloring. *SIAM Journal on Computing*, 53(5):1602–1640, 2024.

[BGNS22] Hans L Bodlaender, Carla Groenland, Jesper Nederlof, and Céline MF Swennenhuis. Parameterized problems complete for nondeterministic FPT time and logarithmic space. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 193–204. IEEE, 2022.

[BH21] Jakub Balabán and Petr Hliněnỳ. Twin-width is linear in the poset width. In *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2021.

[BHH99] Peter Brucker, Thomas Hilbig, and Johann Hurink. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 94(1-3):77–99, 1999.

[BHJ24] Jakub Balabán, Petr Hliněnỳ, and Jan Jedelskỳ. Twin-width and transductions of proper k-mixed-thin graphs. *Discrete Mathematics*, page 113876, 2024.

[BJ22] Hauke Brinkop and Klaus Jansen. High multiplicity scheduling on uniform machines in FPT-time. *arXiv preprint arXiv:2203.01741*, 2022.

[BJG08] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Science & Business Media, 2008.

[BJK14] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

[BJS80] Bruno, Jones, and Kimming So. Deterministic scheduling with pipelined processors. *IEEE Transactions on Computers*, C-29(4):308–316, April 1980. Conference Name: IEEE Transactions on Computers.

[BK99] P. Brucker and S. Knust. Complexity results for single-machine problems with positive finish-start time-lags. *Computing*, 63(4):299–316, December 1999.

[BKD10] Peter Brucker, Sigrist Knust, and Christoph Dürr. The scheduling zoo. http://schedulingzoo.lip6.fr/, 2010. [Online; accessed 02-July-2024].

[BKTW21] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width i: tractable fo model checking. *ACM Journal of the ACM (JACM)*, 69(1):1–46, 2021.

[BL96] Jacek Błażwicz and Zhen Liu. Scheduling multiprocessor tasks with chain constraints. *European Journal of Operational Research*, 94(2):231–241, October 1996.

[BM93] Hans L Bodlaender and Rolf H Möhring. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6(2):181–188, 1993.

[Bod21] Hans L Bodlaender. Parameterized complexity of bandwidth of caterpillars and weighted path emulation. In *Graph-Theoretic Concepts in Computer Science: 47th International Workshop, WG 2021, Warsaw, Poland, June 23–25, 2021, Revised Selected Papers 47*, pages 15–27. Springer, 2021.

[BPTW12] Jacek Blazewicz, Grzegorz Pawlak, Michal Tanas, and Wojciech Wojciechowicz. New algorithms for coupled tasks scheduling–a survey. *RAIRO-Operations Research*, 46(4):335–353, 2012.

[BvdW20] Hans L Bodlaender and Marieke van der Wegen. Parameterized complexity of scheduling chains of jobs with delays. In *15th International Symposium on Parameterized and Exact Computation (IPEC)*, 2020.

[CEH+04] Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In Jean-Jacques Levy, Ernst W. Mayr, and John C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics*, pages 209–222, Boston, MA, 2004. Springer US.

[CFK+15] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and

Saket Saurabh. *Parameterized Algorithms*, volume 5. Springer, 2015.

[CJZ18] Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of exact and approximation algorithms for scheduling problems. *Journal of Computer and System Sciences*, 96:1–32, 2018.

[CKX06] Jianer Chen, Iyad A Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In *Mathematical Foundations of Computer Science 2006: 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006. Proceedings 31*, pages 238–249. Springer, 2006.

[CMYZ17] Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2017.

[CZ21] Bo Chen and Xiandong Zhang. Scheduling coupled tasks with exact delays for minimum total job completion time. *J. Sched.*, 24(2):209–221, 2021.

[DF99] R. Downey and M. Fellows. *Parameterized Complexity*. Springer, 1999.

[DFR96] Rodney G Downey, Michael R Fellows, and Kenneth W Regan. Descriptive complexity and the W hierarchy. In *Proof Complexity and Feasible Arithmetics*, pages 119–134, 1996.

[Dij68] Edsger W. Dijkstra. Letters to the editor: go to statement considered harmful. *Commun. ACM*, 11(3):147–148, 1968.

[DKD97] Moshe Dror, Wieslaw Kubiak, and Paolo Dell'Olmo. Scheduling chains to minimize mean flow time. *Information Processing Letters*, 61(6):297–301, March 1997.

[DL89] J. Du and J.Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM J. Discrete Math.*, 2(4):473–487, 1989.

[DLY91] Jianzhong Du, Joseph YT Leung, and Gilbert H Young. Scheduling chain-structured tasks to minimize makespan and mean flow time. *Information and Computation*, 92(2):219–236, 1991. Number: 2 Publisher: Elsevier.

[DPPH00] Ulrich Dorndorf, Erwin Pesch, and Toàn Phan-Huy. A time-oriented branch-and-bound algorithm for resource-constrained project scheduling with generalised precedence constraints. *Management Science*, 46:1365–1384, 2000.

[DPV11] Daniel Dadush, Chris Peikert, and Santosh Vempala. Enumerative lattice algorithms in any norm via m-ellipsoid coverings. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 580–589. IEEE, 2011.

[EdW14] Jan Elffers and Mathijs de Weerdt. Scheduling with two non-unit task lengths is NP-complete. *arXiv preprint arXiv:1412.3095*, 2014.

[EFKR01] Daniel W Engels, Jon Feldman, David R Karger, and Matthias Ruhl. Parallel processor scheduling with delay constraints. In *SODA*, pages 577–585, 2001.

[EFM85] J Erschler, G Fontan, and C Merce. Un nouveau concept de dominance pour l'ordonnancement de travaux sur une machine. *RAIRO-Operations Research*, 19(1):15–26, 1985.

[EFMR83] J. Erschler, Gérard Fontan, Colette Mercé, and François Roubellat. A new dominance concept in scheduling n jobs on a single machine with ready times and due dates. *Oper. Res.*, 31(1):114–127, 1983.

[Eng00] Daniel Wayne Engels. *Scheduling for Hardware/Software Partitioning in Embedded System Design*. PhD thesis, Massachusetts Institute of Technology, 2000.

[EST15] Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: classes and completeness. *Algorithmica*, 71:661–701, 2015.

[FG98] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 1998.

[FGM22] David Fischer, Julian Golak, and Matthias Mnich. Exponentially faster fixed-parameter algorithms for high-multiplicity scheduling. *CoRR*, abs/2203.03600, 2022.

[FL96] Lucian Finta and Zhen Liu. Single machine scheduling subject to precedence delays. *Discrete Applied Mathematics*, 70(3):247–266, October 1996.

[FN96] Birger Franck and Klaus Neumann. Priority-rule methods for the resource-constrained project scheduling problem with minimal and maximal time lags–an empirical analysis. In *The 5th International Workshop on Project Management and Scheduling*, pages 88–91, 1996.

[FT87] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.

[Gab82] Harold N Gabow. An almost-linear algorithm for two-processor scheduling. *Journal of the ACM (JACM)*, 29(3):766–780, 1982.

[GGJK78] Michael R Garey, Ronald L Graham, David S Johnson, and Donald Ervin Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34(3):477–495, 1978.

[GJ77] M. R. Garey and D. S. Johnson. Two-processor scheduling with start-times and deadlines. *SIAM Journal on Computing*, 6(3):416–426, September 1977.

[GJ78] M.R. Garey and D.S. Johnson. Strong NP-completeness results: motivation, examples, and implications. *J. Assoc. Comput. Mach.*, 25(3):499–508, 1978.

[GJ79] Michael R Garey and David S Johnson. *Computers and Intractability*, volume 174. Freeman San Francisco, 1979.

[GJTY83] M.R. Garey, D.S. Johnson, R.E. Tarjan, and M. Yannakakis. Scheduling opposing forests. *SIAM Journal on Algebraic Discrete Methods*, 4(1):72–93, 1983.

[GLLK79] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[GR20] Michel X Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. *Journal of the ACM (JACM)*, 67(6):1–21, 2020.

[GRB16] Michaël Gabay, Christophe Rapine, and Nadia Brauner. High-multiplicity scheduling on one machine with forbidden start and completion times. *Journal of Scheduling*, 19:609–616, 2016.

[GRR19] Frank Gurski, Carolin Rehs, and Jochen Rethmann. Knapsack problems: a parameterized point of view. *Theoretical Computer Science*, 775:93–108, 2019.

[GWY01] Valery S Gordon, Frank Werner, and OA Yanushkevich. Single machine preemptive scheduling to minimize the weighted number of late jobs with deadlines and nested release/due date intervals. *RAIRO-Operations Research*, 35(1):71–83, 2001.

[HBS18] Farhad Habibi, Farnaz Barzinpour, and Seyed Sadjadi. Resource-constrained project scheduling problem: review of past and recent developments. *Journal of Project Management*, 3(2):55–88, 2018.

[HH24] Klaus Heeger and Danny Hermelin. Minimizing the weighted number of tardy jobs is W[1]-hard. *arXiv preprint arXiv:2401.01740*, 2024.

[HKPS21] Danny Hermelin, Shlomo Karhi, Michael Pinedo, and Dvir Shabtay. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research*, 298(1):271–287, 2021. Number: 1 Publisher: Springer.

[HMK23] Claire Hanen and Alix Munier Kordon. Fixed-parameter tractability of scheduling dependent typed tasks subject to release times and deadlines. *Journal of Scheduling*, pages 1–15, 2023.

[HMMK22] Claire Hanen, Maher Mallem, and Alix Munier-Kordon. Parameterized complexity of single-machine scheduling with precedence, release dates and deadlines. In *15th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP)*, 2022.

[HPV75] John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. On time versus space and related problems. In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 57–64, 1975.

[Hu61] T.C. Hu. Parallel sequencing and assembly line problems. *Oper. Res.*, 9:841–848, 1961.

[JKMS13] Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013.

[JKZ24] Klaus Jansen, Kai Kahler, and Esther Zwanger. Exact and approximate high-multiplicity scheduling on identical machines. *arXiv preprint arXiv:2404.17274*, 2024.

[JS11] Antoine Jouglet and David Savourey. Dominance rules for the parallel machine total weighted tardiness scheduling problem with release dates. *Computers & Operations Research*, 38(9):1259–1266, 2011.

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[KK18a] Imed Kacem and Hans Kellerer. Approximation schemes for minimizing the maximum lateness on a single machine with release times under non-availability or deadline constraints. *Algorithmica*, 80(12):3825–3843, 2018.

[KK18b] Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, 21:493–503, 2018.

[KK22] Dušan Knop and Martin Kouteckỳ. Scheduling kernels via configuration LP. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[KKL+19] Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. Multitype integer monoid optimization and applications. *arXiv:1909.07326 [cs, math]*, September 2019. arXiv: 1909.07326.

[Knu74] Donald E. Knuth. Computer programming as an art. *Commun. ACM*, 17(12):667–673, 1974.

[KOŠ19] Pavel Klavík, Yota Otachi, and Jiří Šejnoha. On the classes of interval graphs of limited nesting and count of lengths. *Algorithmica*, 81(4):1490–1511, 2019.

[KOS23] Mostafa Khatami, Daniel Oron, and Amir Salehipour. Scheduling coupled tasks on parallel identical machines. *Optimization Letters*, pages 1–13, 2023.

[KSC20] Mostafa Khatami, Amir Salehipour, and T.C.E. Cheng. Coupled task scheduling with exact delays: literature review and models. *European Journal of Operational Research*, 282(1):19–39, 2020.

[KT21] Alix Munier Kordon and Ning Tang. A fixed-parameter algorithm for scheduling unit dependent tasks with unit communication delays. In *European Conference on Parallel Processing*, pages 105–119. Springer, 2021.

[KZ20] Martin Kouteckỳ and Johannes Zink. Complexity of scheduling few types of jobs on related and unrelated machines. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2020.

[Law73] E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Sci.*, 19:544–546, 1973.

[LJ83] Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[LL08] Joseph Leung and Chung-Lun Li. Scheduling with processing set restrictions: a survey. *International Journal of Production Economics*, 116:251–262, 12 2008.

[Llo81] E.L. Lloyd. Concurrent task systems. *Oper. Res.*, 29(1):189–201, 1981.

[LQ92] Jean B. Lasserre and Maurice Queyranne. Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. In Egon Balas, Gérard Cornuéjols, and Ravi Kannan, editors, *Proceedings of the 2nd Integer Programming and Combinatorial Optimization Conference, Pittsburgh, PA, USA, May 1992*, pages 136–149. Carnegie Mellon University, 1992.

[LRK78] J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26(1):22–35, 1978.

[LRKB77] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Ann. of Discrete Math.*, 1:343–362, 1977.

[LST90] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46:259–271, 1990.

[LVW84] J.Y.-T. Leung, O. Vornberger, and J.D. Witthoff. On some variants of the bandwidth minimization problem. *SIAM J. Comput.*, 13(3):650–667, 1984.

[LZ07] Haibing Li and Hairong Zhao. Scheduling coupled-tasks on a single machine. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, pages 137–142, 2007.

[Mal22] Maher Mallem. Parameterized complexity of a single machine scheduling problem. In *23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, 2022.

[Mar82] Charles U. Martel. Preemptive scheduling with release times, deadlines, and due times. *J. ACM*, 29(3):812–829, 1982.

[Mas03] Monaldo Mastrolilli. Efficient approximation schemes for scheduling problems with release dates and delivery times. *Journal of Scheduling*, 6(6):521–531, 2003.

[MHMK22a] Maher Mallem, Claire Hanen, and Alix Munier-Kordon. Parameterized complexity of a parallel machine scheduling problem. In *17th International Symposium on Parameterized and Exact Computation (IPEC)*, 2022.

[MHMK22b] Maher Mallem, Claire Hanen, and Alix Munier-Kordon. Scheduling coupled tasks with time windows: a parameterized complexity analysis. hal-03837715, 2022.

[MHMK24a] Maher Mallem, Claire Hanen, and Alix Munier-Kordon. A new structural parameter on single machine scheduling with release dates and deadlines. In Amitabh Basu, Ali Ridha Mahjoub, and

Juan José Salazar González, editors, *Combinatorial Optimization*, pages 205–219, Cham, 2024. Springer Nature Switzerland.

[MHMK24b] Maher Mallem, Claire Hanen, and Alix Munier-Kordon. Parameterized complexity: a two-dimensional approach to study scheduling problems. In *25ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, 2024.

[MK21] Alix Munier Kordon. A fixed-parameter algorithm for scheduling unit dependent tasks on parallel machines with time windows. *Discrete Applied Mathematics*, 290:1–6, 2021.

[MORC07] Carlo Mannino, Gianpaolo Oriolo, Federico Ricci, and Sunil Chandran. The stable set problem and the thinness of a graph. *Operations Research Letters*, 35(1):1–9, 2007.

[MRR92] Madhav V Marathe, R Ravi, and C Pandu Rangan. Generalized vertex covering in interval graphs. *Discrete Applied Mathematics*, 39(1):87–93, 1992.

[MS03] Alix Munier and Francis Sourd. Scheduling chains on a single machine with non-negative time lags. *Mathematical Methods of Operations Research*, 57(1):111–123, April 2003.

[MvB18] Matthias Mnich and René van Bevern. Parameterized complexity of machine scheduling: 15 open problems. *Computers & Operations Research*, 100:254–261, December 2018.

[MW15] Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, December 2015.

[O'N11] Thomas E O'Neil. Sub-exponential algorithms for 0/1 knapsack and bin packing. *Unpublished Manuscript*, 2011.

[OP97] Alex J Orman and Chris N Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72(1-2):141–154, 1997.

[PT99] Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3, 1999.

[Sch70] Linus Schrage. Solving resource-constrained network problems by implicit enumeration—nonpreemptive case. *Operations Research*, 18(2):263–278, 1970.

[Sim78] Barbara Simons. A fast algorithm for single processor scheduling. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 246–252, October 1978. ISSN: 0272-5428.

[Sim83] Barbara Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal on Computing*, 12(2):294–299, 1983.

[Sin01] Gaurav Singh. Performance of critical path type algorithms for scheduling on parallel processors. *Operations Research Letters*, 29(1):17–30, August 2001.

[Sin07] Gaurav Singh. Performance of critical path type algorithms with communication delay. *International Journal of Operations Research*, 4(2):8, 2007.

[TCH+23] Istenc Tarhan, Jacques Carlier, Claire Hanen, Antoine Jouglet, and Alix Munier Kordon. Parameterized analysis of a dynamic programming algorithm for a parallel machine scheduling problem. In *European Conference on Parallel Processing*, pages 139–153. Springer, 2023.

[Tim03] V.G. Timkovsky. Identical parallel machines vs. unit-time shops and preemptions vs. chains in scheduling complexity. *European J. Oper. Res.*, 149(2):355–376, 2003.

[Uet02] Marc Uetz. *Algorithms for Deterministic and Stochastic Scheduling*. Cuvillier Verlag, 2002.

[Ull75] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System sciences*, 1975.

[vBBB+16] René van Bevern, Robert Bredereck, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. In Yury Kochetov, Michael Khachay, Vladimir Beresnev, Evgeni Nurminski, and Panos Pardalos, editors, *Discrete Optimization and Operations Research*, pages 105–120, Cham, 2016. Springer International Publishing.

[vBNS17] René van Bevern, Rolf Niedermeier, and Ondřej Suchý. A parameterized complexity view on non-preemptively scheduling interval-constrained jobs: few machines, small looseness, and small slack. *Journal of Scheduling*, 20:255–265, 2017.

[War59] Elizabeth B Ware. Job shop simulation on the ibm 704. In *Preprints of papers presented at the 14th national meeting of the Association for Computing Machinery*, 1959.

[Wik92] Erick D Wikum. *One-Machine Generalized Precedence Constrained Scheduling*. PhD thesis, Georgia Institute of Technology, 1992.

[WLN94] Erick D Wikum, Donna C Llewellyn, and George L Nemhauser. One-machine generalized precedence constrained scheduling problems. *Operations Research Letters*, 16(2):87–99, 1994. Publisher: Elsevier.

[YHL04] Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5):333–348, September 2004.

[Yu96] Wenci Yu. *The Two-Machine Flow Shop Problem with Delays and the One-Machine Total Tardiness Problem*. PhD thesis, Technische Universiteit Eindhoven, 1996.