# Représentations discrètes pour l'ordonnancement et la planification robustes

Tom Portoleau


# THÈSE

**En vue de l'obtention du**

## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

**Délivré par :** *l'Université Toulouse - Jean Jaurès*

---

**Présentée et soutenue le** *22/06/2022* **par :**

Tom PORTOLEAU

**Représentations discrètes pour l'ordonnancement et la planification robustes**

---

**JURY**

| | | |
|---|---|---|
| CHRISTIAN ARTIGUES | Directeur de Recherche | Directeur de thèse |
| OLGA BATTAIA | Professeure | Examinatrice |
| ROMAIN GUILLAUME | Maître de conférence | Directeur de thèse |
| SAFIA KEDAD SIDHOUM | Professeure des Universités | Rapporteuse |
| PATRICE PERNY | Professeur des Universités | Examinateur |
| MICHAËL POSS | Directeur de recherche | Rapporteur |
| ANDRÉ ROSSI | Professeur des Universités | Rapporteur |

---

**École doctorale et spécialité :**

*MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture*

**Unité de Recherche :**

*Laboratoire d'analyse et d'architecture des systèmes*

**Directeur(s) de Thèse :**

*Christian ARTIGUES* et *Romain GUILLAUME*

**Rapporteurs :**

*Safia KEDAD SIDHOUM*, *Michaël POSS* et *André ROSSI*

# Remerciements

J'aimerais tout d'abord remercier Safia Kedad-Sidhoum, Michael Poss et André Rossi pour avoir accepté de rapporter cette thèse, ainsi que les autres membres du jury Olga Battaia et Patrice Perny pqui ont également apporté des conseils avisés.
Je ne remercierais jamais assez Christian Artigues et Romain Guillaume, qui ont encadré cette thèse, et avec qui cela a été un plaisir de travailler durant ces trois années, que cela soit du point de vue scientifique, mais également humain.
J'adresse aussi des remerciement les plus chaleureux à l'équipe ROC, au sein de laquelle j'ai effectué cette thèse, qui m'a apporté un environnement de travail remarquable, particulièrement par les membres qui la compose. Merci à toutes et tous pour le bon temps passé ensemble, que cela au laboratoire ou en dehors, qui ont crée tant de souvenirs que je chérirais. Je tiens à remercier tout particulièrement Valentin, pour avoir accepté de "prendre l'air" même par temps froid si souvent, et Louis, avec qui j'ai partagé un bureau dans lequel se sont déroulées les échanges les plus étranges et -parfois- enrichissants.
Finalement je remercie mes amis (notamment Paul, Quentin et Quentin) qui m'ont rendu le quotidien plus doux dans les périodes moins faciles, ainsi que ma famille qui a toujours été présente et m'a soutenu malgré leur évidente -et compréhensible- incompréhension de ce que je faisais.

## Résumé

Classiquement, en recherche opérationnelle, on cherche à résoudre un problème dont l'ensemble des paramètres est connu avec certitude. Pourtant, c'est rarement le cas en pratique. Par exemple, pour un problème où l'on cherche à ordonnancer un ensemble de tâches, il est possible qu'une des tâches prenne du retard, rendant caduc le reste le planning initialement calculé. Plusieurs approches existent pour aborder ces incertitudes, et parmi elles, nous nous intéressons plus particulièrement à l'optimisation robuste, où les incertitudes sont modélisées par des scénarios, et où l'on cherche à calculer une solution qui se dégrade le moins possible dans son pire scénario. Néanmoins, ces approches ont tendance à produire des solutions trop pessimistes. Pour pallier cela, nous proposons un modèle d'aide à la décision, basé sur des arbres de décisions, qui propose à un décideur des solutions alternatives qui s'adaptent au scénario pendant qu'il se réalise, en fonction des informations accessibles au décideur. Nous avons appliqué ces arbres de décision à des problèmes d'ordonnancement de ligne d'assemblage aéronautique, que l'on peut voir comme un problème d'ordonnancement multi-mode, sur des instances de références ainsi que sur des instances industrielles réelles, et avons montré l'intérêt de notre modèle. Nous proposons ensuite d'améliorer la résolution de certains problèmes d'optimisation robuste avec une variante d'une méthode classique, la méthode de Benders Adversariale. Cette méthode repose en partie sur l'approximation de l'ensemble des scénarios par un sous-ensemble, d'abord vide, puis auquel on ajoute des scénarios un par un. Dans notre variante, les scénarios sont ajoutés par petits groupes en étant représentés dans ce qu'on appelle un graphe de budget. Nous montrons que la méthode de sélection du groupe de scénario est critique, et appliquons notre variante sur un problème de planification de production.

**Mots clés :** Optimisation robuste, Arbre de décision, Optimisation combinatoire, Ordonnancement de projet, Planification de production

# Abstract

Classically, in operations research, one seeks to solve a problem whose set of parameters is known with certainty. However, this is rarely the case in practice. For example, for a problem in which we want to schedule a set of tasks, it is possible that one of the tasks falls behind, making the rest of the initially calculated schedule infeasible. Several approaches exist to deal with those uncertainties, and among them, we are particularly interested in robust optimization, where the uncertainties are modeled by scenarios, and where we try to compute a solution that degrades as little as possible in its worst case scenario. Nevertheless, those approaches tend to produce solutions that are too pessimistic. To overcome this, we propose a decision analysis model, based on decision trees, which proposes to a decision maker alternative solutions that adapt to the scenario as it unfolds, according to the information available to the decision maker. We have applied those decision trees to aircraft assembly line scheduling problems, which can be viewed as a multi-mode project scheduling problem, on reference instances as well as on real industrial instances, and have shown the interest of our model. We then propose to improve the solution of some robust optimization problems with a variant of a classical method, the Adversarial Benders method. This method relies in part on the approximation of the set of scenarios by a subset, initially empty, to which scenarios are added one by one. In our variant, scenarios are added in small groups by being represented in what is called a budget graph. We show that the method of selecting the scenario group is critical, and apply our variant on a production planning problem.

**Keywords :**

Robust optimization, Decision tree, Combinatorial optimization, Project scheduling, Lot-sizing

# Table des matières

# Introduction

La prise en compte des incertitudes dans les problématiques d'aide à la décision et d'optimisation est un sujet vastement étudié depuis des décennies, et particulièrement d'un point de vue théorique. Pourtant, il est peu courant que les industriels intègrent, de manière concrète, l'incertitude inhérente à leurs activités, alors même qu'ils possèdent parfois des outils et modèles théoriques pour résoudre leurs problèmes. On peut expliquer cela en partie de deux façons. D'abord, la prise en compte des incertitudes a tendance à rendre la résolution de problèmes d'optimisation plus difficiles. Ensuite, cela s'explique aussi par une certaine confiance en leurs opérateurs, agents, compagnons, etc, qui, de par leur savoir-faire et leur connaissance du terrain, parviennent en général à prendre des bonnes décisions pour mitiger les effets négatifs d'aléas non désirés.

Cette thèse, qui cherche à apporter des éléments de réponse à cette problématique, s'inscrit dans le cadre du projet ANR *PER4MANCE* (Planification Et Répartition Flexible du travail entre les OpérateuRs des chaînes d'asseMblage AéroNautiques : une approChe systémique pour gérer les risques Ergonomiques et économiques), qui fait intervenir plusieurs acteurs académiques (ISAE-Supaéro, LAAS-CNRS, SCOTE, IRIT) et deux acteurs industriels, Dassault-Aviation et Airbus. Les objectifs de ce projet sont multiples. D'abord, il s'agit d'effectuer une étude ergonomique des diverses tâches effectuées par les opérateurs sur les lignes d'assemblages aéronautiques, dans le but d'une part de quantifier la pénibilité de ces tâches, et d'autre part d'intégrer de nouvelles contraintes liées à l'ergonomie dans les modèles mathématiques utilisés pour le calcul des plannings des opérateurs. Les autres objectifs concernent la prise en compte des incertitudes sur les lignes d'assemblages lors de la création des plannings pour les opérateurs. Ces incertitudes sont séparées en deux catégories distinctes, et chacune des deux étant couverte par une thèse financée par ce projet. La première concerne les incertitudes rares et peu prévisibles qui peuvent être approchées par des méthodes de réparation de solution, ou de ré-optimisation. La seconde catégorie couvre les incertitudes à impact plus modéré, relativement fréquentes, et prévisibles. C'est à ce type d'incertitude que l'on s'intéressera durant cette thèse, à travers le prisme de l'optimisation robuste. Plus particulièrement, nous proposerons dans cette thèse de nouveaux outils et méthodologies à la fois pour l'aide à la décision pour des cas industriels pratiques et pour la résolution théorique de certains problèmes d'optimisation robuste.

Dans le chapitre 1, nous proposons une revue de littérature dans laquelle nous introduisons les concepts et contextes théoriques dans lesquels se placent les travaux présentés dans ce manuscrit. Nous commençons par un bref historique sur les techniques et modèles mis au point pour aborder les problèmes d'ordonnancement jusqu'à aujourd'hui. Ensuite nous présentons plus en détails les formalismes permettant de représenter les incertitudes dans un problème d'optimisation et différentes manières de les approcher, avec un focus particulier sur les problèmes d'optimisation multi-étape et les méthodes de résolution

de problèmes d'ordonnancement robuste. Finalement nous concluons ce chapitre avec quelques résultats de complexité algorithmique sur des problèmes liés à l'ordonnancement de projet sous incertitude.

Le second chapitre porte spécifiquement sur le problème d'ordonnancement de lignes d'assemblages aéronautiques, dans un cadre déterministe. Nous y présentons d'abord comment ce problème se présente sous la forme d'un *Multi-Mode Resource-Constrained project Scheduling Problem* (MMRCPSP) avec un objectif de nivellement de ressources, puis un modèle de programmation par contraintes pour le résoudre. Nous proposons ensuite une heuristique basée sur ce modèle, de type *Large Neighbourhood Search*, pour laquelle nous définissons un nouveau voisinage, spécifiquement adapté à l'objectif du problème. Nous concluons avec des résultats expérimentaux montrant l'intérêt de notre voisinage aussi bien sur des instances réelles fournies par Airbus que sur des instances de MMRCPSP de la litttérature. Les travaux présentés dans ce chapitre ont mené à un article en révision dans *Computers and Industrial Engineering* [Borreguero 2021], co-écrit avec Tamara Borreguero Sanchidrián, alors doctorante chez Airbus Madrid et à l'université polytechnique de Madrid. Seules mes contributions apparaissent dans ce chapitre. Ces travaux ont obtenu le 2ème prix au concours du meilleur papier étudiant à la conférence PMS 2022 [Portoleau 2022a].

Dans le chapitre 3, nous introduisons les arbres de décision robustes, dont l'idée fondamentale repose sur le fait qu'à certains moments durant le déroulement d'un ordonnancement calculé de manière robuste dans un contexte incertain, un décideur a accès à de l'information à propos du scénario en cours. Nous commençons par introduire formellement les arbres de décision robustes ainsi que ce que l'on nomme information. Nous expliquons ensuite comment calculer un tel arbre et définissons le problème de partitionnement robuste qui doit être résolu en chaque nœud lors de la création de l'arbre, et dont l'objectif est de calculer une partition de l'ensemble des scénarios en fonction des informations disponibles à un moment donné, avec une problématique axée sur la sélection d'information pertinente. Nous présentons ensuite deux algorithmes pour résoudre ce problème, un premier, naïf, puis un second, qui reprend l'idée d'un *Branch and Bound*. Nous avons choisi un problème d'ordonnancement à une machine dans lequel on cherche à minimiser le retard maximal afin d'évaluer l'intérêt des arbres de décision robustes, et les notions et algorithmes présentés dans ce chapitre sont accompagnés d'exemples basés sur ce problème. Nous concluons avec des résultats expérimentaux attestant de l'intérêt des arbres de décision robustes. Ce chapitre a fait l'objet d'une publication dans la conférence internationale avec actes *Information Processing and Management of Uncertainty in Knowledge-Based Systems* 2020 [Portoleau 2020b]. Ces travaux ont été également présentés dans les conférences ROADEF 2020 [Portoleau 2020a] et PMS 2020/2021 [Portoleau 2021b].

Le chapitre 4 propose d'appliquer le concept d'arbres de décision robustes introduit dans le chapitre 3 au problème d'ordonnancement de lignes d'assemblage aéronautiques présenté dans le chapitre 2. Nous commençons par introduire le contexte industriel et

rappelons le problème en jeu. Ensuite, après avoir rappelé brièvement l'idée des arbres de décision robustes, nous expliquons comment les utiliser pour ce problème et proposons une méthode pour résoudre le problème de partitionnement robuste spécifique au problème considéré. Encore une fois, les divers concepts et algorithmes présentés sont illustrés avec des exemples. Nous concluons avec des résultats expérimentaux sur des instances issues d'instances d'ordonnancement de projet multi-mode de la littérature et sur des instances réelles fournies par Airbus. L'ensemble des résultats de ce chapitre fait l'objet d'un article soumis au journal *European Journal of Operational Research*. Une version libre de ce papier est disponible dans HAL (voir [Portoleau 2021d]). Ces travaux ont été également présentés aux conférences ROADEF 2021 [Portoleau 2021a] et EURO 2021 [Portoleau 2021c].

Le dernier chapitre porte quant à lui sur un problème de planification de production sous incertitude, et pour lequel nous proposons une variante de la méthode de Benders adverse, une méthode itérative bien connue pour aborder certains types de problème d'optimisation robuste, dont l'idée repose sur l'approximation de l'ensemble de scénarios par une liste de scénarios discrets, à laquelle on ajoute les scénarios un par un, à chaque itération de la méthode. Nous commençons par présenter la méthode de Benders Adverse et comment résoudre le problème de planification robuste avec celle-ci, en utilisant des formulations mathématiques basées sur les graphes de budget. Nous présentons ensuite la variante que nous proposons, elle basée sur les graphes partiels de budget, que nous définissons, et dont l'idée est d'ajouter à chaque itération de la méthode non pas un unique scénario, mais un ensemble de scénarios bien choisi. Nous introduisons différentes méthodes de sélection de scénarios. Nous concluons avec des résultats expérimentaux montrant que notre variante semble plus efficace que la méthode de Benders adverse classique pour résoudre ce problème. Les travaux présentés dans ce chapitre ont été présentés à la conférence ROADEF 2022 et ont été finalistes du prix du meilleur article étudiant [Portoleau 2022b].

Finalement, les différentes parties de cette thèse sont discutées dans une conclusion, dans laquelle nous évoquons aussi différentes perspectives de recherches.
Des publications correspondant à des travaux réalisés pendant la thèse mais qui ne sont pas en lien avec celle-ci sont fournies en annexe.

CHAPITRE 1

# Etat de l'art

**Sommaire**



## 1.1 Ordonnancement de projet

L'ordonnancement de projet peut être vu comme l'ensemble des théories, techniques et outils permettant l'élaboration d'un planning dans le but d'achever un ensemble de tâches, qui constituent un projet.

Dans [Snyder 1987] et [Morris 1987] les auteurs s'accordent pour marquer le début de l'ordonnancement de projet moderne avec l'année 1958, qui a vu l'apparition quasi simultanée des techniques CPM (critical path method ou méthode du chemin critique) et PERT (program evaluation and review technic ou technique d'évaluation et d'examen des programmes). Ces méthodes, bien que peu utilisées dans leur définition originelle aujourd'hui, ont été initialement développées pour des projets militaires américains et ont connu de nombreuses applications, telles que la construction du World Trade Center entre 1966 et 1975. Ces deux approches -bien distinctes à l'origine- ont depuis bien évolué et sont aujourd'hui indistinctement connue sous le nom de PERT. La méthode PERT, qui est aujourd'hui enseignée dans de nombreux cours de management, est un bon exemple pour illustrer ce à quoi ressemble l'ordonnancement de projet moderne.

### 1.1.1 PERT

L'idée de PERT est de représenter les différentes activités d'un projet sous forme de graphe d'activités et d'en étudier certaines propriétés. L'idée de ce graphe est de représenter les différentes tâches qui constituent un projet, et leurs relations de précédence. Ainsi, un arc partant d'un noeud $a$ vers noeud $b$ signifie simplement que la tâche $b$ ne peut être démarrée tant que la tâche $a$ n'est pas terminée. On peut ensuite étiqueter les arcs avec la durée des tâches à leurs origines respectives. Dans ce cas, on appelle chemin critique le plus long chemin dans ce graphe, et l'on dit que les tâches sur ce chemin sont des tâches critiques. Ici, "critique" fait référence au fait que la longueur de ce chemin est égale à la durée minimale du projet, mais aussi au fait que si l'une des tâches qui le constituent se retrouve retardée, cela impactera l'ensemble du projet. La taille du graphe d'activités est clairement proportionnelle à la taille du projet. Ainsi, pour les plus gros projets, cette technique n'a pas pu être appliquée en pratique avant l'apparition d'ordinateurs suffisamment puissants dans les années 60.
Cette approche possède en réalité deux défauts majeurs, qui touchent tous deux l'expressivité du graphe d'activités. Tout d'abord, le graphe d'activités suppose que les durées des tâches sont connues, ou plutôt, ne sont soumises à aucune forme d'incertitude. Cette hypothèse, utile lorsqu'il s'agit de s'attaquer à la partie calculatoire d'un problème se révèle peu adéquate lorsque l'on souhaite modéliser un problème industriel réel, et peut mener à des calculs, certes plus faciles qu'en prenant en compte des incertitudes, de plannings malheureusement inutilisables en pratique. De nouvelles représentations de graphe d'activités ont vu le jour depuis pour prendre en compte les incertitudes sur la durée des tâches, que nous aborderons à la fin de ce chapitre. L'autre défaut est que le graphe d'activités ne tient pas compte du fait que pour certains projets, des tâches aient éventuellement besoin de ressources pour être réalisées, ce qui implique alors l'existence de contraintes autres que les contraintes de précédence. Ainsi, certains projets ne pouvaient pas être approchés et analysés correctement par PERT.

### 1.1.2 L'ordonnancement en tant que problème combinatoire

Parallèlement à l'ordonnancement de projet s'est développée l'étude de certains problèmes combinatoires appelés problèmes d'ordonnancement. Dans [Pinedo 2012] l'ordonnancement est défini comme "l'allocation des ressources aux tâches sur des périodes données et dont le but est d'optimiser un ou plusieurs objectifs". Dans [Potts 2009] l'auteur cite l'article [Johnson 1954] comme point de départ à partir duquel les problèmes d'ordonnancement constituent un domaine à part entière dans la recherche opérationnelle. On peut distinguer ces problèmes en plusieurs familles distinctes.
Tout d'abord on peut citer les problèmes dits à une machine et les problèmes à machines parallèles. Ces problèmes ont été étudiés pour la première fois dans les années 50, dans lesquelles on trouve quelques articles de référence, comme [Jackson 1955], [Jackson 1956] et [Smith 1956] ou encore [McNaughton 1959]. Dans ces problèmes, on se donne un certain nombre de machines, et l'on cherche à affecter chaque tâche à l'une de ces machines. On peut chercher à -dans le cas de plusieurs machines parallèles- minimiser le temps d'exécu-

tion total de l'ensemble des tâches, ou encore, si les tâches ont chacune une date de rendu, à minimiser le retard de la tâche le plus en retard, ou la somme des retards de toutes les tâches.
On trouve également la famille des problèmes d'ordonnancement d'atelier. Le problème du flow-shop consiste à ordonnancer une séquence de tâches, de sorte qu'une fois un ordre fixé, chaque tâche passera dans ce même ordre dans une séquence de machine, tout en respectant un ensemble de contraintes. On peut également citer le problème du job-shop, dans lequel chaque travail (ou *job*) se décompose en une séquence de tâches qui doit être effectuée dans un ordre précis, et chaque tâche est attribuée à une (ou plusieurs dans le cas du job-shop flexible) machine spécifique. Finalement, il existe aussi les problèmes d'open-shop, où l'on cherche à ordonnancer un ensemble de tâches étant donné un ensemble de machines et le temps que doit passer chaque tâche sur chaque machine.
Tous ces problèmes existent dans de nombreuses variantes, qui peuvent faire appel à des contraintes supplémentaires, ou à des objectifs particuliers. Excepté un problème d'ordonnancement à une machine qui sera abordé dans le Chapitre 2, les autres problèmes exposés ici ne seront pas traités dans cette thèse.

### 1.1.3 Ordonnancement de projets sous contraintes de ressources

Le type de problème que nous allons principalement traiter dans cette thèse est le problème d'ordonnancement de projets sous contraintes de ressources (RCPSP, de l'anglais *Resource-Constrained Project Scheduling Problem*).

L'article [Hartmann 2022] présente une classification des problèmes d'ordonnancement de projets sous contraintes de ressources ainsi qu'un aperçu des méthodes de résolution existantes.

Le RCPSP n'est autre qu'une variante NP-difficile du PERT qui considère un ensemble de ressources en plus de l'ensemble des tâches de telle sorte que chaque tâche mobilise tout au long de son exécution un nombre positif ou nul d'unités sur chaque ressource. Chaque ressource ayant une capacité limitée, le nombre de tâches pouvant être exécutées en parallèle par chaque ressource est également limité car la somme des nombres d'unités requises par les tâches à chaque instant ne doit pas excéder sa capacité.

Ce modèle est assez général car il permet de représenter les problèmes à une machine, les problèmes à machines parallèles et les problèmes d'atelier de type flow-shop, job-shop et open-shop. De plus il se trouve au cœur de nombreux problèmes industriels. Comme nous le verrons dans le chapitre 1, le problème d'ordonnancement de lignes d'assemblage aéronautiques résolu dans cette thèse est un RCPSP qui comprend en outre des modes d'exécution multiples pour les tâches (MMRCPSP pour *Multi-Mode Resource-Constrained Project Scheduling Problem*).

Dans le MMRCPSP, chaque tâche a un ensemble de modes d'exécution. Un mode définit une durée possible de la tâche et un nombre d'unités requises sur chaque ressource pour obtenir cette durée. La logique est qu'un mode avec une durée plus petite mobilise plus de ressources qu'un mode avec une durée plus grande.

La littérature sur le MMRCPSP est assez riche en ce qui concerne la minimisation de la durée totale de l'ordonnancement (makespan) [Drexl 1993, De Reyck 1999, Węglarz 2011,

Coelho 2011].

Une vue d'ensemble assez récente des modèles et des approches de résolution des (MM)RCPSP peut être trouvée dans [Schwindt 2015]. En relation avec la problématique industrielle dans l'assemblage aéronautique, nous nous intéressons à la problématique de lissage des ressources utilisées avec une durée totale d'ordonnancement fixée. Dans la littérature sur le MMRCPSP, on parle de *resource leveling* [Demeulemeester 2002, Schwindt 2015, Coughlan 2015]. Il s'agit de minimiser la quantité maximum de chaque ressource utilisée à chaque instant (le pic d'utilisation). Lorsque des coûts d'acquisition sont associés aux ressources on parle de *resource investment problem* [Gerhards 2020] ou *resource-availability cost* [Demeulemeester 1995].

## 1.2 Optimisation et ordonnancement robuste

Dans le cadre de cette thèse, les problèmes qui seront étudiés le seront en supposant qu'ils ont une composante incertaine. On distingue en général deux grandes familles d'approches pour aborder les problèmes soumis à de l'incertitude. La première est l'optimisation stochastique. Dans ces approches, on suppose que les paramètres incertains du problème peuvent être décrits par des lois de probabilités. Quand c'est le cas, on cherche alors en général à optimiser des quantités, usuelles quand on parle de probabilités, comme la moyenne ou la variance. Par exemple, on peut chercher à maximiser en moyenne la probabilité qu'une solution satisfasse toutes les contraintes du problème -on appelle ce genre de problème *chance constraint*-, ou encore on peut chercher à optimiser la valeur de la fonction objectif d'une solution en moyenne. Cependant avec ce genre d'approche, il est possible que la valeur de la solution varie grandement selon la réalisation des incertitudes, ce qui n'est pas souhaitable dans certains cas. On peut régler le problème en cherchant également à minimiser la variance de la qualité de la solution, et considérer les deux critères à la fois par des approches multi-objectif, mais cela a tendance à rendre les problèmes encore plus complexes à résoudre. La seconde grande famille d'approches permet en quelque sorte de faire d'une pierre deux coups, c'est-à-dire se prémunir le plus possible des "mauvaises" réalisations des incertitudes tout en garantissant une certaine qualité de la solution. On nomme cela l'optimisation robuste, et c'est cette approche de l'optimisation sous incertitude que l'on va considérer dans cette thèse.

### 1.2.1 Ensemble de scénarios

En optimisation robuste, on suppose en général qu'on ne connaît pas de distribution de probabilité sur les paramètres incertains du problème, mais on suppose que l'on connaît un ensemble de scénarios.

Ces ensembles de scénarios peuvent avoir plusieurs formes. On note $X = (X_i)_{i \in I}$ une réalisation -ou scénario- des paramètres incertains, $X_i$. On peut citer quelques exemples.

Ensemble de scénarios discret : les scénarios sont décrits par une liste finie de scénarios

possibles. L'ensemble des scénarios peut s'écrire :

$$\Omega = \{X_1, X_2, \cdots, X_n\}.$$

Scénarios intervalles : chaque paramètre incertain se réalise dans un intervalle. Un scénario est alors le produit cartésien de tous les intervalles d'incertitude. L'ensemble des scénarios peut s'écrire :

$$\Omega = \prod_{i \in I} [\underline{X_i}, \overline{X_i}]$$

où $\underline{X_i}$ et $\overline{X_i}$ sont respectivement les bornes inférieures et supérieures de l'intervalle d'incertitude du paramètre $X_i$. Ce type d'incertitudes sera considéré dans les chapitres 3 et 4 de cette thèse.

Scénarios sous budget d'incertitudes : l'amplitude de l'incertitude par rapport à une valeur nominale est contrôlée par un paramètre appelé budget d'incertitudes, noté $\Gamma$. On peut distinguer deux sous-familles classiques pour ce genre d'incertitude. On parle de *Cardinality Constrained Uncertainty* lorsque chaque paramètre incertain est soumis à un budget différent, et que le nombre de paramètre pouvant varier est borné. On peut l'écrire :

$$\Omega = \{X | X_i \in [\hat{X}_i - \Gamma_i, \hat{X}_i + \Gamma_i], \sum_i \mathbb{1}(X_i \neq \hat{X}_i) \leq \Gamma\}$$

où $\hat{X}_i$, est la valeur nominale du paramètre $X_i$, et $\Gamma_i$ le budget d'incertitudes alloué à ce paramètre. La seconde sous-famille, nommée *Norm Uncertainty* permet en quelque sorte de corréler les différentes incertitudes en forçant la norme de la différence entre le scénario nominal et le scénario considéré à être plus petit que le budget d'incertitudes. Cela s'écrit :

$$\Omega = \{X | \|\hat{X} - X\| \leq \Gamma\}$$

où $\hat{X}$ est le scénario nominal et $\|.\|$ une norme quelconque. Bien entendu, le choix de la norme a une importance sur la forme de l'ensemble des scénarios. Ce type d'incertitudes sera considéré dans le chapitre 5 de ce manuscrit.

### 1.2.2 Critères de robustesse

Pour mesurer la robustesse d'une solution, il existe plusieurs critères de robustesse, la plupart introduits par [Kouvelis 1997] qui sont largement utilisés dans les approches d'optimisation robuste. Le premier, le plus classique, est le critère *minmax*. L'objectif de ce critère est de calculer la solution qui est la plus intéressante dans son pire scénario possible. Plus formellement cela s'écrit :

$$s^* = \arg\min_{s \in S} \max_{\omega \in \Omega} f(\omega, s)$$

où $f(\omega, s)$ est la valeur du critère que l'on cherche à optimiser d'une solution $s$ dans le scénario $\omega$. Le critère *minmax* est très courant dans la littérature, mais est également connu

pour être -très- pessimiste [Bertsimas 2004]. En effet, celui-ci cherche à optimiser une solution seulement par rapport à son scénario pire cas. Même si cela donne une garantie -ou borne- sur la qualité de la solution, ce critère a tendance à forcer les solutions à être très "prudentes", ou conservatives, ce qui a pour conséquence que les solutions robustes obtenues à partir de ce critère sont en général de mauvaise qualité sur l'ensemble des scénarios -mais évidemment jamais pire que sur leur pire scénario. Ce critère reste cependant très adapté dans certains cas, par exemple lorsque le pire cas implique des risques humains, à éviter ou minimiser le plus fortement possible. Ce critère sera utilisé dans les chapitres 3, 4 et 5 de ce manuscrit, et l'on cherchera à réduire son pessimisme par la considération du budget d'incertitude au chapitre 5.

Pour réduire ce pessimisme, il existe un autre critère, nommé *minmax-regret* -ou aussi *robust deviation*- où l'on cherche à minimiser la différence entre le coût d'une solution dans un scénario donné et le coût de la solution optimale pour ce scénario. Formellement cela peut s'écrire :

$$s^* = \arg\min_{s \in S} \max_{\omega \in \Omega} |f(\omega, s) - f(\omega, s^*_\omega)|$$

où $s^*_\omega$ est la meilleure solution possible dans le scénario $\omega$. Le regret peut être considéré comme une perte d'opportunité, parce qu'il représente la différence entre la performance d'une solution donnée, et la performance de la solution qui aurait été choisie si l'on avait connu l'état de la nature a priori.

D'autres moyens de réduction du pessimisme, plus spécifiques aux problèmes d'ordonnancement, sont présentés plus loin dans ce chapitre. Dans les chapitres 3 et 4 nous introduisons un modèle, basé sur les arbres de décision, où l'on cherche à apporter de l'optimisme au critère *minmax*. Ce modèle se base sur la possibilité d'avoir accès à des informations à propos du déroulement du scénario en cours et de pouvoir s'y adapter. Dans la littérature, les problèmes qui font cette hypothèse sont appelés les problèmes d'optimisation multi-étape (*multi-stage*) et sont l'objet de la prochaine section de ce chapitre.

### 1.2.3 Optimisation robuste multi-étapes

En programmation mathématique, on peut écrire de manière générique un problème d'optimisation robuste à 2 étapes :

$$\min_{x_1} \max_{\omega} \min_{x_2} \quad f(\omega, x_1, x_2) \tag{1.1}$$

$$s.t \quad g(\omega, x_1, x_2) \leq 0 \quad \forall \omega \in \Omega \tag{1.2}$$

où $f$ et $g$ sont des fonctions quelconques, les variables $x_1$ sont les variables du premier niveau de décision, c'est-à-dire les décisions qui sont prises immédiatement, et les variables $x_2$ sont quant à elles des variables de recours, elles dénotent la possibilité de prendre une décision une fois que le scénario est découvert. A priori, la solution "rêvée" à ce problème est une solution où les variables de recours s'adaptent de manière optimale à chacun des scénarios. Dans la littérature on peut retrouver cette solution sous le nom de solution totalement adaptative (*fully adjustable solution*). Cependant il a été montré que

dans le cas général, même avec seulement deux niveaux de décision, résoudre ce problème est NP-Difficile, et intraitable en pratique [Ben-Tal 2004]. On peut alors essayer de faire des hypothèses pour le rendre plus abordable.

Par exemple, toujours dans [Ben-Tal 2004], les auteurs montrent que si l'on restreint les variables de recours à être des fonctions affines du scénario révélé, ou plus formellement poser $x_2 = Q\omega + q$ alors le problème :

$$\min_{x_1,Q,q} \max_{\omega} \quad f(\omega, x_1, Q\omega + q) \tag{1.3}$$

$$s.t \quad g(\omega, x_1, Q\omega + q) \leq 0 \quad \forall \omega \in \Omega \tag{1.4}$$

où $f$ et $g$ sont linéaires et $g(u, x_1, Q\omega + q) = A(\omega)x_1 + B(Q\omega + q)$ avec $A$ une matrice qui dépend du scénario et $B$ une matrice qui n'en dépend pas, alors la complexité du problème devient polynomial. Plus précisément, si on laisse $B$ être une fonction du scénario, on obtient un problème quadratique, qui peut également être facile, selon la forme de l'ensemble des scénarios.

Une autre manière de rendre le problème soluble en pratique, toujours en restreignant les variables de recours, est une méthode proposée dans [Hanasusanto 2015] appelée la K-adaptabilité. L'idée de cette méthode est de considérer un nombre fini (K) de politiques de recours et d'appliquer la meilleure d'entre elles au second niveau de décision. Par exemple, dans un problème d'ordonnancement où les dates de rendus sont incertaines et où on souhaite minimiser la somme des retards, une politique de recours possible est d'ordonnancer au plus tôt les tâches dont la due-date est la plus tôt (*Earliest Due Date*). Du point de vue de la programmation mathématique, leur méthode revient à résoudre :

$$\min_{x_1, x_2^k} \max_{\omega} \min_{k \leq K} \quad f(\omega, x_1, x_2^k) \tag{1.5}$$

$$s.t \quad g(\omega, x_1, x_2^k) \leq 0 \quad \forall \omega \in \Omega, k \leq K \tag{1.6}$$

où les variables $x_2^k$, pour $k \leq K$ correspondent au recours en suivant la $k$-ieme politique prédéfinie.

Une autre manière d'approcher le problème est l'approche dite d'adaptabilité finie (*Finite Adaptability*) proposée dans [Bertsimas 2010] et [Postek 2016]. Cette fois-ci, l'idée n'est pas de contraindre les recours possibles, mais de "relâcher" l'ensemble des scénarios. Pour ce faire, les auteurs proposent de calculer une partition -de taille fixée $k$- de l'ensemble des scénarios. Ainsi, le problème devient

$$\min_{x_1} \max_{\omega} \min_{i \leq k} \quad f(\omega, x_1, x_2^i) \tag{1.7}$$

$$s.t \quad g(\omega, x_1, x_2^1) \leq 0 \quad \forall \omega \in \Omega_1 \tag{1.8}$$

$$g(\omega, x_1, x_2^2) \leq 0 \quad \forall \omega \in \Omega_2 \tag{1.9}$$

$$\dots \tag{1.10}$$

$$g(\omega, x_1, x_2^k) \leq 0 \quad \forall \omega \in \Omega_k \tag{1.11}$$

où $(\Omega_1, \Omega_2, \cdot, \Omega_k)$ est une partition de $\Omega$, et $x_2$ des recours précalculés. Avec cette approche les recours $x_2$ sont calculés non pas pour être adaptés à un scénario donné, mais à un sous-ensemble précalculé de scénarios. L'approche que nous proposons dans ce chapitre reprend cette idée à la différence près que la partition n'est pas donnée en entrée du problème mais est calculée localement à chaque niveau de décision.

En pratique, ces différentes approches ont étés validées sur de nombreux problèmes. En ce qui concerne les problèmes d'ordonnancement, on peut citer [Cohen 2021], dans lequel les auteurs proposent une approche multi-étape pour résoudre un problème d'ordonnancement sur machines parallèles dans lequel le décideur peut réordonnancer à chaque fois qu'une tâche se termine, en résolvant un PLNE. Ces méthodes sont également utilisées pour des problèmes de flots ([Ordónez 2007], [Atamtürk 2007]) de gestion de trafic [Mudchanatongsuk 2005], de conception de réseau [Silva 2018], de conception de circuit intégré [Mani 2006], de gestion de portefeuille de projet [Calafiore 2005], de protection contre les inondations [Postek 2019], ou encore de planification de production d'électricité [Bertsimas 2012]. Notons que des travaux considèrent également l'optimisation robuste adaptative avec budget d'incertitude [Ayoub 2016].

### 1.2.4 Ordonnancement sous incertitudes

Pour le cas particulier des problèmes d'ordonnancement, des travaux étudient la complexité des contreparties robustes minmax des problèmes d'ordonnancement déterministes avec ou sans budget d'incertitude [Aloulou 2008, Bougeret 2019]. Il existe aussi dans la littérature des approches plus adaptées à ce genre de problème. Nous en donnons ici quelques exemples.

**Ordonnancement proactif-réactif.** Dans les problèmes d'ordonnancement du monde réel, comme dans les problèmes d'ordonnancement des chaînes de montage aéronautiques qui feront l'objet de notre étude de cas industriel, plusieurs paramètres sont sujets à l'incertitude. De plus, la connaissance que le décideur a de ces paramètres s'inscrit dans un contexte dynamique, c'est-à-dire que les valeurs réelles des paramètres sont révélées progressivement dans le temps. La prise en compte des aspects incertains et dynamiques dans les approches de résolution des problèmes d'ordonnancement est nécessaire pour contrôler le risque et la flexibilité de l'ordonnancement.

Historiquement, deux approches générales ont émergé pour traiter l'incertitude dans les problèmes d'ordonnancement.

D'une part, les méthodes proactives élaborent un ordonnancement de base initial tout en prenant en compte les éventuels aléas afin de le rendre aussi robuste que possible. Cependant, un défaut majeur des méthodes d'ordonnancement proactives est leur pessimisme [Nikulin 2006], les solutions les plus robustes ont tendance à être de faible qualité en termes de fonction objectif.

C'est particulièrement le cas lorsque les incertitudes sont importantes et fréquentes.

Dans ces cas, les méthodes réactives, qui visent à calculer rapidement de nouvelles solutions pour faire face à l'imprévu et adapter le planning au scénario qui se présente, sont plus appropriées car elles ne se concentrent pas spécialement sur l'ordonnancement initial mais plutôt sur la manière de le modifier en ligne, c'est-à-dire pendant l'exécution. Elles sont généralement basées sur des règles de priorité, ou de recherche locale [Rajendran 1999] qui permettent au décideur de calculer rapidement de nouvelles solutions en fonction du scénario en cours. Cependant, contrairement aux approches proactives, il n'y a aucune garantie concernant la valeur de l'objectif des solutions calculées avec une méthode réactive.

Dans la littérature sur l'ordonnancement, les approches qui combinent une phase proactive visant à émettre un ordonnancement de base robuste et une phase réactive qui adapte l'ordonnancement de base en cas de perturbations majeures ont été largement étudiées sous la terminologie d'ordonnancement proactif-réactif [Van de Vonder 2007].

Récemment, [Davari 2017] a remarqué que dans cette série d'approches, les phases proactives et réactives étaient plutôt traitées séparément alors qu'elles devraient s'influencer mutuellement. Ainsi, dans [Davari 2017, Davari 2019], les auteurs proposent une approche proactive-réactive intégrée où ils visent à trouver la meilleure politique, qui est dans leur cas un planning initial robuste et un ensemble de réactions donnant des transitions d'un planning à un autre en réponse à une perturbation, étant donné un certain coût de réaction.

Dans un travail pionnier, [Drummond 1994] a proposé l'approche dite *Just In Case*, dans laquelle est calculé un ordonnancement contingent multiple, où les transitions d'un ordonnancement de base vers des ordonnancements alternatifs sont anticipées à certains événements ayant une forte probabilité de perturbation. Cette approche a été depuis largement développée pour les problèmes de planification en IA, abordant entre autres les problèmes d'incrémentalité et de limites de mémoire. [Dearden 2003, Meuleau 2002].

**RCPSP robuste.** Dans cette section, on s'intéresse plus particulièrement à ce qui existe dans la littérature concernant la résolution du RCPSP robuste. On peut commencer par rappeler qu'il existe plusieurs critères de robustesse.

Par exemple, le critère du min-max regret, qui est considéré dans [Artigues 2013b], où les auteurs proposent deux algorithmes pour résoudre le problème RCPSP robuste avec ce critère, tous deux basés sur de la relaxation de scénarios proposée dans [Assavapokee 2008]. Dans [Mogaadi 2016], les auteurs étudient, sur la base de scénarios, deux modèles robustes, le modèle min-max qui se concentre sur la minimisation de l'objectif de robustesse absolue et le modèle min-max regret dont l'objet est de minimiser le regret

absolu. Ils proposent une approche génétique adaptative et robuste avec une population initiale sophistiquée et une heuristique d'amélioration Forward-Backward. L'algorithme proposé est appliqué à l'ensemble de données PSPLIB J30 avec des durées d'activité modifiées. Les résultats obtenus montrent la performance de l'algorithme génétique combiné avec l'heuristique d'amélioration par rapport à la version de base. Différents niveaux de perturbation ont été testés pour déterminer la dégradation de performance correspondante.

Dans [Bruni 2018], les auteurs étudient le problème robuste d'ordonnancement de projets sous contrainte de ressources à deux niveaux, dans le cadre d'un polytope d'incertitudes budgétisé, où l'on cherche une fois de plus à minimiser le makespan dans le pire des cas, en supposant que les durées des activités sont sujettes à une incertitude d'intervalle. Le premier niveau de décision attribue les ressources aux différentes activités du projet, alors que les variables du second niveau décident des temps de départ de ces activités. Le modèle permet de contrôler le niveau de robustesse au moyen d'un facteur de protection lié à l'aversion au risque du décideur. Comme on l'a vu plus haut, ce genre de problème est difficile. Pourtant les auteurs présentent deux approches de décomposition exacte. Une expérimentation extensive, sur des instances de référence standards de la littérature, est réalisée pour évaluer et comparer la performance des méthodes proposées, également par rapport à l'approche de solution exacte de l'état de l'art. Une autre façon d'approcher la robustesse est de la considérer dans une approche multi-objectif, où l'idée est de trouver un compromis entre le pessimisme inhérent à la robustesse et la qualité du planning calculé.

Dans [Al-Fawzan 2005], les auteurs discutent de la question de la conception d'un calendrier de projet qui est non seulement court dans le temps, mais aussi moins vulnérable aux perturbations dues aux retouches et autres conditions indésirables. À cette fin, ils introduisent le concept de robustesse du calendrier et développent un modèle de planification de projet bi-objectif avec des contraintes de ressources. Ces deux objectifs sont d'abord de maximisation de la robustesse, et de minimisation du makespan. Ils proposent un algorithme de recherche tabou afin de générer un ensemble approximatif de solutions efficaces. Plusieurs variantes de l'algorithme sont testées et comparées sur un large ensemble de problèmes de référence. Dans la même idée d'approcher la robustesse par une approche multi-objectif, on peut noter [Palacio 2017] dans lequel les auteurs proposent une approche exacte basée sur deux modèles de programmation linéaire en nombres entiers mixtes (MILP) pour résoudre le RCPSP. Le premier MILP vise à minimiser le makespan, tandis que le second MILP maximise la robustesse du planning. Les formulations mathématiques sont résolues en utilisant une approche lexicographique. Ils illustrent l'efficacité des modèles proposés en résolvant des instances standards pour le RCPSP disponibles dans la bibliothèque de problèmes d'ordonnancement de projets (PSLIB). Les résultats des calculs montrent qu'il est possible de trouver des solutions optimales alternatives avec une robustesse maximale et un makespan minimal pour des instances comportant jusqu'à 90 activités. De plus, puisque le problème du RCPSP sous incertitude est un problème difficile, beaucoup d'efforts ont été faits pour trouver des heuristiques efficaces. Dans [Chtourou 2008], les auteurs présentent un algorithme en deux étapes pour l'ordonnancement robuste de projets sous contrainte de ressources. La première étape de l'algorithme résout le RCPSP pour minimiser le makespan en utilisant uniquement une heuristique basée sur des règles de priorité, à savoir un schéma amélioré de génération d'ordonnancement en série avec une composante aléatoire.

L'idée d'un tel schéma est de générer un ordonnancement réalisable en insérant successivement au plus tôt les tâches dans le planning de sorte à respecter les contraintes de ressources et de précédence. La partie "aléatoire" de leur méthode vient de l'ordre dans lequel on regarde les tâches à insérer. Les tâches sont considérées dans un ordre compatible avec les contraintes de précédence, et lorsque plusieurs tâches sont indistinguables de ce point de vue, elles sont sélectionnées dans un ordre aléatoire. Le problème est ensuite résolu de manière similaire pour maximiser la robustesse de l'ordonnancement tout en considérant le makespan obtenu dans la première étape comme un seuil d'acceptation. La sélection du meilleur ordonnancement dans cette phase se base sur l'un des 12 indicateurs prédictifs de robustesse alternatifs formulés dans le but de maximiser la robustesse. Des tests de simulation approfondis des planifications générées démontrent clairement les avantages de la prise en compte de la robustesse des planifications en plus du makespan. À titre d'illustration, pour 10 problèmes de l'ensemble standard bien connu $J30$, les planifications robustes et non robustes sont exécutées avec une augmentation de 10% de la durée appliquée aux mêmes 20% des activités du projet choisies au hasard. Sur 1000 itérations par instance du problème, les planifications robustes affichent un makespan plus court dans 55% des cas, tandis que les planifications non robustes ne sont les plus performantes que dans 6% des cas.

## 1.3 Exemple, PERT Généralisé

Dans cette dernière section, nous présentons une première contribution [1] sur des généralisations des méthodes PERT, dans le but de les rendre pertinentes dans le cadre de problème d'ordonnancement de projet sous incertitude. Un algorithme pseudo-polynomial pour résoudre le PERT robuste avec budget d'incertitude, au sens où l'on cherche la date de fin minimale du projet réalisable compte tenu des incertitudes, a été proposé dans [Minoux 2007]. Nous nous intéressons ici plutôt à des analyses de criticité des tâches lorsque leurs durées sont définies par des intervalles. Pour le PERT standard, un état de l'art des résultats connus sur ces analyses de criticité est présenté dans [Fortin 2010].

### 1.3.1 Cas déterministe

Un graphe d'activités est défini par un graphe $G(V,A)$ où $V = \{1,...,n\}$ est un ensemble de sommets, qui représentent les activités -ou tâches- du projet, et $A \subseteq V \times V$ est un ensemble d'arcs, qui représentent les contraintes de précédence entre les activités. Chaque arc $(i,j) \in A$ est pondéré par une valeur qui correspond à la durée de la tâche $i$. On suppose que le graphe contient deux sommets, $s = 1$ et $t = n$ qui représentent deux tâches fictives, de durées nulles, et de sorte que l'activité $s$ précède toutes les autres, alors que toutes les activités précèdent la tâche $t$. Comme on sera amené à travailler avec des chemins dans ce graphe, on introduit quelques notations supplémentaires. Un chemin $p$ dans $G$ est un ensemble ordonné d'activités. On désigne par $\mathscr{P}$ l'ensemble de tous les chemins possibles dans $G$, et par $\mathscr{P}_i$ l'ensemble de tous les chemins $p$ de $G$ tels que le noeud $i$ appartient au

1. Il s'agit d'un travail en cours, en collaboration avec Cyril Briand

chemin $p$. On définit également $\mathscr{P}_i^+$ (resp. $\mathscr{P}_i^-$) l'ensemble des chemins commençant par $i$ (resp. finissant par $i$), et $\mathscr{P}_{i,j}$ l'ensemble des chemins de $i$ à $j$. Aussi on note $l_p$ la longueur d'un chemin $p$ et $l^*_{i,j}$ la longueur du plus long chemin de $i$ à $j$.

On s'intéresse au calcul de trois quantités particulières, liées à la définition de ce graphe.

La première est la date de départ au plus tôt -*earliest starting time* - d'une tâche $i$, que l'on note $est_i$. On peut calculer cette valeur grâce au graphe d'activités, puisqu'elle est égale à la longueur du plus long chemin entre $s$ et $i$. Plus formellement :

$$est_i = \max_{p \in \mathscr{P}_i^-} l_p$$

Cette date correspond à la date minimale à laquelle une tâche peut être commencée, pour peu qu'aucune tâche qui ne la precède n'ait été retardée.

De façon similaire, on peut calculer la date de départ au plus tard -*latest starting time*- d'une tâche grâce au graphe d'activités, car elle est égale à la différence entre la valeur du plus long chemin dans le graphe (qui correspond à la durée totale du projet) et la valeur du plus long chemin dans le graphe passant par $i$. On peut écrire cela comme suit :

$$lst_i = \max_{p \in \mathscr{P}} l_p - \max_{p \in \mathscr{P}_i^+} l_p$$

Cette date correspond à la date maximale à laquelle une tâche peut être démarrée, sans qu'elle n'augmente la durée de complétion du projet.
On définit enfin la marge d'une activité -*float*- comme étant la différence entre sa date de départ au plus tard et sa date de départ au plus tôt. Ou encore :

$$f_i = lst_i - est_i = \max_{p \in \mathscr{P}} l_p - \max_{p \in \mathscr{P}_i} l_p$$

Lorsque cette valeur vaut 0 pour une tâche donnée, on dit que cette tâche est critique, en ce sens que si elle retardée pour une raison quelconque, la durée totale du projet en sera impactée. On peut remarquer que par définition toutes les tâches situées sur le plus long chemin de $G$ sont critiques.
Puisque $G$ est un graphe orienté et sans cycle, ces trois quantités peuvent être calculées en temps polynomial.

### 1.3.2 Cas incertain

On suppose maintenant que la durée des tâches est incertaine, et est comprise dans un intervalle. Plus formellement, si on note $d_i$ la durée de la tâche $i$, on suppose que $d_i$ prend sa valeur dans un intervalle $[d_i^-, d_i^+]$.

Un scénario $d \in D = \{(d_i)_{i \leq n}\}$ est une instanciation de toutes les durées $d_i$. On dit que $d$ est un scénario *extrême* si pour chaque $i \leq n$, $d_i \in \{d_i^-, d_i^+\}$. On note $D^{ext}$ l'ensemble de tous les scénarios extrêmes. $d^+$ (resp. $d^-$) désigne le scénario dans lequel tous les $d_i$ sont à leur valeur maximale $d_i^+$ (resp. minimale $d_i^-$). On note $l_p(d)$ la longueur du chemin $p$ dans le scénario $d$ et $l^*_{i,j}(d)$ la longueur du plus long chemin entre $i$ et $j$ dans le scénario $d$.

On peut maintenant définir les dates de départ au plus, plus tard et la marge d'une activité $i$ en fonction d'un scénario $d$ :

— Date de départ au plus tôt :

$$est_i(d) = \max_{p \in \mathscr{P}_i^-} l_p(d)$$

— Date de départ au plus tard :

$$lst_i(d) = \max_{p \in \mathscr{P}} l_p(d) - \max_{p \in \mathscr{P}_i^+} l_p(d)$$

— Marge :

$$f_i(d) = lst_i(d) - est_i(d) = \max_{p \in \mathscr{P}} l_p(d) - \max_{p \in \mathscr{P}_i} l_p(d)$$

Pour étendre la notion de criticité d'une activité dans le cas incertain, on distingue deux cas particuliers. On dit qu'une tâche $i$ est nécessairement critique si dans le scénario qui maximise la plus grande marge, que l'on note $f_i^{max} = \max_{d \in D} f_i(d)$ sa marge vaut 0. Cela signifie en effet que dans tous les scénarios la tâche est critique. De la même manière, on dit qu'une tâche est possiblement critique si dans le scénario qui minimise sa plus petite marge, que l'on note $f_i^{min} = \min_{d \in D} f_i(d)$, cette tâche est critique. On peut étendre de façon similaire les dates de départ au plus tôt et plus tard en définissant $est_i^{max} = \max_{d \in D} est_i(d)$, $est_i^{min} = \min_{d \in D} est_i(d)$ et $lst_i^{max} = \max_{d \in D} lst_i(d)$, $lst_i^{min} = \min_{d \in D} lst_i(d)$.

Il est tentant de penser, étant donnée la définition de la marge que $f_i^{max} = lst_i^{max} - est_i^{max}$ ou encore que $f_i^{max} = lst_i^{max} - est_i^{min}$. C'est faux en général. On peut s'en convaincre facilement en remarquant que le scénario qui minimise ou maximise la date de départ au plus tôt n'a aucune raison d'être le même que celui qui maximise ou minimise la date de départ au plus tôt.

En termes de complexité, il a été montré les résultats suivants [Fortin 2010] :

| Fonction | $est^{min}$ | $est^{max}$ | $lst^{min}$ | $lst^{max}$ | $f^{min}$ | $f^{max}$ |
|---|---|---|---|---|---|---|
| Complexité | P | P | P | P | NP | P |

On voit dans le tableau que seul le calcul de la marge minimale d'une tâche est NP-Difficile, les autres valeurs pouvant être calculées en temps polynomial.

### 1.3.3 Cas incertain avec contraintes de précédence généralisées

On s'intéresse maintenant à une extension où l'on suppose que les tâches sont liées par des contraintes de précédence généralisées. Ces contraintes peuvent forcer des temps minimaux ou maximaux entre les temps de départs des tâches, et sont courantes dans les problèmes industriels. Plus formellement, pour chaque paire de tâche $(i, j)$, il existe une durée $d_{i,j}$ telle que si l'on note $S_i$ la date de départ de la tâche $i$, alors $S_j - S_i \geq d_{i,j}$ En prenant compte de cela, le graphe d'activités change un peu de forme comparativement aux cas précédent, puisqu'il n'est maintenant plus acyclique. On suppose toujours que ces délais $d_{ij}$ entre les tâches sont incertains et prennent leur valeur dans l'intervalle $[d_{i,j}^-, d_{i,j}^+]$.

Toutes les notations introduites dans la section précédente sont conservées, en remplaçant $i \leq n$ par $(i, j) \in A$ et $d_i$ par $d_{i,j}$. On supposera également que tous les scénarios sont cohérents, dans le sens où il n'existe pas de scénario pour lequel les délais entre les tâches créent des contraintes impossibles à respecter. Cela se traduit dans le graphe par le fait qu'il n'existe pas de cycle de longueur positive ou nulle.

Nous avons cherché à déterminer les classes de complexité du calcul des six quantités introduites dans la section précédente, et avons conclu pour trois d'entre elles.

**Théorème 1.** *Le temps de calcul de $est_i^{max}$ et $est_i^{min}$ est polynomial pour toute activité i.*

*Démonstration.* Il est clair que $est_i^{min}$ (resp. $est_i^{max}$) est atteint pour le scénarios $d^-$ (resp. $d^+$), nous allons donc maintenant considérer notre graphe $G$ sous l'un de ces scénarios, selon la quantité que nous souhaitons calculer. Considérons maintenant le graphe $G'(V,A)$ avec des poids $d'_{i,j} = -d_{i,j}$. Ce graphe n'a pas de cycle orienté négatif, et l'algorithme de Bellman-Ford donne exactement les valeurs souhaitées pour chaque activité.

□

Nous nous intéressons maintenant au calcul de la date de départ au plus tard minimale. Pour tout chemin $p$ de $G$ on note $d^{max}(p)$ le scénario tel que $d_{i,j} = d^+_{i,j}$ si l'arc $(i, j)$ est dans le chemin $p$ et $d_{i,j} = d^-_{i,j}$ sinon.

**Lemme 1.** *Pour chaque activité $i \in V$, il existe un chemin $p_i$ de i vers n tel que :*

$$\min_{d \in D} lst_i(d) = lst_i(d^{max}(p_i))$$

*Preuve*. Soit $d^*$ tel que $lst_i(d^*) = \min_{d \in D} lst_i(d)$ et $p$ l'un des plus longs chemins de $i$ à $n$ sous le scénario $d^*$.

Passons de $d^*$ à $d^{max}(p)$ en augmentant les délais $d_{j,k}$ à $d^+_{j,k}$ si $j$ et $k$ sont consécutifs dans $p$, ou en les diminuant à $d^-_{j,k}$ sinon. Dans ce nouveau scénario, $p$ est toujours un des plus longs chemins de $i$ à $n$ et sa longueur a augmenté de $\delta = \sum_{(j,k) \in p} (d^{max}_{j,k}(p) - d^*_{j,k})$.

De plus, puisqu'il n'existe aucun circuit de longueur positive, la longueur de tout chemin potentiel le plus long ne peut avoir été augmentée de plus de $\delta$. Si on note $L$ la longueur du plus long chemin dans le scénario $d^*$ , $L'$ celle dans le scénario $d^{max}(p)$ et $\delta'$ tel que $L + \delta' = L'$ et $\delta' \leq \delta$. La date de départ au plus tard de l'activité $i$ sur le scénario $d^{max}(p)$ est :

$$\begin{aligned} lst_i(d^{max}(p)) &= L' - l_p(d^{max}(p)) \\ &= L + \delta' - (l_p(d^*) + \delta) \\ &= lst_i(d^*) + (\delta' - \delta) \end{aligned}$$

Puisque $\delta' \leq \delta$, $lst_i(d^{max}(p)) \leq lst_i(d^*)$, il en résulte que $lst_i(d^{max}(p)) = lst_i(d^*) = \min_{d \in D} lst_i(d)$. □

**Théorème 2.** *Soit i une activité. On a alors :*

$$lst_i^{min} = \min_{j \in succ(i)} lst_i(d^{max}((i) \cup p_j))$$

*où $p_j$ est un chemin issu du lemme précédent appliqué à une activité $j$.*

*Preuve*. D'après lemme précédent, on sait qu'il existe un chemin $p_i$ de $i$ vers $n$ de sorte que $lst_i^{min} = lst_i(d^{max}(p_i))$. Soit $j$ le successeur de $i$ dans $p_i$, on a :

$$\begin{aligned} lst_j(d^{max}(p_i)) &= l^*_{1,n}(d^{max}(p_i)) - l^*_{j,n}(d^{max}(p_i)) \\ &= l^*_{1,n}(d^{max}(p_i)) - l_{p_i}(d^{max}(p_i)) + d^+_{i,j} \\ &= lst_i^{min} + d^+_{i,j} \end{aligned} \tag{1.12}$$

Maintenant, soit $p_j$ un chemin issu du lemme précédent en l'appliquant à l'activité $j$. Comme $j \in p_i$ qui est un plus long chemin dans $d^{max}(p_i)$ et que le graphe d'activités ne contient pas de cycle positif on peut remarquer que :

$$\begin{aligned} l^*_{i,n}(d^{max}((i)\cup p_j)) &\geq l^*_{j,n}(d^{max}((i)\cup p_j)) + d^+_{i,j} \\ l^*_{1,n}(d^{max}((i)\cup p_j)) - l^*_{i,n}(d^{max}((i)\cup p_j)) &\leq l^*_{1,n}(d^{max}((i)\cup p_j)) - l^*_{j,n}(d^{max}((i)\cup p_j)) - d^+_{i,j} \\ lst_i(d^{max}((i)\cup p_j)) &\leq l^*_{1,n}(d^{max}((i)\cup p_j)) - l^*_{p_j}(d^{max}((i)\cup p_j)) - d^+_{i,j} \end{aligned} \tag{1.13}$$

Supposons par l'absurde que $lst_i^{min} < lst_i(d^{max}((i)\cup p_j)$.
A partir des équations (1.12) et (1.13) on peut écrire :

$$lst_j(d^{max}(p_i)) < l^*_{1,n}(d^{max}((i)\cup p_j)) - l_{p_j}(d^{max}((i)\cup p_j)) \tag{1.14}$$

Soit $p'$ un plus long chemin dans $G$ de 1 à $n$ pour le scénario $d^{max}((i)\cup p_j)$.
On doit distinguer deux cas :

— $i \notin p'$ : alors $l^*_{1,n}(d^{max}((i)\cup p_j)) = l^*_{1,n}(d^{max}(p_j))$ et en l'injectant dans l'équation (1.14) on obtient $lst_j(d^{max}(p_i)) < lst_j^{min}$, ce qui contredit la définition de $lst_j^{min}$.

— $i \in p'$ alors l'activité $i$ est critique pour $d^{max}((i)\cup p_j)$, ce qui implique $lst_i(d^{max}((i)\cup p_j)) = lst_i(d^{max}(i)\cup p_j)) = est_i(d^{max}((i)\cup p_j)) = l_{1,i}(d^{max}((i)\cup p_j))$. Ensuite tous les arcs du chemin $p'$ avant $i$ sont à la valeur minimale. Si ça n'était pas le cas, cela signifierait qu'un de ces arcs appartiendrait à $p_j$, mais comme $p'$ est un plus long chemin ; $p_j$ contiendrait un cycle, ce qui est impossible puisque tous les cycles sont de longueur négative ou nulle. Ainsi $lst_i(d^{max}((i)\cup p_j)) = l_{1,i}(d^{max}((i)\cup p_j)) = est_i^{min} \leq lst_i^{min}$, ce qui contredit l'hypothèse.

Ainsi, en raisonnant par l'absurde on en déduit que $lst_i^{min} = lst_i(d^{max}((i)\cup p_j)$.
On conclut la démonstration en remarquant qu'une telle activité $j$ appartient à $succ(i)$. □

Cette preuve ressemble à celle de [Fortin 2010] à propos du PERT dans le cas incertain, mais nous avons considéré les cas où la présence de cycle dans le graphe rend certains arguments moins évidents. Grâce à ce théorème on peut, par programmation dynamique à partir de la fin du graphe d'activités, calculer les valeurs $lst^{min}$ pour toutes les tâches du projet.

Finalement, A partir des deux théorèmes précédents, et remarquant que les contraintes

de précédence classiques sont un cas particulier des contraintes de précédence généralisées, nous pouvons en déduire les résultats suivants :

| Fonction | $est^{min}$ | $est^{max}$ | $lst^{min}$ | $lst^{max}$ | $f^{min}$ | $f^{max}$ |
|---|---|---|---|---|---|---|
| Complexité | P | P | P | ? | NP | ? |

Malheureusement nous ne sommes pas parvenus à conclure quant aux complexités des problèmes de calcul de $lst^{max}$ et $f^{max}$.

CHAPITRE 2

# Méthode à voisinage étendu pour l'ordonnancement de projet à modes multiples déterministe

**Sommaire**



Les travaux présentés dans ce chapitre ont mené à un article en révision dans *Computers and Industrial Engineering* [Borreguero 2021], co-écrit avec Tamara Borreguero Sanchidrián, alors doctorante chez Airbus Madrid et à l'université polytechnique de Madrid. Seules mes contributions apparaissent dans ce chapitre. Ces travaux ont obtenu le 2ème prix au concours du meilleur papier étudiant à la conférence PMS 2022 [Portoleau 2022a].

## 2.1 Introduction

L'industrie aéronautique a connu une profonde transformation au cours des dernières années. La demande d'avions a augmenté, mais aussi la complexité et la personnalisation des produits. En conséquence, les fabricants d'avions ont dû produire davantage d'unités d'avions plus complexes tout en essayant de réduire les délais de mise sur le marché,

les délais de production et les coûts [Mas 2015]. Pour faire face à ces défis et à d'autres, l'industrie aéronautique s'est orientée vers la mise en œuvre des tendances de l'industrie 4.0 [Kagermann 2013]. Comme dans d'autres industries, la mondialisation et la numérisation sont devenues des caractéristiques centrales. En réalité, l'industrie aéronautique a été en avance dans l'utilisation de solutions numériques. Un large éventail d'outils de gestion du cycle de vie des produits a été déployé dans l'industrie aéronautique et a donné lieu à des processus hautement numérisés, de la conception à la maintenance des aéronefs [Mas 2014]. Cependant, les processus de planification et d'ordonnancement sont restés pratiquement inchangés. La plupart des activités liées à ces processus continuent d'utiliser des procédures manuelles qui reposent sur les connaissances d'experts.

Pourtant, la numérisation des processus d'ordonnancement est nécessaire non seulement pour la mise en œuvre du système d'exécution de la fabrication, mais aussi pour le déploiement de l'industrie 4.0. [Schläpfer 2014] énumère trois objectifs pour l'industrie 4.0, qui ne peuvent être atteints sans améliorer les processus de planification et d'ordonnancement :

— Une plus grande flexibilité grâce à une planification, un contrôle et une exécution dynamiques ;
— Une meilleure productivité et efficacité des ressources pour les produits personnalisés ;
— Des délais plus courts grâce à l'application d'analyses intelligentes.

En général, lorsque l'on cherche à améliorer de cette manière des lignes d'assemblage industrielles, on se retrouve avec deux types de problèmes distincts. Les premiers sont les problèmes d'équilibrage de lignes d'assemblage (ou *assembly line balancing*) dont le principe est le suivant. Le problème prend en entrée un ensemble de tâches -soumises à des contraintes de précédence- à effectuer sur des stations de travail. Il y a alors deux objectifs classiques possibles.

1. S'il y a un temps de cycle fixé, c'est-à-dire un horizon temporel fixé, la question est : Comment minimiser le nombre de stations de travail dont on a besoin pour le respecter ?
2. S'il y a un nombre de stations de travail disponibles fixé, la question est : Quel est le plus petit temps de cycle réalisable avec cet ensemble de stations ?

On ne détaillera pas plus ces problèmes d'équilibrage dans cette thèse. Pour une revue de littérature relativement récente sur le sujet, le lecteur peut se référer à [Sivasankaran 2014]. L'autre type de problèmes, celui auquel on va s'intéresser dans ce chapitre, concerne l'ordonnancement de lignes d'assemblage. Avant de présenter les problèmes d'ordonnancement, il faut noter qu'une tentative a été faite pour intégrer les deux problèmes en un seul. Par exemple, dans [Andres 2008] et [Shahanaghi 2010] les auteurs considèrent des problèmes où la durée des tâches dépend de la façon dont elles sont ordonnancées, avec respectivement un temps de préparation dépendant de la séquence pour le premier, et la prise en compte d'un effet de détérioration pour le second. Une approche plus concrète, plutôt orientée vers l'ingénierie, a été proposée dans [Borreguero 2015] où les auteurs développent une méthodologie pour les lignes d'assemblage d'Airbus afin que leurs outils intègrent les aspects des deux problèmes précédents.

En ce qui concerne l'aspect de l'ordonnancement sur les lignes d'assemblage, on trouve moins d'occurrence de ce problème dans la littérature, puisque ces problèmes sont souvent traités sous le prisme plus générique des problèmes d'ordonnancement. Cependant plusieurs techniques ont été développées pour le cas spécifique des lignes d'assemblage. Dans [Sabuncuoğlu 2008], les auteurs proposent d'améliorer la méthode "Beam Search", une heuristique basée sur l'algorithme Branch and Bound, puis l'appliquent à un problème d'ordonnancement de lignes d'assemblage. Dans [Zhang 2000], les auteurs présentent un problème industriel issu des chaînes de montage du constructeur automobile Toshiba, et proposent de le résoudre par relaxation Lagrangienne en tirant parti des caractéristiques géométriques de leur problème.

Pour répondre aux trois objectifs cités plus haut de l'industrie 4.0, ce chapitre propose une heuristique compétitive pour l'ordonnancement d'une chaîne d'assemblage final d'avions. L'étude de cas industriel considéré fourni par Airbus est la ligne d'assemblage finale de l'A330.

Ce problème peut être classé dans la catégorie des problèmes d'ordonnancement de projets sous contraintes de ressources (RCPSP) et plus particulièrement des problèmes d'ordonnancement de projets sous contraintes de ressources (MMRCPSP) avec un objectif de lissage des ressources (voir §1.1.3).

Le contexte industriel et une présentation informelle du problème sont donnés dans la section 2.2. Une présentation plus formelle du problème de MMRCPSP considéré ainsi qu'une formulation de programmation par contraintes sont décrites dans la section 2.3. La méthode de voisinage étendu que nous proposons pour ce problème est détaillée dans 2.4. La section 2.5 présente des résultats expérimentaux sur des problèmes industriels posés par Airbus. Une comparaison avec des travaux précédents sur un problème de MMRCPSP de la littérature est effectuée dans la section 2.6. Pour finir, des conclusions et perspectives sont brièvement évoquées dans la section 2.7.

## 2.2 Contexte industriel et présentation informelle du problème

Les chaînes d'assemblage aéronautiques sont constituées d'une série de stations où différentes tâches sont exécutées. La décomposition en tâches est telle que chaque produit doit passer par toutes les stations en suivant un chemin fixe. En outre, la ligne est souvent synchronisée, ce qui signifie que le temps pendant lequel chaque produit reste sur une station est toujours le même, et est égal à la vitesse globale de la ligne d'assemblage. Dans ce contexte, nous considérons le temps de cycle comme le temps nécessaire à un avion pour visiter chacune des stations. Dans notre problème, le temps de cycle maximum est fixé.

L'affectation des tâches à une station est dans la plupart des cas liée à des questions industrielles. Par exemple, cela peut-être du fait des technologies d'assemblage ou encore le besoin de gabarits spécifiques qui ne peuvent pas être facilement déplacés. Par conséquent, l'affectation des tâches entre les stations n'est effectuée qu'une seule fois, lors de la définition de la ligne. Bien qu'il puisse y avoir un petit pourcentage de tâches qui peuvent être exécutées dans plus d'une station, une fois que les tâches sont assignées, elles sont rarement déplacées d'une station à l'autre, donc on supposera sans perte de généralité que

l'affectation des tâches est fixée par station.

En tenant compte de cela, la décision d'ordonnancement consiste à établir l'ordre dans lequel les tâches seront effectuées ainsi que les ressources allouées à chacune d'entre elles, étant donné le temps de cycle de la ligne et un ensemble de tâches par station. Le résultat est généralement affiché sous la forme d'un diagramme de Gantt (figure 2.1) où chaque tâche se voit attribuer une date de début/fin et un ensemble d'opérateurs. Du point de vue de l'opérateur, on utilise une vue de suivi des performances (figure 2.2). La principale différence est que le traqueur de performance, c'est-à-dire l'outil qui permet de suivre le déroulement des activités, est orienté vers le travailleur, de sorte que chaque ligne fournit des informations sur la tâche à effectuer par un opérateur tout au long du temps de cycle. L'accès à ces informations sera une propriété clef dans les prochains chapitres de ce manuscrit.

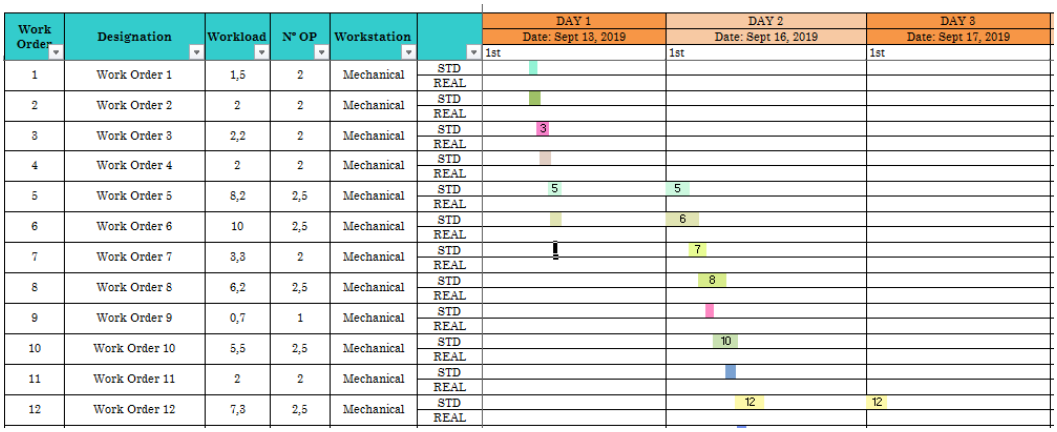



FIGURE 2.1 – Diagramme de Gantt

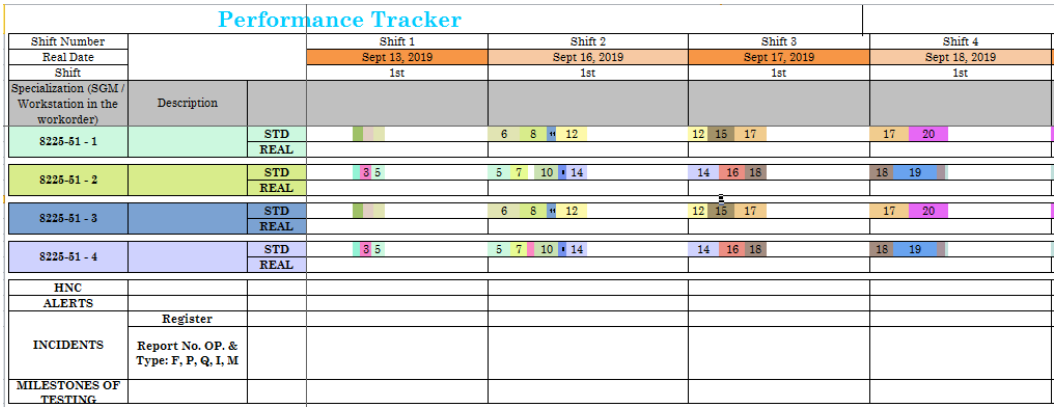



FIGURE 2.2 – Exemple de vue de suivi de performances

Ces dernières années, l'automatisation des stations aéronautiques a connu d'importantes améliorations. Malgré cela, le montage aéronautique reste très demandeur en opérateurs hautement qualifiés. Souvent, certaines tâches doivent être effectuées par des tra-

vailleurs ayant une compétence spécifique, et tous n'ont pas les mêmes compétences. Ces différentes compétences sont gérées par l'utilisation de 'profils' qui regroupent une ou plusieurs compétences. Chaque opérateur se voit attribuer un profil, en fonction de ses compétences. De plus, chaque tâche peut être effectuée par des opérateurs ayant un ou plusieurs profils. Par exemple, si profil 1 ne comprend que des tâches mécaniques élémentaires (comme le perçage et le rivetage) et profil 2 comprend les précédentes ainsi que l'installation de tuyaux, un travail impliquant le rivetage et le perçage peut être effectué l'un ou l'autre des deux profils.

Les contraintes entre les tâches peuvent être de natures différentes. Le cas le plus courant est celui des contraintes de précédence : lorsqu'une tâche ne peut être commencée avant qu'une précédente ne soit terminée. Par exemple : les harnais ne peuvent pas être acheminés tant que les supports auxquels ils sont attachés n'ont pas été rivetés à la structure de l'avion, ou un test fonctionnel ne peut pas être effectué tant que le système qu'il teste n'a pas été complètement installé.

Un autre type de contraintes est celui des contraintes d'incompatibilité ou disjonctives : elles signifient que certaines activités peuvent être effectuées dans n'importe quel ordre, à condition qu'elles ne soient pas exécutées en même temps. C'est le cas de certaines tâches qui, pour des raisons de santé et de sécurité, doivent être effectuées avec le moins de personnes possible dans le hangar, par exemple l'application d'inhibiteurs de corrosion. Cela se produit également lors d'essais qui nécessitent un état spécifique de l'avion : les essais hydrauliques doivent être effectués sous tension, mais l'avion doit être hors tension pour les essais de carburant.

Enfin, des décalages maximaux -on parle aussi de contraintes de précédence généralisées- peuvent également intervenir. Par exemple, les tests de collage doivent être effectués à la fin des installations, et la protection par collage doit être réalisée dans le même jour ouvrable où le test a été réussi. Dans certains cas, il faut non seulement un délai maximal mais aussi la même équipe de travail.

En accord avec le temps de cycle fixe, la fonction objectif sera de minimiser la consommation de ressources par station. Comme nous l'avons dit, cela revient à minimiser la somme des pics de demande des opérateurs par profil, c'est-à-dire le nivellement des ressources.

Plus succinctement, ce problème sera modélisé par un problème d'ordonnancement de projet avec contraintes de ressources et multi-mode, ou MMRCPSP, avec des contraintes de précédence généralisées, des contraintes disjonctives et un objectif de nivellement des ressources. Ce problème est connu pour être NP-Difficile [Kolisch 1997a].

## 2.3 Présentation formelle et formulation de programmation par contraintes

### 2.3.1 Notations et présentation formelle

Nous avons un ensemble de tâches $\mathscr{I}$ et un ensemble de ressources $\mathscr{R}$ décomposé en deux sous ensembles : l'ensemble des opérateurs $\mathscr{R}^O$ regroupés en profils, de sorte que

$o \in \mathscr{R}^O$ correspond à un profil particulier d'opérateurs, et l'ensemble des ressources cumulatives $\mathscr{R}^A$ qui représente généralement des zones à capacité limitée en termes de tâches qui peuvent y être réalisées. Ainsi $C_a$ désigne la capacité de la zone/ressource cumulative $a \in \mathscr{R}^A$. Une tâche $i \in \mathscr{I}$ peut être exécutée selon plusieurs modes $\mathscr{M}_i$. Un mode $m \in \mathscr{M}_i$ détermine la durée de la tâche $p_{i,m}$, ainsi que la quantité d'opérateurs utilisée $b_{i,m,o}, \forall \in \mathscr{R}^O$ et l'occupation $b_{i,m,a}$ de chaque zone $a \in \mathscr{R}^A$. Par ailleurs nous notons *Prec* l'ensemble des contraintes de précédence classiques et $Prec_G$ l'ensemble des contraintes de précédence généralisées qui dans l'étude de cas industriel sont associées à un délai maximal fixe $\Delta$. Nous notons $\mathscr{D}$ l'ensemble des paires de tâches incompatibles, ou disjonctives, et $CT$ le temps de cycle à respecter.

Etant donné ces notations, le problème peut être formulé comme suit :

$$\min \quad z = \sum_{o \in \mathscr{R}^O} \max_{t=0,\ldots,CT} B_{o,t} \tag{2.1}$$

$$\text{s. t.} \quad S_j - S_i \geq p_{i,m_i} \qquad \forall (i,j) \in Prec \tag{2.2}$$

$$S_j - S_i \leq \Delta \qquad \forall (i,j) \in Prec_G \tag{2.3}$$

$$S_j - S_i \geq p_{i,m_i} \vee S_i - S_j \geq p_{j,m_j} \qquad \forall i,j \in \mathscr{D} \tag{2.4}$$

$$B_{k,t} = \sum_{i \in \mathscr{I}, t \in [S_i, S_i + p_{i,m_i} - 1]} b_{i,k,m_i} \qquad \forall k \in \mathscr{R}, \forall t = 0,\ldots,CT \tag{2.5}$$

$$B_{a,t} \leq C_a \qquad \forall a \in \mathscr{R}^A, \forall t = 0,\ldots,CT \tag{2.6}$$

$$S_i \in [0,\ldots,CT[ \qquad \forall i \in \mathscr{I} \tag{2.7}$$

$$m_i \in \mathscr{M}_i \qquad \forall i \in \mathscr{I} \tag{2.8}$$

Les variables de décision sont les dates de début des tâches $S_i$, le mode $m_i$ utilisé pour chaque tâche et la quantité $B_{k,t}$ de chaque ressource $k$ utilisée à la période de temps $t$. L'objectif (2.1) consiste en la minimisation de la somme des pics d'utilisation de chaque profil d'opérateurs. Les contraintes (2.2) représentent les contraintes de précédence début/fin classiques alors que les contraintes (2.3) donnent les contraintes de décalages maximaux. Les contraintes (2.4) expriment l'impossibilité d'exécuter simultanément deux tâches incompatibles. Les quantités $B_{k,t}$ sont exprimées par les contraintes (2.5) comme la somme des quantités utilisées par les tâches en cours d'exécution pendant la période $t$. Pour les ressources cumulatives/zones cette quantité est bornée par la capacité via les contraintes (2.6). Les domaines de définition des variables $S_i$ et $m_i$ sont enfin données par les contraintes (2.7) et (2.8). Le problème est évidemment NP-difficile puisque sa version décisionnelle admet la version décisionnelle du RCPSP comme cas particulier.

Dans l'article coécrit avec Tamara Borreguero Sanchidrián ([Borreguero 2021]) plusieurs formulations en programmation linéaire en nombres entiers (PLNE) sont proposées pour résoudre ce problème. Cependant, les modèles obtenus n'ont pu être utilisés que pour résoudre de petites instances. La Programmation par Contraintes (PPC) s'est avérée au contraire être une méthode efficace comme le montrent les résultats expérimentaux, et nous donnons ci-après une formulation PPC.

### 2.3.2 Formulation de programmation par contraintes

Nous proposons donc un modèle de programmation par contraintes basé sur des contraintes d'ordonnancement standard. Pour la modélisation et la résolution, nous utilisons la bibliothèque d'ordonnancement par contraintes CP Optimizer. Nous faisons référence ci-dessous aux éléments de modélisation de base que nous utilisons. Nous nous référons à [Laborie 2018] pour une définition plus détaillée de ces éléments.

— Une **tâche** $i \in \mathscr{I}$ est modélisée par une variable de décision $T_{i,m}$ de type `intervalle` $T_i$ avec une date de disponibilité 0 et une date de rendu $CT$. Dans le langage de modélisation de CP Optimizer, cela s'écrit, pour toutes les tâches $i \in \mathscr{I}$ :

$$\texttt{dvar interval } T_i \texttt{ in } 0..CT$$

— **Modes** : Un mode $m \in \mathscr{M}_i$ est représenté par une variable intervalle `optionnelle`, c'est-à-dire que sans spécification contraire, cette tâche n'a pas à être présente dans la solution. Cela s'écrit, avec $p_{i,m}$ la durée de la tâche $i$ dans le mode $m$ :

$$\texttt{dvar interval optional } T_{i,m} \texttt{ in } 0..CT \texttt{ size } p_{i,m}$$

Chaque tâche $i \in \mathscr{I}$ est une `alternative` entre les différentes tâches optionnelles qui représentent ses modes :

$$\texttt{alternative}(T_i, (T_{im})_{m \in \mathscr{M}})$$

Puisque les variables $T_i$ ne sont pas optionnelles, au moins une des tâches $T_{im}$ doit être présente dans la solution.

— **Contraintes de précédence** : Des contraintes de précédence standards entre une tâche $i$ et son successeur $i'$, pour chaque couple $(i,i') \in$ Prec s'écrivent :

$$\texttt{endBeforeStart}(T_i, T_{i'})$$

De la même manière, les contraintes de précédences généralisées pour un couple de tâches $(i,i') \in \text{Prec}_\text{G}$ sont définies avec :

$$\texttt{startBeforeEnd}(T_{i'}, T_i, -\Delta)$$

— **Contraintes de ressources** : L'utilisation d'une ressource $r \in \mathscr{R}$ par une tâche $i \in \mathscr{I}$ dans un mode $m \in \mathscr{M}$ est modélisée avec une fonction `pulse`, qui vaut 0 en dehors de l'intervalle d'exécution de la tâche et qui vaut $b_{i,m,r}$ quand la tâche est en train d'être exécutée. On peut alors écrire la consommation totale d'une ressource comme une somme de fonction `pulse` :

– pour chaque zone $a \in \mathscr{R}^A$, $\sum_{\substack{i \in \mathscr{I}, \\ m \in \mathscr{M}}} \texttt{pulse}(T_{i,m}, b_{i,m,a}) \leq C_a$

– pour chaque type d'opérateur $o \in \mathscr{R}^O$, $\sum_{\substack{i \in \mathscr{I}, \\ m \in \mathscr{M}}} \texttt{pulse}(T_{i,m}, b_{i,m,o}) \leq n_o$

— **Contraintes d'incompatibilité** : Pour chaque paire $(i,i')$ de tâches incompatibles, on utilise la contrainte `noOverlap` :

$$\texttt{noOverlap}(T_i, T_{i'})$$

— **Objectif** : $\min \sum_{o \in \mathscr{R}^O} n_o$

## 2.4 Méthode de recherche à voisinage étendu

### 2.4.1 Principe de base

Dans les sections précédentes, nous avons introduit la formulation PPC dans le but de résoudre notre problème de manière exacte. Cependant, étant NP-Difficile, les méthodes exactes ne sont pas nécessairement les plus appropriées. La plus grande instance résolue par le modèle de PPC en 20 secondes comporte 100 tâches alors que le problème réel d'ordonnancement de lignes d'assemblage aéronautiques en comporte plus de 700. Dans cette section, nous proposons une méthode heuristique basée sur la technique de recherche par grand voisinage (*Large Neighbourhood Search*, ou LNS) que nous évaluons par rapport au solveur CP Optimizer et à l'heuristique utilisée en pratique sur des instances industrielles par le constructeur aéronautique. Pour illustrer le principe général des techniques LNS, inspiré de [Palpant 2004], on propose le schéma suivant :

(0) Calculer une solution initiale $\mathscr{S}$ du problème $\mathscr{P}$.
(1) Fixer une partie de la solution $\mathscr{S}$ "critique" du point de vue de la fonction objectif.
(2) Calculer une nouvelle solution $\mathscr{S}'$ pour le problème $\mathscr{P}'$, où $\mathscr{P}'$ est le problème obtenu depuis $\mathscr{P}$ en ajoutant les contraintes induites par l'étape (1).
(3) Si $\mathscr{S}'$ est une meilleure solution que $\mathscr{S}$ alors $\mathscr{S} \leftarrow \mathscr{S}'$.
(4) Si le critère d'arrêt n'est pas satisfait, aller à l'étape (1).
(5) Renvoyer $\mathscr{S}$.

Les principes génériques de LNS ne spécifient aucun type de diversification, c'est-à-dire une manière d'explorer rapidement un voisinage proche d'une solution en cours, mais reposent plutôt sur l'idée que si le voisinage d'une solution est suffisamment grand, la qualité des optima locaux dans le voisinage tend à être meilleure.

Clairement, l'un des principaux enjeux des algorithmes LNS est de décider quelle partie de la solution initiale fixer à l'étape (2) pour que son voisinage contienne de meilleures solutions ; en d'autres termes, comment évaluer la criticité d'une partie de la solution ?

### 2.4.2 Méthode LNS pour l'ordonnancement à modes multiples avec objectif de nivellement de ressources

Comme mentionné dans la section 2.1, la méthode LNS est généralement appliquée aux problèmes d'ordonnancement où l'objectif est de minimiser un critère lié au temps ou un coût d'externalisation. Un grand voisinage typique d'une solution dans ce contexte consiste à sélectionner un intervalle de temps et à fixer toutes les tâches programmées en dehors de cet intervalle et à compacter autant que possible les activités programmées à l'intérieur de cet intervalle afin d'utiliser au mieux les ressources disponibles. C'est le cas de la première méthode LNS proposée pour le RCPSP [Palpant 2004]. Notamment, la recherche

par défaut du solveur CP Optimizer que nous avons utilisé dans la section précédente implémente également une méthode LNS basée sur ce principe [Laborie 2007]. Or dans notre problème, nous cherchons au contraire à minimiser l'utilisation maximale d'un ensemble de ressources données. Pour ainsi dire, étant donné un intervalle de temps on préfère non pas compacter, mais étaler les tâches programmées dans cet intervalle. Ainsi, nous cherchons à identifier l'ensemble des tâches qui sont impliquées dans les pics d'utilisations maximales de chaque ressource. C'est à partir de cet ensemble (ou, pour être plus précis, de son complémentaire) que nous allons calculer la partie de la solution que l'on fixera dans notre méthode LNS.

Décrivons maintenant ce problème plus formellement. Considérons une solution $\mathscr{S}$ où chaque tâche $i \in \mathscr{I}$ a un temps de départ $\bar{S}_i \in [0, CT]$, un mode $\bar{m}_i \in \mathscr{M}_i$ et un nombre maximal d'opérateurs $\bar{n}_o$ ainsi qu'un nombre d'opérateurs assignés à la tache $i$ $b_{i,\bar{m}_i,o}$ pour chaque profil d'opérateurs $o \in \mathscr{R}^O$. L'ensemble des tâches qui forment les pics d'utilisation maximales des ressources est l'ensemble de tous les ensembles critiques tels que définis ci-dessous :

**Définition 1.** *Un ensemble critique $\tilde{\mathscr{I}}$ est un ensemble de tâches superposées qui atteint le nombre maximal d'opérateurs pour au moins un profil $o \in \mathscr{R}^O$. Plus formellement :*

$$\tilde{\mathscr{I}} = \{i \in \mathscr{I} | \exists t \in [0, CT], \exists o \in \mathscr{R}^O, \bar{S}_i \leq t < \bar{S}_i + p_{i,\bar{m}_i}, \sum_{\substack{j \in \mathscr{I}, \\ \bar{S}_j \leq t < \bar{S}_j + p_{j,\bar{m}_j}}} b_{j,\bar{m}_j,o} = \bar{n}_o\}$$

On peut remarquer que l'utilisation des ressources ne change qu'au début ou à la fin d'une tâche. Soit $\mathscr{T}$ l'ensemble des différents temps de départ et de fin des tâches. L'ensemble de tous les ensembles critiques peut être énuméré par un algorithme de balayage (ou `sweep`) qui teste la condition de la définition 1 pour chaque ensemble construit par la tâche qui chevauche chaque point de temps dans $\mathscr{T}$. L'algorithme 1 décrit l'algorithme de balayage qui calcule l'ensemble de toutes les tâches "pics" $\mathscr{C}$ en $\mathscr{O}(|\mathscr{I}|^2|\mathscr{R}^O|)$ temps.

Afin de générer un voisinage de bonne qualité, nous laissons libres (c'est-à-dire non fixées) toutes les tâches qui contribuent à l'utilisation maximale de la ressource objective (celles appartenant à l'ensemble des pics calculés par l'algorithme `sweep` ainsi que les tâches qui doivent les précéder par une contrainte de précédence, et nous fixons toutes les autres.

On résout ensuite ce nouveau problème en tenant compte de la borne fournie par la valeur de la solution initiale et des contraintes induites par les tâches fixées, dans un temps limité. Si une solution a été trouvée, elle remplace la solution initiale comme meilleure solution et on recommence. Cependant, si aucune solution n'a été trouvée, nous résolvons un nouveau problème, en fixant moins de tâches et en fixant un temps de résolution plus grand, en utilisant le principe auto-adaptatif initialement proposé par [Palpant 2004]. Pour être plus précis, à chaque fois que le solveur ne parvient pas à trouver une solution, nous fixons 10% d'activités en moins et ajoutons 10 secondes au temps de résolution maximum. Ces valeurs ont été déterminées empiriquement en utilisant des instances issues de benchmark.

Dans notre implémentation, nous utilisons CP Optimizer comme une boîte noire pour

**Algorithme 1** Algorithme `sweep` pour le calcul des tâches "pics"

**Entrée:** Un problème $\mathscr{P}$ et une solution $\mathscr{S} = \{(\bar{S}_i, \bar{m}_i)_{i\in\mathscr{I}}, (\bar{n}_o)_{o\in\mathscr{R}^O}\}$
  $\mathscr{C} \leftarrow \emptyset$
  $\mathscr{T} \leftarrow \{\bar{S}_i | i \in \mathscr{I}\} \cup \{\bar{S}_i + p_{i,\bar{m}_i} | i \in \mathscr{I}\}$
  **pour** $o \in \mathscr{R}^o$ **faire**
    **pour** $t \in \mathscr{T}$ **faire**
      $\tilde{\mathscr{I}} \leftarrow \emptyset$ ; $cons \leftarrow 0$
      **pour** $i \in \mathscr{I}$ **faire**
        **si** $b_{i,\bar{m}_i,o} > 0$ **and** $\bar{S}_i \leq t < \bar{S}_i + p_{i,\bar{m}_i}$ **alors**
          $\tilde{\mathscr{I}} \leftarrow \tilde{\mathscr{I}} \cup \{i\}$ ; $cons \leftarrow cons + b_{i,\bar{m}_i,o}$
        **fin si**
      **fin pour**
      **si** $cons = \bar{n}_o$ **alors**
        $\mathscr{C} \leftarrow \mathscr{C} \cup \tilde{\mathscr{I}}$
      **fin si**
    **fin pour**
  **fin pour**
  **retourne** $\mathscr{C}$

résoudre les différents sous-problèmes générés à l'aide du modèle de programmation par contraintes décrit dans la section 2.3. L'algorithme 2 fournit le pseudo-code de notre implémentation de la méthode LNS pour le problème d'ordonnancement de lignes d'assemblage aéronautiques.

Notez que `presenceOf`$(T_{i,m})$ dans le langage de l'Optimiseur CP est une contrainte qui impose la présence de la tâche optimale $T_{i,m}$, tandis que `startAt`$(T_i, t)$ est une contrainte qui fixe l'heure de début de la tâche $T_i$ à la valeur $t$. Ces deux contraintes sont utilisées pour fixer les modes et les temps de démarrage des tâches dans $W \setminus W'$ tandis que les tâches dans $W'$ sont libérées et forment le sous-problème LNS. Il est aussi à noter que dans cet algorithme $\tau'$ est une valeur bien inférieure à $\tau$ donnant le temps consacré au solveur CP pour obtenir une solution initiale.

### 2.4.3 Exemple déroulé de la méthode LNS

Dans cette section nous allons détailler le fonctionnement de l'algorithme LNS sur une petite instance du problème de MMRCPSP. L'instance que l'on va considérer comporte 10 tâches, et deux ressources de type "zone" $\mathscr{R}^A = \{1,2\}$ dont les capacités sont respectivement 15 et 13 et deux ressources de type "opérateurs" $\mathscr{R}^O = \{3,4\}$. Le temps de cycle CT vaut 15, il n'y a pas de contraintes d'incompatibilité et l'ensemble de contraintes de précédences (uniquement simples) est :

$$\text{Prec} = \{(1,4),(2,5),(2,8),(2,10),(3,7),(3,9),(4,8),(4,9),(5,6),(6,9),(7,8),(7,10)\}$$

Les autres caractéristiques de l'instance sont données dans la table 2.1. Pour l'exemple, on dira que l'on alloue une minute au total pour l'algorithme LNS.

Tout d'abord, on calcule une solution initiale $s^0$, qui est la première solution trouvée

**Algorithme 2** Méthode LNS pour l'ordonnancement à modes multiples avec objectif de nivellement des ressources

**Entrée:** Un problème d'ordonnancement de lignes d'assemblage $\mathscr{P}$, sa formulation de PPC (Section 2.3) et deux limites de temps $\tau$ et $\tau'$.

Initialiser la solution $\mathscr{S}^* = \{(S_i^*, p_i^*, m_i^*)_{i\in\mathscr{I}}, (n_o^*)_{o\in\mathscr{R}^O}\}$ comme étant la première solution de $\mathscr{P}$ calculée avec CP Optimizer avec $\tau'$ comme limite de temps

$\pi_{ratio} \leftarrow 100$

$\tau_{base} \leftarrow 10$

$\tau_{inc} \leftarrow 0$

**tant que** temps écoulé $< \tau$ **faire**

$\tilde{W} \leftarrow \texttt{sweep}(\mathscr{P}, \mathscr{S}^*)$ (récupérer les tâches "pics")

$W^* \leftarrow \tilde{W} \cup \{W' \in W | (w', w) \in Prec\}$ (récupérer les tâches qui les précède)

$W' \leftarrow$ un sous-ensemble $W^*$ qui contient $\pi_{ratio}\%$ de tâches sélectionnées aléatoirement.

$\mathscr{P}' \leftarrow \mathscr{P}$

**pour** $i \in W \setminus W'$ **faire**

$\mathscr{P}' \leftarrow \mathscr{P}' \cup \texttt{presenceOf}(T_{i,m_i^*})$

$\mathscr{P}' \leftarrow \mathscr{P}' \cup \texttt{startAt}(T_i, S_i^*)$

**fin pour**

$\mathscr{P}' \leftarrow \mathscr{P}' \cup \{\sum_{o\in\mathscr{R}^O} n_o < \sum_{o\in\mathscr{R}^O} n_o^*\}$

Récupérer la solution $\mathscr{S} = \{(\bar{S}_i, \bar{m}_i)_{i\in\mathscr{I}}, (\bar{n}_p)_{p\in P}\}$ en résolvant $\mathscr{P}'$ avec CP Optimizer avec comme temps limite $t = \min(t_{base} + t_{inc}, T - \text{temps écoulé})$

**si** $\sum_{o\in\mathscr{R}^O} \bar{n}_o < \sum_{o\in\mathscr{R}^O} n_o^*$ **alors**

$\mathscr{S}^* \leftarrow \mathscr{S}$

$\tau_{inc} \leftarrow 0$

$\pi_{ratio} \leftarrow 100$

**sinon si** $\mathscr{S}$ est vide **alors**

$\tau_{inc} \leftarrow \tau_{inc} + 10$

$\pi_{ratio} \leftarrow \pi_{ratio} - 10$

**fin si**

**fin tant que**

**retourne** $\mathscr{S}^*$

| Tâche | Mode | Durée | $b_{i,1,m}$ | $b_{i,2,m}$ | $b_{i,3,m}$ | $b_{i,4,m}$ |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 0 | 5 | 3 | 5 |
| | 2 | 8 | 0 | 4 | 2 | 4 |
| | 3 | 10 | 0 | 2 | 1 | 3 |
| 2 | 1 | 4 | 0 | 3 | 3 | 2 |
| | 2 | 5 | 9 | 0 | 3 | 2 |
| | 3 | 7 | 0 | 2 | 1 | 2 |
| 3 | 1 | 2 | 0 | 7 | 4 | 3 |
| | 2 | 2 | 6 | 0 | 5 | 4 |
| | 3 | 5 | 0 | 7 | 1 | 2 |
| 4 | 1 | 2 | 5 | 0 | 8 | 6 |
| | 2 | 8 | 4 | 0 | 5 | 3 |
| | 3 | 10 | 4 | 0 | 4 | 1 |
| 5 | 1 | 2 | 6 | 0 | 7 | 10 |
| | 2 | 6 | 4 | 0 | 6 | 7 |
| | 3 | 8 | 2 | 0 | 6 | 6 |
| 6 | 1 | 2 | 6 | 0 | 5 | 9 |
| | 2 | 7 | 0 | 9 | 4 | 9 |
| | 3 | 7 | 3 | 0 | 5 | 8 |
| 7 | 1 | 5 | 0 | 6 | 2 | 9 |
| | 2 | 7 | 0 | 4 | 2 | 5 |
| | 3 | 10 | 5 | 0 | 1 | 2 |
| 8 | 1 | 8 | 7 | 0 | 4 | 6 |
| | 2 | 10 | 0 | 7 | 3 | 3 |
| | 3 | 10 | 4 | 0 | 2 | 4 |
| 9 | 1 | 6 | 7 | 0 | 9 | 7 |
| | 2 | 6 | 0 | 2 | 9 | 8 |
| | 3 | 7 | 7 | 0 | 6 | 5 |
| 10 | 1 | 8 | 0 | 2 | 8 | 5 |
| | 2 | 10 | 0 | 1 | 5 | 1 |
| | 3 | 10 | 4 | 0 | 7 | 3 |

TABLE 2.1 – Une instance à 10 tâches pour le MMRCPSP

par le solveur lorsqu'il résout cette instance. Cette solution est affichée dans la figure 2.3. Cette solution est calculée en moins d'une seconde.

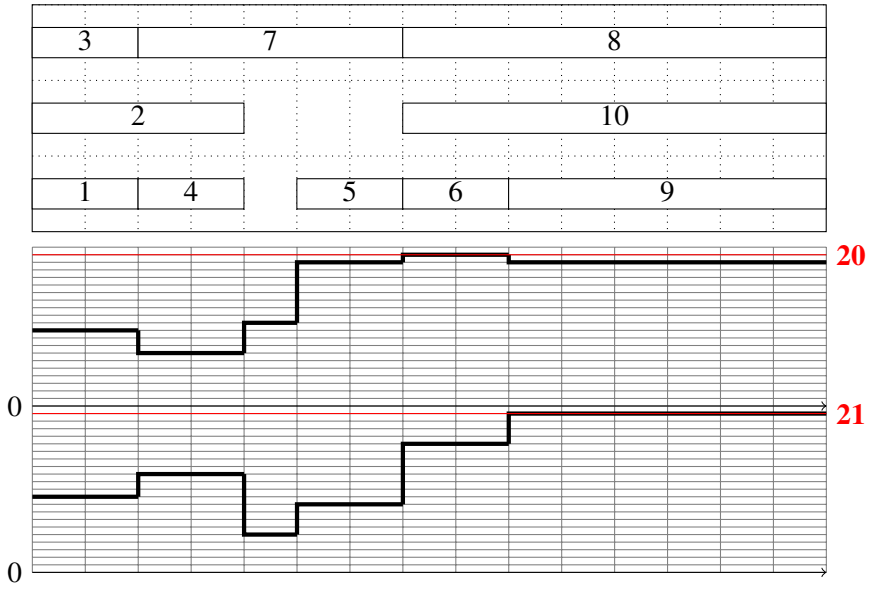



FIGURE 2.3 – Solution initiale $s^0$

Les traits rouges indiquent l'utilisation maximale de chacune des ressources de type "opérateurs". Ainsi, la valeur objectif de cette solution est $20+21=41$. Grâce à l'algorithme *sweep*, on calcule l'ensemble des tâches qui constituent ces pics d'utilisation maximale, et on obtient $\tilde{W}=\{6,8,9,10\}$. Maintenant si l'on étend cet ensemble de tâches avec toutes les tâches de $\mathscr{I}$ qui sont des prédécesseurs des tâches de $\tilde{W}$, on a $W^*=\{2,3,4,5,6,7,8,9,10\}$, soit toutes les tâches sauf la première. Ainsi, la seule tâche que l'on pourra fixer pour chercher un voisinage à la prochaine itération de l'algorithme sera la tâche 1. Cette condition ne restreint pas beaucoup le voisinage de $s^0$ qui sera exploré ici, mais sur des instances plus grandes, l'ensemble des tâches que l'on peut avoir à fixer peut-être beaucoup plus important.

On résout maintenant à nouveau le problème, mais cette fois en contraignant la tâche 1 à garder la même date de départ et le même mode que dans la solution $s^0$, et avec un temps limite $\tau=10$ secondes. Cette fois, le solveur trouve une nouvelle solution $s^1$, encore une fois en moins d'une seconde. Cette nouvelle solution est présentée dans la figure 2.4.

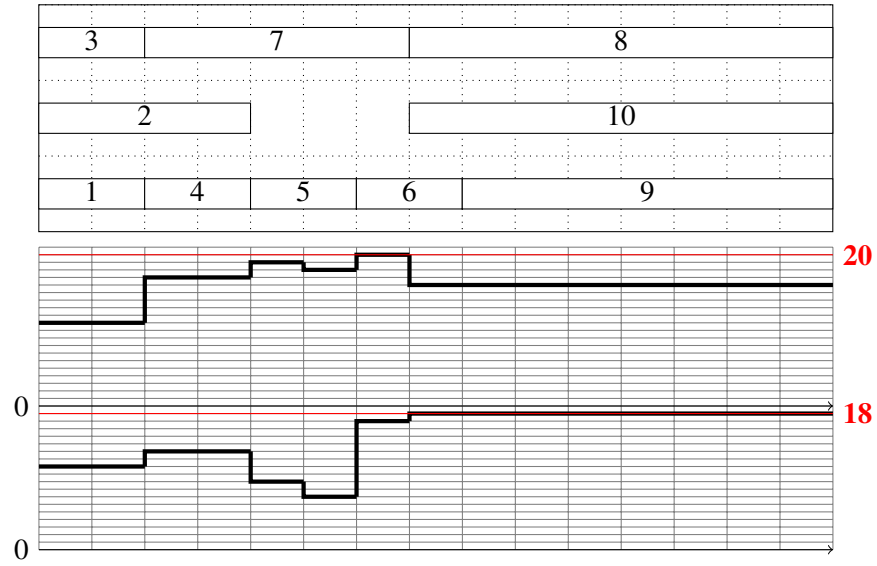



FIGURE 2.4 – Une autre solution $s^1$

On peut voir que cette solution $s^1$ a une valeur objectif qui vaut $20+18=38$, elle est donc meilleure que $s^0$. Une fois encore, on calcule l'ensemble des tâches "pics" et on trouve à nouveau $\tilde{W}=\{6,8,9,10\}$, puis $W^*=\{2,3,4,5,6,7,8,9,10\}$. Ce cas est intéressant, puisque le voisinage que l'on considérera à la prochaine itération est le même que celui que l'on a considéré à l'itération courante. Puisque le temps de calcul et le voisinage sera le même, le solveur ne parviendra pas à trouver une solution qui est strictement meilleure que $s^1$, qui, puisque cette instance est toute petite, et que ce sous-problème est résolu à l'optimum par le solveur, est donc ici un optimum local. On tombe alors dans le cas où l'on se laisse un peu plus de temps de calcul, c'est-à-dire 20 secondes, et on décide de fixer moins de variables, ce qui dans ce cas revient à laisser la tâche 1 libre à nouveau.

A l'itération suivante, le solveur ne parvient toujours pas à améliorer $s^1$. C'est un cas particulier : le solveur n'a toujours pas réussi à trouver de meilleure solution, mais cette fois-ci, puisqu'aucune variable n'était fixée, le "voisinage" considéré était en fait l'espace de solutions tout entier. On a, par hasard, prouvé l'optimalité de la solution $s^1$ (encore une fois car l'instance exemple est de petite taille). Il est important de noter que c'est un cas particulier, et que cela n'a aucune raison de se produire dans le cas général. Sur des instances plus difficiles, le solveur ne parviendrait probablement pas à résoudre le problème sans aucune variable fixée dans le temps imparti (et si c'était le cas, alors l'utilisation de la méthode LNS ne serait pas vraiment judicieuse...). En général, l'algorithme s'arrête à la fin du temps alloué, et renvoie la meilleure solution trouvée. Ici, c'est la solution $s^1$.

TABLE 2.2 – Caractéristiques des petites et moyennes instances

| Set | $\lvert\mathscr{I}\rvert$ | $\lvert Prec\rvert$ | $\lvert$Prec_G$\rvert$ | $\lvert\mathscr{D}\rvert$ | $\lvert\mathscr{R}^O\rvert$ | $\lvert\mathscr{R}^A\rvert$ | $\sum_{i\in\mathscr{I}}\lvert\mathscr{M}_i\rvert$ |
|---|---|---|---|---|---|---|---|
| Set1-8 | 8 | 6 | 1 | 1 | 2 | 2 | 12 |
| Set2-8 | 8 | 8 | 1 | 1 | 2 | 2 | 16 |
| Set3-8 | 8 | 7 | 1 | 1 | 2 | 2 | 17 |
| Set4-8 | 8 | 7 | 1 | 1 | 2 | 2 | 16 |
| Set3-9 | 9 | 8 | 1 | 1 | 2 | 2 | 18 |
| Set3-10 | 10 | 9 | 1 | 1 | 2 | 2 | 18 |
| Set3-11 | 11 | 10 | 1 | 1 | 2 | 2 | 18 |
| Set4-9 | 9 | 8 | 1 | 1 | 2 | 2 | 16 |
| Set4-10 | 10 | 9 | 1 | 1 | 2 | 2 | 16 |
| Set4-11 | 11 | 10 | 1 | 1 | 2 | 2 | 16 |
| Set1-70 | 70 | 64 | 0 | 11 | 4 | 2 | 84 |
| Set1-100 | 100 | 90 | 3 | 14 | 4 | 2 | 134 |

## 2.5 Résultats sur les instances industrielles

### 2.5.1 Description des instances

Les expérimentations ont été réalisées sur un PC DELL Inspiron 1525 Intel(R) Core(TM)2 Duo CPU, T5550 @ 1.83 GHz et 3.0 Go de RAM. Les instances sont constituées, d'une part, d'un certain nombre d'instances réelles fournies par Airbus.

Tout d'abord, pour comparer la formulation de PPC proposée dans cette thèse avec la formulation de PLNE proposée dans [Borreguero 2021], quatre ensembles de petites instances de 8 tâches ont été utilisés. De plus, les ensembles 3 et 4 ont été étendus afin de créer des instances jusqu'à 11 tâches. Leurs caractéristiques sont répertoriées dans le tableau 2.2 qui donne, pour chaque famille d'instances, le nombre de tâches, le nombre de contraintes de précédence, le nombre de contraintes de décalage temporel maximal, le nombre de contraintes d'incompatibilités, le nombre total de profils d'opérateurs, le nombre de zones et le nombre total de modes différents. Dans l'ensemble, 75 combinaisons différentes d'ensembles de données, de nombre de tâches, de temps de cycle et de nombre d'événements ont été testées pour chaque formulation. Les deux dernières lignes de la table 2.2 donnent les caractéristiques de deux instances réelles de taille plus grande (70 et 100 tâches, respectivement).

Par ailleurs un autre ensemble de 7 instances réelles (notées A, B, C, D, E, F, et G) comportant de 90 à 721 tâches a été testé.

En raison de l'utilisation de CP Optimizer 12.6.0, l'horizon temporel a été mis à l'échelle dans chaque instance afin d'obtenir des durées entières. Ce n'est pas un problème pour les applications réelles, car les durées (exprimées en minutes) peuvent être considérées comme étant entières sans perte de généralité. Le mode de recherche par défaut de CP Optimizer a été utilisé.

TABLE 2.3 – Comparaison PLNE-PPC sur les petites et moyennes instances

| Inst | CT | SEE-M (STD) t(s) | OEE-M (STD) t(s) | OEE-M (MIN) t(s) | CP optimizer t(s) |
|---|---|---|---|---|---|
| Set1-8 | 11.5 | 21.09 | 1.93 | 0.67 | 0.01 |
| Set2-8 | 34.75 | 19.66 | 0.58 | 0.47 | 0.02 |
| Set3-8 | 14 | 121.98 | 8.24 | 0.22 | 0.13 |
| Set4-8 | 13 | 36.58 | 3.82 | 0.11 | 0.02 |
| Set3-11 | 17 | 4133 | 452.53 | 1.45 | 0.29 |
| Set4-11 | 13 | 473.54 | 76.32 | 13.07 | 0.05 |
| Set1-70 | 1200 | — | — | — | 2.3 |
| Set1-100 | 1700 | — | — | — | 20.7 |

TABLE 2.4 – Comparaison : CP Optimizer - LNS

| Inst. | $\|W\|$ | CP Optimizer | | | LNS | | |
|---|---|---|---|---|---|---|---|
| | | 1min | 15min | 30min | 1min | 15min | 30min |
| A | 90 | 6 | 6 | 6 | 6 | 6 | 6 |
| B | 159 | 17 | 17 | 13 | 20 | 14 | 13 |
| C | 455 | 20 | 19 | 19 | 19 | 18 | 18 |
| D | 455 | 20 | 20 | 18 | 22 | 18 | 18 |
| E | 721 | 24 | 21 | 21 | 25 | 21 | 21 |
| F | 486 | 23 | 20 | 19 | 23 | 17 | 17 |
| G | 165 | 20 | 16 | 15 | 24 | 15 | 13 |

### 2.5.2 Comparaison entre CP Optimizer et la PLNE

Les temps de résolution de la PPC et des trois formulations de PLNE SEE-M, OEE-M et OFF-M et le temps de cycle CT décrits dans [Borreguero 2021] sont répertoriés dans le tableau 2.3. Toutes les instances ont été résolues à l'optimalité par la PPC. Les petites instances ont pris moins d'une seconde, tandis que les instances plus importantes ont été résolues en 20 secondes au maximum, alors que le modèle PLNE n'a pas pu obtenir de solutions réalisables pour les instances plus grandes. Les résultats établissent clairement la supériorité du solveur CP Optimizer par rapport aux modèles PLNE sur le problème d'ordonnancement considéré.

### 2.5.3 Comparaisons entre LNS et CP Optimizer en tant qu'heuristique

Afin d'évaluer l'efficacité de l'algorithme LNS en tant qu'heuristique, nous utilisons l'ensemble d'instances A à G, dont certaines sont beaucoup plus difficiles à résoudre.

Dans nos expérimentations, nous avons exécuté l'algorithme de résolution du modèle PPC (toujours avec CP Optimizer) et l'algorithme LNS sur les instances avec trois limites de temps différentes : 1 minute, 15 minutes et 30 minutes. Le tableau 2.4 montre la valeur objectif des meilleures solutions trouvées par les deux algorithmes dans les différentes limites de temps. Dans la première colonne, le nombre de tâches par instance est égale-

TABLE 2.5 – Amélioration de la méthode LNS par rapport à CP Optimizer

| Inst. | $\|W\|$ | 1min | 15min | 30min |
|---|---|---|---|---|
| A | 90 | 0% | 0% | 0% |
| B | 159 | -17.6% | 17.6% | 0% |
| C | 455 | 5% | 5.2% | 5.2% |
| D | 455 | -10% | 10% | 0% |
| E | 721 | -4.2% | 0% | 0% |
| F | 486 | 0% | 15% | 10.7% |
| G | 165 | 20% | 6.2% | 13.3% |

TABLE 2.6 – Comparaison : LNS - Heuristique SSG

| Inst | $\|W\|$ | Heuristic | LNS | impr |
|---|---|---|---|---|
| A | 90 | 6 | 6 | 0% |
| B | 159 | 18 | 13 | 27.7% |
| C | 455 | 18 | 18 | 0% |
| D | 455 | 18 | 18 | 0% |
| E | 721 | 22 | 21 | 4.5% |
| F | 486 | 26 | 17 | 23% |
| G | 165 | | 13 | $\infty$ |

ment affiché, afin de fournir un indice sur la complexité de l'instance. Les deux méthodes ont fourni des solutions réalisables pour les 6 instances. Nous pouvons également noter que la LNS est peu performante dans le délai d'une minute. Cela n'est pas vraiment surprenant, puisque l'algorithme LNS commence par explorer le voisinage de la première solution réalisable trouvée par CP Optimizer en utilisant le modèle PPC. Cependant, nous avons observé que l'algorithme LNS trouve souvent des solutions de meilleure qualité que l'approche PPC dans des délais plus longs. Le tableau 2.5, qui donne l'amélioration en pourcentage du LNS sur CP Optimizer, montre que lorsque la limite de temps est de 15 minutes ou 30 minutes, l'approche LNS n'est jamais moins performante que CP Optimizer. La plus grande amélioration est obtenue pour une limite de temps CPU de 15 minutes avec une amélioration sur 5 instances sur 7.

Les instances A à G étant des instances réelles du constructeur, nous avons également eu l'occasion de comparer les performances de l'algorithme LNS aux résultats de l'heuristique de génération d'ordonnancement en série (SSG) orientée activité actuellement déployée chez le constructeur aéronautique. Cette comparaison est présentée dans le tableau 2.6. On peut voir que la LNS conduit à de meilleures solutions pour 4 des 6 instances. Pour les deux cas où elle est arrivée à la même solution (C, D), la méthode LNS a pu prouver l'optimalité, qui ne peut être prouvée par l'heuristique. De plus, pour l'instance G, l'heuristique n'est pas capable de trouver une solution alors que LNS parvient à en calculer une.

TABLE 2.7 – Amélioration de la méthode LNS par rapport à CP Optimizer pour le problème : Multi-mode Resource Investment Problem

| Inst. | 1 min | 15 min | 30 min |
|---|---|---|---|
| $MRIP_{30}$ | 5.27% | 11.15% | 0.82% |
| $MRIP_{50}$ | 5.61% | 15.47% | 1.14% |
| $MRIP_{100}$ | 10.58% | 17.46% | 3.40% |

## 2.6 Résultats sur un problème à modes multiples de la littérature

Dans cette section, nous proposons d'aller un peu plus loin dans les expériences. Dans [Gerhards 2020], l'auteur propose un modèle de programmation par contraintes pour résoudre le Multi-mode resource investment Problem (MMRIP). Formellement, ce problème est très similaire au problème d'ordonnancement de la chaîne de montage dont il est question ici. Les différences notables sont, d'une part, la présence de ressources non renouvelables, et d'autre part, la fonction objectif, où l'on veut minimiser la somme pondérée des maximums des ressources utilisées. De la même manière que nous avons utilisé notre modèle de CP comme boîte noire pour l'algorithme LNS, nous allons cette fois utiliser le modèle de PPC que l'auteur propose comme boîte noire et comparer les résultats obtenus par LNS avec les résultats de son modèle de PPC. Nous utilisons le même algorithme de balayage que celui présenté dans la section précédente. Nous testons notre algorithme sur les mêmes instances que celles utilisées par l'auteur, tirées du jeu de données RIPlib. Ces instances sont séparées en trois sous-ensembles, avec 30, 50 et 100 tâches. Notez que nous avons retiré de ces instances celles qui ont été résolues par CP Optimizer en moins d'une minute, afin que les résultats mettent plus précisément en évidence la comparaison sur les instances difficiles de ce jeu de données. Les deux méthodes ont été testées en 1, 15 et 30 minutes. L'amélioration moyenne (en %) est affichée dans le tableau 2.7.

En observant les résultats, nous pouvons remarquer plusieurs choses. Pour tous les ensembles d'instances, l'algorithme LNS améliore très peu les solutions retournées par la solution du modèle PPC après 30 minutes, ce qui est explicable par le fait que la plupart des instances sont résolues de manière optimale par CP Optimizer après ce temps, alors que pour notre méthode, plus la solution en cours est de bonne qualité, plus le voisinage en cours a de chance d'être très large. En effet, une "bonne solution" aura tendance à avoir un profil d'utilisation des ressources plus "aplati", ainsi l'ensemble des tâches qui constituent les pics d'utilisation est plus grand, ce qui mène à un important voisinage, et donc un temps de calcul plus long pour améliorer la solution. En revanche, c'est avec un délai de 15 minutes que la différence de qualité des solutions est la plus notable : dans ce laps de temps, le modèle PPC n'est pas encore résolu, alors que l'algorithme LNS a pu explorer un sous-ensemble plus intéressant de solutions. On peut noter que cette différence augmente avec la taille des instances.

## 2.7 Conclusions & Perspectives

Dans ce chapitre, nous avons proposé un modèle de programmation par contraintes. Résolu directement par le solveur CP Optimizer, il a obtenu des résultats prometteurs, même avec des instances réelles et plus importantes.

Remarquant que les méthodes intégrées dans le solveur CP Optimizer, en particulier la méthode LNS décrite dans [Godard 2005] sont orientées temps et visent globalement à compacter l'ordonnancement, nous avons proposé une nouvelle méthode LNS plus adaptée à l'objectif de nivellement des ressources. Nous avons utilisé un algorithme de balayage rapide pour calculer les pics d'utilisation des ressources et un voisinage auto-adaptatif pour réordonnancer efficacement les tâches impliquées dans ces pics et leurs prédécesseurs.

La méthode surpasse CP Optimizer sur les instances industrielles pour des limites de temps CPU moyennes (15 et 30 minutes).

De plus, elle parvient à diminuer de plus de 20% l'utilisation maximales de ressources atteintes par l'heuristique actuellement utilisée par le constructeur sur les instances industrielles, ce qui permet de réaliser des gains substantiels.

Enfin, la méthode LNS proposée améliore sensiblement les résultats du modèle CP proposé par [Gerhards 2020] sur des instances difficiles d'un problème connexe d'investissement en ressources multi-modes pour des limites de temps CPU courtes (1 et 15 minutes).

Une direction de recherche future concerne les techniques de filtrage associées pour résoudre le problème étudié. Le schéma de solution PPC englobe également des techniques de propagation de contraintes qui peuvent être utilisées comme prétraitement pour calculer le nombre pertinent d'événements ; cette question permettrait d'améliorer considérablement les performances. Une direction de recherche prometteuse est de proposer une heuristique hybride MILP/CP de recherche de grand voisinage, comme celle proposée pour le challenge MISTA 2013 sur le RCPSP multi-mode [Artigues 2013a]. Des contraintes et des objectifs ergonomiques pour les opérateurs pourraient également être considérés pour le problème en question, comme récemment proposé dans un problème connexe par [Arkhipov 2018]. Enfin la prise en compte des incertitudes dans le problème fera l'objet des deux chapitres suivants.

CHAPITRE 3

# Arbres de décision robustes et application à un problème à une machine

**Sommaire**



Ce chapitre a fait l'objet d'une publication dans la conférence internationale avec actes *Information Processing and Management of Uncertainty in Knowledge-Based Systems* 2020 [Portoleau 2020b]. Ces travaux ont été également présentés dans les conférences ROADEF 2020 [Portoleau 2020a] et PMS 2020/2021 [Portoleau 2021b].

## 3.1 Introduction

Inspirés par des méthodes de planification et d'ordonnancement contingents (voir §1.2.4), et empruntant des idées aux approches robustes multi-étapes [Bertsimas 2011] (voir §1.2.3)), nous proposons ici une approche basée sur le calcul d'un arbre de décision.

Nous considérons des problèmes d'ordonnancement répétés où certains paramètres sont connus et constants à chaque itération d'ordonnancement et certains d'entre eux sont connus avec imprécision, selon les scénarios. Afin de tirer profit des paramètres connus et constants, nous introduisons une méthode proactive-réactive dans laquelle nous supposons

que le décideur peut avoir accès à certaines informations sur les paramètres imprécis pour réduire l'incertitude à des moments prédéfinis de l'exécution de l'ordonnancement, où l'ordonnancement peut être modifié. Cependant, l'obtention d'informations sur les paramètres incertains peut être coûteuse et une reprogrammation fréquente peut être perturbante. Par conséquent, à chaque point de décision, le planificateur a le choix d'utiliser l'information ou non et de réagir ou non en fonction de l'impact de la réaction sur la robustesse du planning. En utilisant intelligemment ces informations, nous construisons dans une phase hors ligne un arbre de décision qui sera utilisé pour résoudre le problème d'ordonnancement à chaque répétition de ce dernier.

L'idée des arbres de décision est qu'ils sont facilement utilisables par un décideur, et permettent de traiter des incertitudes faibles mais identifiées en proposant dynamiquement des solutions en fonction de l'arrivée de nouvelles informations, tout en garantissant la qualité des solutions dans le pire des cas.

Afin de valider notre modèle, nous avons choisi de l'appliquer au problème d'ordonnancement simple $1||L_{max}$, dans lequel nous supposons que les durées des tâches sont connues et fixes, et que les dates d'échéances sont incertaines. Nous détaillons pourquoi nous avons choisi ce problème dans la section 3.4. La motivation du modèle des arbres de décision robustes vient du monde industriel. Ce chapitre a pour but d'introduire les arbres de décisions robustes sur un problème "jouet", nous appliquerons ce modèle au problème industriel présenté dans le chapitre 1 au prochain chapitre.

Le plan de ce chapitre est le suivant. La section 3.2 définit formellement les modèles d'incertitude et d'information ainsi que le concept d'arbre de décision robuste et le problème qui lui est lié. Ensuite, nous détaillons l'algorithme que nous avons conçu pour résoudre ces problèmes, à savoir l'algorithme général de construction de l'arbre de décision robuste (Section 3.3) et l'algorithme de résolution du sous-problème de partitionnement robuste (Section 3.4). Nous présentons ensuite quelques résultats numériques dans la section 3.5 en comparant notre approche à un algorithme d'ordonnancement proactif-réactif plus standard.

## 3.2 Notations et définitions

### 3.2.1 Modèle d'incertitude

En pratique, il est souvent plus facile pour un décideur d'établir des limites sur des paramètres incertains, comme les délais de traitement ou les dates d'échéances, que de construire un modèle probabiliste précis qui nécessite souvent une grande quantité de données. Dans ce chapitre, nous considérons l'incertitude par intervalle. Étant donné un paramètre incertain $x$, nous désignons par $X = [x_{min}, x_{max}]$ l'intervalle dans lequel il prend sa valeur. Nous ne faisons aucune hypothèse sur la loi probabiliste suivie par $x$ dans cet intervalle. Étant donné un ensemble de paramètres incertains $(x_i)_{i\in I}$, nous définissons l'ensemble des affectations possibles des paramètres par $\Omega = \prod_{i\in I} X_i$ (c'est-à-dire que nous supposons qu'il n'y a pas de corrélation entre eux, toutes les réalisations $x_i$ sont indépendantes). Nous appelons scénario une affectation de chaque paramètre $x_i, \forall i \in I$ tel que $(x_i)_{i\in I} \in \Omega$.

Dorénavant, lorsque $\omega$ est un scénario, $s$ un ordonnancement et $f$ l'objectif, nous désignons par $f(\omega, s)$ la valeur objective de $s$ dans le scénario $\omega$ (notez que pour notre cas d'application, $f = L_{max}$). Dans ce chapitre, nous considérons le problème $1||L_{max}$, c'est-à-dire que nous voulons ordonnancer un ensemble $I$ de tâches avec des temps de traitement déterministes $p_i, i \in I$ et des dates d'échéances incertaines $d_i \in D_i, i \in I$ sur une seule machine de façon à ce que le retard maximal soit minimal. Ce problème dans sa version déterministe peut être résolu en temps polynomial en insérant les tâches en suivant la règle *Earliest Due Date* (EDD), ou date de rendu au plus tôt. Cependant, il est à noter que ce problème, dans sa version robuste (avec objectif *minmax*) peut être également résolu en temps polynomial [Aloulou 2008], en appliquant la règle EDD en fixant les dates de rendu sur la borne inférieure de leur intervalle d'incertitude.

**Exemple 1.** *On considère une petite instance du problème* $1||L_{max}$ *avec trois tâches :*

- *task 1 :* $p_1 = 10$, $d_1 \in D_1 = [10, 11]$
- *task 2 :* $p_2 = 6$, $d_2 \in D_2 = [11, 17]$
- *task 3 :* $p_3 = 4$, $d_3 \in D_3 = [13, 20]$

*L'ensemble des scénarios pour cette instance est* $\Omega = D_1 \times D_2 \times D_3$ *et* $\omega = (10, 12, 19)$ *est un scénario. Nous conserverons cette instance tout au long de ce chapitre pour illustrer les notions et les algorithmes que nous présentons.*

### 3.2.2 Modèle d'information

Comme expliqué dans la section 3.1, nous supposons qu'à un moment donné pendant l'exécution de l'ordonnancement, certaines informations deviennent accessibles. Dans notre modèle, une information permet au planificateur de resserrer l'intervalle d'incertitude d'une réalisation future.

**Définition 2.** *Pour un paramètre incertain donné* $x \in X$ *et un moment de décision* $t$ *pendant l'exécution de l'ordonnancement, nous appelons une information sur* $x$ *une valeur* $k_x^t$ *et un opérateur dans* $\{\leq, \geq\}$.

Par exemple, une information est $x \leq k_x^t$. Ainsi, le décideur, à partir du temps $t$, est capable de réduire l'ensemble des affectations possibles en mettant à jour l'intervalle $X$, $x \in X_{k_x^t}^{inf} = [x_{min}, k_x^t]$ ou $x \in X_{k_x^t}^{sup} = [k_x^t, x_{max}]$. Notons que l'information dépend de $t$, ainsi l'ordonnanceur peut être amené à demander une information sur la même donnée plusieurs fois au cours de l'exécution de la planification. Notre modèle vise à utiliser au mieux les informations disponibles, et à sélectionner les plus pertinentes.
Pour le problème considéré dans ce chapitre, nous supposons que pour un moment de décision $t$ et une tâche $i$ donnés, nous avons une information $k_d^t$ si $\min(t + p_i, 2t) \in D_i$. Si c'est le cas, $k_d^t = \min(t + p_i, 2t)$. Sinon on considère que l'on ne dispose d'aucune information sur la tâche $i$ . Cette hypothèse sur la disponibilité de l'information est arbitraire, mais elle exprime en fait deux questions naturelles qu'un planificateur peut se poser sur des échéances incertaines : "Si la tâche $i$ commence maintenant, peut-elle être achevée sans être en retard ? Si oui, la date d'échéances $d_i$ est-elle éloignée de maintenant ?". Dans tous les cas, une réponse à ces questions permet à l'ordonnanceur de borner l'incertitude d'une date d'échéance $d_i$.

**Exemple 2.** *Examinons à nouveau l'instance de l'exemple 1. Nous supposons que nous sommes à un moment de décision $t = 10$ et que la tâche 1 a été planifiée en premier. Étant donné l'hypothèse que nous avons faite sur la disponibilité de l'information, nous avons :*

- *la tâche 1 est complétée, il n'y a pas d'information pertinente à son propos.*
- $\min(t + p_2, 2t) = 16 \in D_2$, *donc* $k^t_{d_2} = 16$.
- $\min(t + p_3, 2t) = 14 \in D_3$, *donc* $k^t_{d_3} = 14$.

*Ainsi, pour tout $\omega \in \Omega$, le décideur est capable de déterminer, à partir du temps $t = 10$, si $d_2 \leq 16$ ou si $d_2 > 16$, et si $d_3 \leq 14$ ou si $d_3 > 14$.*

On introduit une dernière notation qui sera utile dans la suite de ce chapitre. Si $K^t$ est un ensemble d'informations alors on désigne par $\mathscr{P}^{\Omega}_{K^t}$ la partition de $\Omega$ à partir des informations de $K^t$. Ce qui donne formellement :

$$\mathscr{P}^{\Omega}_{K^t} = \{p, p = \prod_{i \in I} X'_i | X'_i \in \{X^{inf}_{k^t_{x_i}}, X^{sup}_{k^t_{x_i}}\} \text{ si } k^t_{x_i} \in K^t \text{ et } X'_i = X_i \text{ sinon}\}$$

Plus visuellement, un ensemble d'information permet de découper l'ensemble des scénarios en plusieurs hyper-rectangles plus petit. La partition $\mathscr{P}^{\Omega}_{K^t}$ est constituée de l'ensemble de ces rectangles.

### 3.2.3 Arbre de décision robuste

**Définition 3.** *On rappelle qu'une solution du problème $1||L_{max}$ est une séquence de l'ensemble des tâches. Dans ce contexte, on appelle une solution partielle une séquence contenant un sous-ensemble de l'ensemble des tâches. La concaténation de solutions partielles revient à enchaîner les séquences dans le même ordre. On note l'opération de concaténation $\oplus$. Par exemple si $s_1 = i_2 \to i_3 \to i_5$ et $s_2 = i_1 \to i_4$ sont deux solutions partielles, alors la concaténation des deux dans le même ordre est une solution partielle $s_1 \oplus s_2 = i_2 \to i_3 \to i_5 \to i_1 \to i_4$. Un arbre de décision robuste $\mathscr{T}$ est un arbre dont les nœuds sont étiquetés avec un sous-ensemble de $\Omega$, et les arcs sont étiquetés avec une solution partielle. Si $v$ est un nœud de $\mathscr{T}$, $\Omega^v$ dénote un sous-ensemble de $\Omega$ associé à $v$. Un arbre de décision robuste satisfait les propriétés suivantes :*

*(i) Désignons par $(e^v_j)_{j \leq J^v}$ les enfants (descendants immédiats) du noeud $v$. Alors $\bigcup_{j \leq J^v} \Omega^{e^v_j} = \Omega^v$.*

*(ii) Pour tout chemin $(v_0, ..., v_m)$ où $v_0$ est la racine de $\mathscr{T}$ et $v_m$ est une feuille, l'ordonnancement obtenu en concaténant toutes les solutions partielles sur les arcs le long du chemin est réalisable. On note cette solution $s^m_0$.*

*(iii) Soit $v$ et $v'$ deux noeuds de $\mathscr{T}$. La solution partielle $s'$ associée à l'arc $(v, v')$ est robuste sur $\Omega^v$ :*

$$s' = \arg\min_{s \in S^{v'}_v} \max_{\omega \in \Omega^{v'}} f(\omega, s^v_0 \oplus s)$$

*où $S^v_{v'}$ l'ensemble de solutions admissibles sur cet arc.*

Un arbre de décision robuste peut être vu comme une représentation compacte d'un ensemble de solutions. Étant donné que le décideur a accès à des informations qui permettent

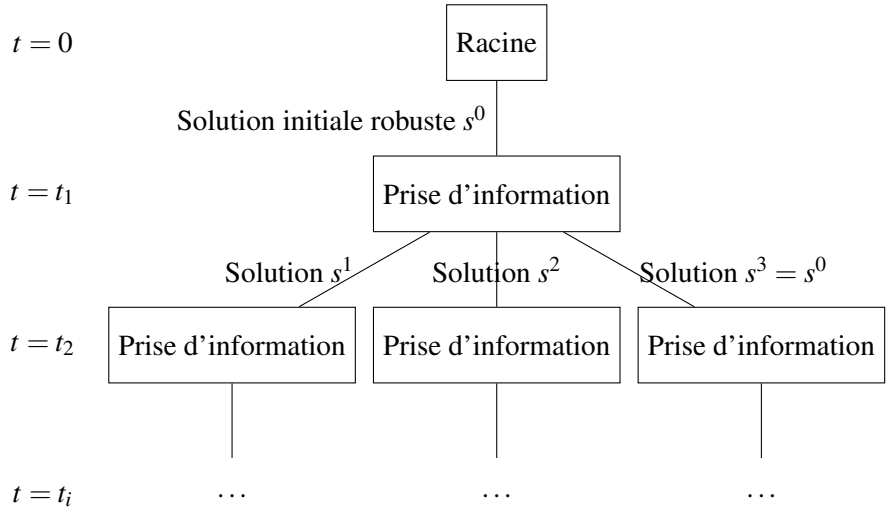



FIGURE 3.1 – Idée générale des arbres de décision robuste.

de diviser l'ensemble des scénarios, l'arbre permet de retrouver une solution, avec une garantie de robustesse, pour certains sous-ensembles de scénarios. Une illustration générique d'un arbre de décision robuste est présentée dans la figure 3.1. Dans ce cas, si le décideur est en mesure de savoir si un scénario en cours $\omega$ est dans $\Omega_1$ ou dans $\Omega_2$, il peut choisir la solution la plus robuste pour chaque cas.

Dans ce chapitre, notre objectif est, pour un ensemble donné de scénarios, de calculer un arbre de décision robuste qui soit réellement utilisable par un décideur pendant l'exécution du planning, permettant d'adapter le planning en ligne et de le rendre aussi robuste que possible, en fonction de certaines connaissances ou informations auxquelles le décideur a accès. Pour cela, nous considérons dans notre modèle que chaque niveau $j$ de l'arbre coïncide avec un instant fixe $t_j$ au cours de l'avancement de l'ordonnancement. Ces moments sont les moments où le décideur a accès à une certaine connaissance des paramètres incertains. Une illustration de ceci est donnée dans l'exemple 3.

**Exemple 3.** *Toujours en utilisant les données de l'exemple 1, si nous considérons un moment de décision $t = 10$, nous pouvons utiliser les informations calculées dans l'exemple 2. Pour cet exemple, nous définissons $\Omega_1 = D_1 \times [11,16] \times D_3$ et $\Omega_2 = \Omega \setminus \Omega_1$. Alors l'arbre de décision robuste représenté sur la figure 3.2 est valide. La première tâche à ordonnancer est la tâche 1, quel que soit le scénario en cours, puis grâce à l'information accessible au temps $t = 10$, le décideur sait si le scénario en cours $\omega$ est dans $\Omega_1$ ou dans $\Omega_2$, et peut alors passer à la solution la plus robuste (respectivement ordonnancer la tâche 2 avant la tâche 3 si $\omega \in \Omega_1$ et la tâche 3 avant la tâche 2 sinon).*

### 3.2.4 Partitionnement des scénarios

Le problème central de notre méthode est, pour tout nœud $n$, de calculer une partition robuste de l'ensemble des scénarios, mais comment comparer la robustesse de deux

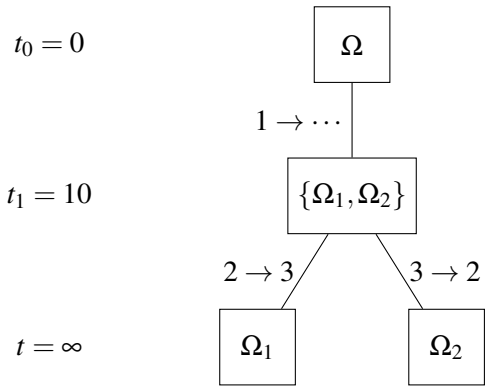



FIGURE 3.2 – Arbre de décision robuste pour l'exemple 3

partitions différentes ? Nous proposons le critère suivant.

**Définition 4.** *On définit le vecteur de Score de Robustesse (RS) d'une partition $\mathscr{P}$ comme le vecteur trié dans l'ordre croissant des valeurs objectives les plus défavorables de la solution robuste optimale (en considérant le critère de robustesse absolue [Kouvelis 1997]) sur chaque élément P de $\mathscr{P}$ :*

$$RS(\mathscr{P}) = sort(\min_{s \in S} \max_{\omega \in P} f(\omega, s))_{P \in \mathscr{P}}$$

*où S est l'ensemble des solutions réalisables.*

Il est à noter que le pire scénario "global" ne disparaît jamais, puisqu'il est toujours contenu dans au moins un des sous-ensembles de la partition. Ainsi, pour n'importe quelle partition $\mathscr{P}$ de $\Omega$ la dernière valeur de $RS(\mathscr{P})$ vaut la valeur objectif de la solution robuste sur $\Omega$.
C'est sur la base de ce vecteur que l'on comparera des partitions de scénarios. Puisque notre objectif est d'amoindrir le pessimisme inhérent au critère *minmax*, il nous faut pour cela une manière de comparer ces vecteurs de score de robustesse qui favorise les partitions "optimistes". Pour cela, étant donné deux partitions $\mathscr{P}$ et $\mathscr{P}'$, on dit que $\mathscr{P}$ est une meilleure partition que $\mathscr{P}'$ si $RS(\mathscr{P}) <_{leximin} RS(\mathscr{P}')$ où $<_{lexi}$ est l'ordre lexicographique. L'intuition derrière ce critère est la suivante : chaque valeur de ce vecteur est la valeur objective de la solution qui minimise sa valeur objective sur le pire scénario de chaque sous-ensemble composant la partition. Comme ces vecteurs sont triés dans l'ordre croissant, les partitions sont comparées en regardant tour à tour leurs sous-ensembles les plus optimistes respectifs qui permettent d'améliorer -dans leurs pire cas- au mieux la garantie de robustesse.

**Exemple 4.** *Considérons la partition $\mathscr{P}$ de $\Omega = \Omega_1 \cup \Omega_2$ de l'exemple 3. Désignons par $RV(\Omega)$ la valeur objective minimale sur $\Omega$, ou plus formellement $RV(\Omega) = \min_{s \in S} \max_{\omega \in \Omega} f(\omega, s)$. Nous avons :*

$RS(\mathscr{P}) = (\min(RV(\Omega_1), RV(\Omega_2)), \max(RV(\Omega_1), RV(\Omega_2))))$

$RS(\mathscr{P}) = (6, 7)$

*Soit maintenant $\mathscr{P}'$ une autre partition de $\Omega = \Omega'_1 \cup \Omega'_2$, avec $\Omega'_1 = D_1 \times D_2 \times [13,14]$ et $\Omega_2 = \Omega \setminus \Omega_1$. Nous calculons $RS(\mathscr{P}') = (4,7)$. Puisque $RS(\mathscr{P}') <_{lexi} RS(\mathscr{P})$, $\mathscr{P}'$ est une meilleure partition en considérant notre critère.*

Comme nous l'avons vu, une information $k_x^t$ permet de diviser en deux l'ensemble des scénarios, puisqu'elle permet de distinguer les scénarios où la donnée $x \in X_{k_x^t}^{inf}$ de ceux où $x \in X_{k_x^t}^{sup}$. Plus généralement, si $m$ informations sont disponibles, nous pouvons diviser l'ensemble des scénarios en $2^m$ sous-ensembles. Nous désignons par $K^t$ l'ensemble de toutes les informations disponibles au temps $t$. Notre objectif est d'utiliser ces sous-ensembles pour créer une partition de taille limitée de l'ensemble des scénarios et de calculer pour chaque sous-ensemble de scénarios dans la partition une nouvelle solution robuste. L'idée est que la diminution de la taille de l'ensemble des scénarios améliore nécessairement le scénario le plus défavorable et conduit donc à de meilleures solutions robustes. La taille maximale de cette nouvelle partition est un paramètre $L$, à fixer par le décideur. De plus, le nombre maximum d'informations que la partition est autorisée à utiliser est un autre paramètre $Q$.

Nous exprimons le problème central (notre méthode implique de le résoudre plusieurs fois) de notre approche, le problème de partitionnement robuste (RPP) :

**Problème de partitionnement robuste**
INSTANCE : Un ensemble de scénarios $\Omega$ de dimension $I$, un ensemble d'informations $K^t$ et deux entiers $Q$ et $L \leq 2^Q$ .
SOLUTION : Une partition $\mathscr{P}$ de $\Omega$ qui vérifie :
1- $|\mathscr{P}| \leq L$
2- $\forall P \in \mathscr{P}, \exists J_P \in \mathbb{N}$ , $P = \bigcup_{j \leq J_P} \prod_{i \in I} P_{i,j}$ avec $P_{i,j} \in \{X_{k_{x_i}^t}^{inf}, X_{k_{x_i}^t}^{sup}, X_i\}$ si $k_{x_i}^t \in K^t$ et $P_{i,j} = X_i$ sinon
3- $|\{k_{x_i}^t \in K^t | \exists P \in \mathscr{P}, \exists j \leq J_P, \exists i \in I, P_i \in \{X_{k_{x_i}^t}^{inf}, X_{k_{x_i}^t}^{sup}\}| \leq Q$
MESURE : $RS(\mathscr{P})$ minimal au sens lexicographique.

La contrainte 1 limite la taille de la partition de la solution, elle doit donc être inférieure à $L$. La contrainte 2 oblige la partition à être composée de sous-ensembles formés par les informations de $K^t$. En d'autres termes, une partition telle que l'un de ses sous-ensembles coupe l'hyperplan défini par $\omega_i = k_{x_i}^t$ n'est pas une solution réalisable. Enfin, la contrainte 3 oblige le nombre maximal d'informations à être inférieur à $Q$. Notez que dans la solution si nous avons $P_{i,j} = X_i$ pour tous les $P_{i,j}$ et $k_{x_i}^t \in K^t$, pour un paramètre $i$, cela signifie que malgré la disponibilité d'une nouvelle information sur le paramètre $i$ au temps $t$, cette information a été ignorée car l'ensemble d'incertitude $X_i$ n'est pas partitionné selon $k_{x_i}^t$. Cela peut être dû à la limite du nombre $Q$ d'informations pouvant être utilisées (contrainte 3) ou à l'absence d'impact positif de cette partition sur l'objectif de leximin (c'est-à-dire que l'information n'est pas localement pertinente pour améliorer la robustesse).

Maintenant que notre modèle est établi, les sections suivantes présentent les algorithmes que nous avons mis en œuvre pour produire un arbre de décision robuste et résoudre le RPP.

## 3.3 Algorithme pour les arbres de décisions robustes

Dans cette section, nous détaillons l'algorithme général permettant de construire un arbre de décision robuste.

En utilisant les définitions précédentes, nous proposons maintenant une méthode pour construire un arbre de décision robuste (voir Définition 3). Dans ce chapitre, nous considérons que les moments de décision (c'est-à-dire les moments où le décideur est capable d'accéder à de nouvelles informations et de modifier l'ordonnancement), notés $t \in T$ sont fixés à l'avance. Cela peut correspondre en pratique à des moments particuliers, comme la fin d'une journée de travail, ou un changement d'équipe où le planning peut être mis à jour. Chaque moment de décision correspond à un niveau dans l'arbre de décision, tel que $t_1$ correspond au premier niveau, $t_2$ au deuxième, etc... A cet égard, la profondeur de l'arbre est contrôlée par le nombre de moments de décision. A chaque bifurcation à un niveau $j$ de l'arbre, une nouvelle solution partielle, cohérente avec l'ordonnancement partiel qui a été exécuté jusqu'à $t_j$, est proposée en fonction de l'ensemble des scénarios actuels. La racine de l'arbre, que nous considérons comme étant le niveau 0, correspond à l'instant $t_0 = 0$, le début du planning. A ce stade, aucune information n'est connue, donc une seule solution robuste est proposée. Ainsi, un seul nœud est créé au niveau 1. A ce nœud, on récupère toutes les informations disponibles au temps $t_1$. En utilisant jusqu'à $Q$ informations, nous divisons l'ensemble des scénarios en -au plus $2^Q$- sous-ensembles formant une partition $\mathscr{P}$. Nous résolvons ensuite le problème de la partition robuste (nous détaillons cette procédure de résolution dans la section 3.4), et obtenons une partition robuste $\mathscr{P}'$. Pour chaque sous-ensemble de $\mathscr{P}'$, une nouvelle solution est proposée et une nouvelle branche est établie, conduisant à un nouveau nœud au niveau suivant. L'ensemble de scénarios considéré dans ce nœud est celui dont il est issu dans $\mathscr{P}'$. Ces étapes sont répétées jusqu'à ce que le dernier moment de décision soit atteint.

## 3.4 Algorithme pour le problème de partitionnement robuste

Dans le cas général, on voit bien que ce problème est hautement combinatoire. En effet, nous devons, pour chaque combinaison d'informations, calculer la meilleure partition en utilisant ces informations. La complexité du RPP dépend de trois facteurs. Les deux premiers sont le nombre de tâches à ordonnancer (disons $n$), car il donne une borne supérieure du nombre d'informations disponibles à chaque moment de la décision, et $Q$ le nombre d'informations que nous sommes autorisés à utiliser à chaque moment de la décision. Le nombre de combinaisons possibles à un moment de la décision est limité par $\sum_{q=1}^{Q} \binom{n}{q}$. Le troisième facteur est la complexité du calcul de la valeur objective robuste min-max $\min_{s\in S} \max_{\omega\in\Omega} f(\omega, s)$ pour tout $\Omega$. Il est clair que si le problème déterministe est déjà difficile, sa variante robuste l'est aussi. Cependant, il existe des cas où un problème d'ordonnancement déterministe est polynomial, alors que sa variante incertaine est NP-Difficile. Comme dit en section 3.2, nous avons choisi le problème $1||L_{max}$ pour tester notre approche spécifiquement parce que sa variante robuste min-max est encore polynomiale. Dans les deux sous-sections suivantes, on propose deux méthodes distinctes pour

résoudre ce problème : une méthode exhaustive, et une autre basée sur un algorithme de *Branch and Bound*.

### 3.4.1 Méthode exhaustive

**Proposition 1.** *Si $f$ admet un scénario de pire cas global, c'est-à-dire qu'il existe un scénario $\omega^{wc}$ tel que : pour tout*

$$\omega' \in \Omega, \forall s \in S, f(\omega^{wc}, s) \geq f(\omega', s)$$

*alors, étant donné une partition $\mathscr{P}$ de $\Omega$ et un entier $L$, il est possible de calculer en temps polynomial une partition $\mathscr{P}'$ telle que $RS(\mathscr{P}')$ est minimale eparmi les partitions aui satisfont :*

*(i)* $|\mathscr{P}'| = \min(L, |\mathscr{P}|)$

*(ii) pour tout $P \in \mathscr{P}$ il existe $P' \in \mathscr{P}'$ de sorte que $P \subset P'$*

On peut noter que pour le problème $1||L_{max}$ , le pire scénario global est $\omega = (d_{i_{min}})_{i \in I}$.

*Démonstration.* Nous prouvons la proposition en montrant que toute partition renvoyée par l'Algorithme 3 vérifie les propriétés de la Proposition 1. Tout d'abord détaillons cet algorithme. Il prend en entrée une partition $\mathscr{P}$ de l'ensemble des scénarios $\Omega$ et un entier $L$. D'abord, on calcule pour chaque sous-ensemble $P \in \mathscr{P}$ la valeur objectif de la solution robuste sur $P$. Ensuite on crée une liste qui contient tous les sous-ensembles triés dans l'ordre croissant selon cette valeur objectif. Si la taille de cette liste, qui est $|\mathscr{P}|$, est plus petite que $L$ alors cette liste forme une partition qui est retournée. Sinon on remplace le $L$-ième sous-ensemble de cette liste par l'union de tous les sous-ensembles après lui et de lui-même. La liste étant de longueur $L$, elle est retournée par l'algorithme.
Retournons maintenant à la preuve. Par construction, une partition $\mathscr{P}'$ retournée par l'Algorithme 3 satisfait (i) et (ii). Nous devons maintenant prouver que $RS(\mathscr{P}')$ est minimal. Tout d'abord, nous pouvons observer que :

$$\min_{s \in S} \max_{\omega \in \bigcup_{j \in J} P_j} f(\omega, s) = \min_{s \in S} \max_{j \in J} \max_{\omega \in P_j} f(\omega, s)$$

pour toute famille d'ensembles disjoints $(P_j)_{j \in J}$. De cette observation on peut déduire que pour toute partition $P''$ qui satisfait (ii), les valeurs contenues dans $RS(\mathscr{P}'')$ sont nécessairement dans $RS(\mathscr{P})$. Ainsi pour toute autre partition $\mathscr{P}''$ qui vérifie (ii) on a, pour $i \leq L-1$,

$$RS(P')_i \leq RS(P'')_i \tag{3.1}$$

car, par construction, $RS(\mathscr{P}')_i$ est la i-ème plus petite valeur possible et les vecteurs $RS$ sont triés dans l'ordre croissant. De plus, puisque $f$ admet un pire scénario global $\omega^{wc}$, il existe $P' \in \mathscr{P}$ tel que $\omega^{wc} \in P'$. Ainsi, pour toute union de sous-ensembles de la forme $\bigcup_{P \in \mathscr{P}} p$ on a :

$$\min_{s \in S} \max_{\omega \in \bigcup_{P \in \mathscr{P}} P} f(\omega, s) = \min_{s \in S} \max_{\omega \in P'} f(\omega, s) = \min_{s \in S} f(\omega^{wc}, s)$$

Ainsi, pour toute partition $P''$ la dernière valeur dans $RS(P'')$ est nécessairement $\min_{s\in S} f(\omega^{wc}, s)$. Enfin, de ce résultat et de (3.1) nous concluons que $RS(P')$ est minimal. □

Un exemple d'exécution de l'algorithme 3 est donné dans l'exemple 5.

**Algorithme 3**

**Entrée:** $\mathscr{P} = [P_1, P_2, ..., P_j]$ une partition de $\Omega$ et un entier $L$.
  **pour** $i \leq j$ **faire**
    $RS[i] \leftarrow \min_{s\in S} \max_{\omega\in p_i} f(\omega, s)$
  **fin pour**
  **pour** $i \leq j$ **faire**
    $RS_S[i] \leftarrow RS[\phi(i)]$
    $\mathscr{P}_S[i] \leftarrow \mathscr{P}[\phi(i)]$
    où $\phi$ est la permutation qui trie $RS$ dans l'ordre croissant.
  **fin pour**
  **si** $|\mathscr{P}_S| > L$ **alors**
    $\mathscr{P}_S \leftarrow [\mathscr{P}_S[1], \mathscr{P}_S[2], ..., \mathscr{P}_S[L-1], \bigcup_{i=L}^{|\mathscr{P}_S|} \mathscr{P}_S[i]]$
  **fin si**
  **retourne** $\mathscr{P}_S$

**Exemple 5.** *Une fois encore, considérons l'instance de l'exemple 1. Supposons que* $t = 10$ *soit un moment de décision et que nous devions développer un nœud de l'arbre. Dans l'exemple 2, nous avons vu que* $K^{t=10} = \{k_{d_2}^{t=10}, k_{d_3}^{t=10}\}$ *avec* $k_{d_2}^{t=10} = 16$ *et* $k_{d_3}^{t=10} = 14$*. Nous pouvons diviser l'ensemble des scénarios en quatre sous-ensembles (comme le montre la figure 3.3). Appliquons l'algorithme 1 avec* $P = [A, B, C, D]$ *et* $L = 3$*. En gardant la même notation, nous avons* $RS = [7, 6, 4, 4]$*, puis* $RS_s = [4, 4, 6, 7]$ *et* $\mathscr{P}_s = [C, D, B, A]$*. Comme* $|P_s| > L$*, nous modifions* $\mathscr{P}_s$ *en* $\mathscr{P}_s = [C, D, A \cup B]$ *et son score de robustesse est de* $[4, 4, 7]$.

**Remarque 1 :** Notez que par définition de la forme de $\mathscr{P}$, toute union de rectangles est réalisable. En d'autres termes, même une forme non rectangulaire est acceptable. Par exemple, en considérant les notations de l'exemple 5, la partition $[A, B \cup C \cup D]$ est une solution réalisable.

Nous proposons maintenant l'Algorithme 4 pour résoudre le problème RPP. Il s'agit d'un algorithme exhaustif, qui appelle l'algorithme 3 pour chaque combinaison (inférieure à $Q$) d'informations, il a donc une complexité exponentielle.

Nous sommes maintenant capables de construire un arbre de décision robuste avec la procédure introduite dans la section 3.3.

### 3.4.2 *Branch and Bound*

On propose maintenant une seconde méthode, basée sur un algorithme de *Branch and Bound* pour explorer plus intelligemment l'ensemble des combinaisons d'information que ce que l'on fait dans l'Algorithme 4.

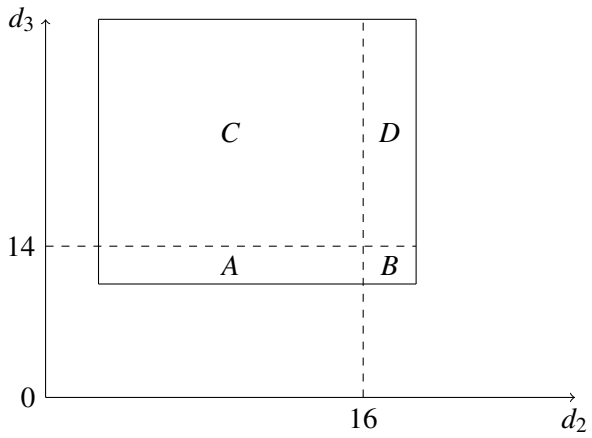



FIGURE 3.3 – Ensemble de scénarios pour l'Exemple 5

**Algorithme 4**

**Entrée:** Un ensemble de scénarios $\Omega$, un ensemble d'information $K^t$ et deux entiers $Q$ et $L$

$\mathscr{P}^* = \{\Omega\}$
**pour** $K \subseteq K^t$ **faire**
  **si** $|K| \leq Q$ **alors**
    $\mathscr{P}' = Algo3(P^{\Omega}_{K}, L)$
    **si** $RS(\mathscr{P}') \leq_{lexi} RS(\mathscr{P}^*)$ **alors**
      $\mathscr{P}^* = \mathscr{P}'$
    **fin si**
  **fin si**
**fin pour**
**retourne** $\mathscr{P}^*$

On se place à un moment décision $t$, et on suppose qu'on a un ensemble $K^t = \{k^t_1, k^t_2, ..., k^t_m\}$ d'informations. L'idée de ce *Branch and Bound* est de brancher sur l'ensemble $U$ de toutes les informations non utilisées, ou ignorées, et d'évaluer la première valeur du vecteur de score de robustesse de la partition renvoyée par l'algorithme 3 avec $\mathscr{P}^{\Omega}_{K^t \setminus U}$ en entrée. On verra que l'on peut faire cela en temps polynomial et sans appeler l'algorithme 3. Si cette valeur est moins bonne que celle de la meilleure solution trouvée jusque-là, on peut arrêter l'exploration dans le sous-arbre de recherche en dessous de ce noeud, sinon, on peut l'évaluer avec l'algorithme 3 et comparer la partition obtenue avec la meilleure partition trouvée jusqu'à présent. Avec cette manière de procéder, si l'on élimine assez tôt une information décisive, les partitions réalisables sans cette information seront de mauvaises qualités, et la branche où l'on élimine cette information sera coupée. On décrit maintenant plus précisément les différents éléments de ce *Branch and Bound*.

**Séparation :**

Le noeud racine de l'arbre est étiqueté avec l'ensemble $U = \emptyset$. Ce noeud a ensuite $m$ noeuds fils, étiquetés avec $U = \{k^t_1\}$, $U = \{k^t_2\}$, ... $U = \{k^t_m\}$. Puis, chacun de ces noeuds

a exactement $m-1$ fils, qu'on obtient en rajoutant un élément à $U$. On s'arrête lorsque $|U| = |K^t| - Q$, car à partir de ce moment-là, si l'on continuait à développer des noeuds l'ensemble d'information $K^t \setminus U$ ne serait plus un ensemble (car $|K^t \setminus U| = Q$).

**Évaluation :**

A un noeud donné (et donc étiqueté par un U donné), on relâche la contrainte qui force le nombre d'informations utilisées à être plus petit que $Q$, et on considère l'ensemble d'information $K' = K^t \setminus U$. On peut alors remarquer que si $E$ est un ensemble d'informations quelconque, et que l'on note $\omega_r(E)$ le scénario qui utilise toutes les informations de $E$ on a :

$$\min_{\substack{s\in S \\ E\subseteq K'}} \max_{\omega\in\Omega^E} f(s,\omega_r(E)) \leq \min_{\substack{s\in S \\ E\subseteq K' \\ |E|\leq Q}} \max_{\omega\in\Omega^E} f(s,\omega_r(E))$$

Le membre de droite de cette inégalité est égal à la première composante du vecteur de score de robustesse de la partition optimale utilisant au plus $Q$ informations parmi celles de $K'$. Cette quantité est facile à calculer puisqu'il suffit de résoudre le problème d'ordonnancement dans sa version déterministe. Ainsi, le membre de gauche qui est la valeur *minmax* utilisant toutes les informations de $K'$ est une borne inférieure de la meilleure (plus petite) première valeur du vecteur de score de robustesse de la partition optimale utilisant seulement les valeurs de $K'$. Puisque le vecteur de score de robustesse trié dans l'ordre lexicographique est notre critère pour comparer les partitions, cette propriété nous permet bien de couper dans l'arbre de recherche.

L' algorithme 5 décrit en pseudo-code le calcul de la meilleure partition en utilisant cette méthode.

On peut remarquer que cette méthode n'est pas exactement un algorithme de *Branch and Bound* à proprement parler, puisque, si le critère ne permet pas d'éliminer le noeud, on doit construire la partition $\mathscr{P}^{\Omega}_{K^t\setminus U}$ pour résoudre le problème en ce noeud. Il s'en inspire simplement pour accélérer la recherche arborescente.

## 3.5 Expérimentations

L'objectif des expériences réalisées est d'évaluer la robustesse de notre modèle d'arbre de décision robuste, la qualité des informations sélectionnées utilisées pour sa construction et sa stabilité en termes de nombre de réactions. Pour les tests numériques, nous avons généré différents types d'instances pour le problème $1||L_{max}$ avec incertitude sur les dates d'échéance en fonction de ce qu'on nomme la classe de l'instance. Nous distinguons deux classes : la classe $A$ avec de petits intervalles d'incertitude, et la classe $B$ avec de grands intervalles d'incertitude. Le second paramètre est le nombre de tâches à planifier. Pour chacune de ces deux classes, nous avons généré des instances avec 50 tâches. Ainsi, l'instance $A_{50}$ est une instance avec 50 tâches, avec de petits intervalles d'incertitude sur les dates d'échéance des tâches.

Comme notre approche utilise la notion d'information pour réduire les incertitudes et fournir une solution plus robuste, nous la comparons à un algorithme proactif-réactif plus standard. Cet algorithme prend deux paramètres en entrée (en plus de l'instance), un taux

**Algorithme 5**

**Entrée:** Un ensemble de scénarios $\Omega$, un ensemble d'information $K^t$ et deux entiers $Q$ et $L$

$\mathscr{P}^* \leftarrow \{\Omega\}$
$v^* \leftarrow +\infty$
$\mathscr{Q} \leftarrow \emptyset$
**tant que** $\mathscr{Q} \neq \emptyset$ **faire**
  $U \leftarrow pop(\mathscr{Q})$
  **pour** $k \in K^t$ tel que $k \notin U$ **faire**
    $K' \leftarrow K^t \setminus U$
    **si** $\min_{\substack{s \in S \\ E \subseteq K'}} \max_{\omega \in \Omega^E} f(s, \omega(E)) \leq v^*$ **alors**
      $\mathscr{P} \leftarrow Algo1(P^{\Omega}_{K'}, L)$
      **si** $RS(\mathscr{P}') \leq_{lexi} RS(\mathscr{P}^*)$ **alors**
        $\mathscr{P}^* \leftarrow \mathscr{P}'$
        $v^* \leftarrow$ premier élément de $RS(\mathscr{P}')$
        $\mathscr{Q} \leftarrow \mathscr{Q} \cup \{U\}$
      **fin si**
    **fin si**
  **fin pour**
**fin tant que**
**retourne** $\mathscr{P}^*$

de réaction $\rho_r \in [0,1]$ et un taux d'information $\rho_i \in [0,1]$. Le principe de l'algorithme est le suivant. Nous commençons l'exécution de l'ordonnancement par un ordonnancement robuste au sens du critère min max. A la fin de chaque tâche, l'algorithme réagit avec une probabilité de $\rho_r$. Lorsqu'il réagit, il calcule une nouvelle solution robuste en utilisant au maximum $100\rho_i\%$ de l'information disponible. La définition d'une information et la manière dont elle est accessible sont strictement les mêmes que celles utilisées pour construire un arbre de décision robuste. Pour construire un arbre de décision robuste avec notre méthode, nous avons besoin de quelques paramètres : une liste de moments de décision $(t_j)_{j \in J}$, le nombre maximal d'informations que nous sommes autorisés à utiliser à chaque moment de décision $Q$, et la taille maximale de la partition que nous calculons à chaque moment de décision $L$. Afin de tester différentes listes de moments de décision, nous avons divisé la durée totale de l'ordonnancement en $T$ intervalles égaux. Dans notre expérience, nous avons calculé des arbres de décision robustes en utilisant les valeurs suivantes :

- Le nombre de moments de décisions $T \in [2,..,10]$.
- Le nombre maximal d'informations à chaque moment de décision $Q \in [1,..,10]$.
- La taille maximale de la partition à chaque moment de décision $L \in [2,..,4]$.

Ainsi, un arbre calculé avec ces paramètres est désigné par $\text{Tree}_{Q,L,T}$. Ces valeurs peuvent sembler faibles, mais comme nous l'avons vu précédemment, ces paramètres ont un impact non seulement sur le temps de calcul pour résoudre le RPP, mais aussi sur la taille de l'arbre. Afin de garder le temps de calcul total d'un arbre raisonnable, nous avons dû restreindre la taille de ces valeurs.

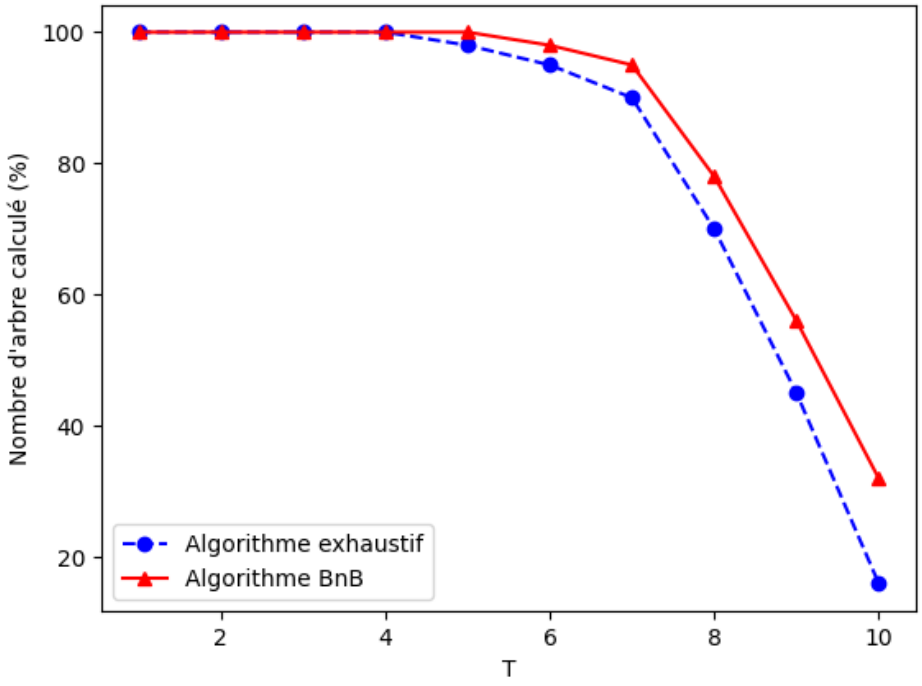



FIGURE 3.4 – Nombre d'arbres calculés dans le temps limite d'une heure

Comme nous l'avons vu précédemment, l'algorithme proactif-réactif prend deux paramètres, $\rho_i$ et $\rho_r$. L'exécution de cet algorithme étant rapide à simuler, nous énumérons pour $\rho_i$ et $\rho_r$ toutes les valeurs dans [0, 1] avec des pas de 0,1.

Pour chaque instance, nous choisissons aléatoirement 500 scénarios (avec une distribution uniforme). Ensuite, pour chaque ensemble de paramètres et chaque instance, nous calculons un arbre de décision robuste, et nous parcourons l'arbre en suivant le chemin correspondant à chaque scénario aléatoire. Le même protocole est utilisé avec les algorithmes réactifs robustes, nous les simulons sur chaque scénario aléatoire.

### 3.5.1 Efficacité du *Branch and Bound*

Bien que le temps de calcul ne soit pas un de nos critères, nous avons vérifié que notre algorithme basé sur le *branch and bound* permettait en pratique de calculer les arbres plus rapidement. Pour cela, nous avons, en fonction du paramètre $T$ regardé la proportion d'arbres calculés dans la limite de temps d'une heure en utilisant chacune des deux méthodes. Pour être plus précis, la proportion signifie le nombre d'arbres effectivement calculés par rapport au nombre total d'arbres pour chaque $T$, c'est-à-dire $3 * 10 * T$. Les résultats sont présentés sur la figure 3.4.

On observe que les résultats des deux méthodes sont similaires jusqu'à $T = 4$, à savoir que tous les arbres possibles sont effectivement calculés dans la limite de temps. En

revanche, à partir de $T = 5$ l'écart entre les deux méthodes se creuse au fur et à mesure que $T$ augmente, jusqu'à $T = 10$, pour lequel la construction de l'arbre avec la méthode exhaustive ne parvient qu'à calculer 16% des arbres alors que celles basée sur le *branch and bound* en résout 32 %. Puisque cette seconde méthode permet d'accélérer le calcul de l'arbre, ce sera celle choisie dans le reste des expérimentations numériques de ce chapitre.

### 3.5.2 Mesure de performances

Nous évaluons maintenant la pertinence des informations utilisées pour construire l'arbre, et la stabilité de la planification proposée par l'arbre. Pour cela, nous avons collecté, pour chaque instance, le nombre moyen (sur les 500 scénarios aléatoires) d'informations utilisées, le nombre moyen de réactions (lorsqu'une solution globale change entre deux moments de décisions) et la distance à l'optimum, qui est, pour un scénario donné $\omega$ et une solution $s$ :

$$100 \cdot \frac{L_{max}(\omega, s) - L_{max}(\omega, s^*)}{L_{max}(\omega, s^*)}$$

où $s^*$ est la solution optimale minimisant sa valeur $L_{max}$ sur le scénario $\omega$. Les instances ont été générées de sorte que $L_{max}(\omega, s^*)$ ne soit jamais mul.
Nous considérons deux couples de critères : (distance par rapport à l'optimum, nombre de réactions) et (distance par rapport à l'optimum, nombre d'informations), et pour les deux, nous traçons deux frontières de Pareto, une pour l'algorithme réactif robuste et une pour l'arbre de décision robuste. Certains des résultats sont présentés sur les figures et . Étant donné que la plupart des arbres n'ont pas pu être calculés avant la limite de temps, il y a moins de points pour les arbres de décision, par exemple $A_{50}$.

Pour les instances $A_{50}$, il est un peu difficile de conclure puisque le front de Pareto est de petite taille, mais on observe que les meilleurs arbres de décision sont plus performants que les meilleurs algorithmes réactifs en termes de nombre de réactions, et au moins aussi bons en termes de nombre d'informations. En revanche, sur les instances $B_{50}$ alors que les deux méthodes ont des résultats similaires -voire légèrement moins bons pour les arbres- lorsque qu'il y a peu de réactions ou que peu d'informations sont utilisées, mais il y a une différence significative dans le cas contraire, et les arbres de décisions robustes semblent plus intéressants. Plus généralement, nous observons que les arbres de décision robustes fournissent de meilleures solutions (pour un nombre donné de tâches) lorsque les intervalles d'incertitude sont plus grands. Intuitivement, cela peut être expliqué par le fait que les arbres de décision sont très contraints par les moments de décision alors que les algorithmes réactifs ne le sont pas. Ainsi, des incertitudes plus grandes permettent aux arbres de décision robustes d'acquérir plus de nouvelles informations que les algorithmes réactifs robustes.

Cependant, cette façon de présenter les résultats ne permet pas de savoir quels arbres de décision robustes apparaissent dans la frontière de Pareto. Le tableau 1 montre quels arbres (et l'ensemble des paramètres avec lesquels ils ont été calculés) sont les points extrêmes des différentes frontières de Pareto. Assez intuitivement, les arbres de décision proposent les meilleures solutions avec peu d'informations et de réactions, lorsque le nombre de moments de décision est faible et que le nombre maximum d'informations utilisables est élevé.

| | Reaction | | Information | |
|---|---|---|---|---|
| | extreme 1 | extreme 2 | extreme 1 | extreme 2 |
| $B_{50}$ | $\text{Tree}_{10,4,2}$ | $\text{Tree}_{10,4,6}$ | $\text{Tree}_{10,4,2}$ | $\text{Tree}_{2,4,6}$ |
| $A_{50}$ | $\text{Tree}_{10,4,3}$ | $\text{Tree}_{3,4,6}$ | $\text{Tree}_{2,4,3}$ | $\text{Tree}_{2,4,6}$ |

TABLE 3.1 – Arbres de décision robustes correspondant aux points extrême des fronts de Pareto montrés dans les figures 3.5 et 3.6.

Dans le même ordre d'idées, les arbres de décision construits avec un nombre élevé de moments de décision (c'est-à-dire les plus profonds) mais avec peu d'informations disponibles à chaque moment donnent également de bonnes solutions. Il est intéressant de noter que cela montre que la profondeur de l'arbre est plus importante pour produire des solutions de qualité que le nombre maximum d'informations. Le tableau montre également que la taille de partition maximale a été utilisée pour les meilleurs arbres, ce qui implique que les résultats pourraient être améliorés en augmentant cette valeur.

## 3.6 Conclusion

Dans ce chapitre, nous avons présenté une approche proactive-réactive pour traiter les problèmes d'ordonnancement incertains. La méthode construit un arbre de décision robuste pour un décideur qui est réutilisable tant que les paramètres du problème appartiennent à l'ensemble d'incertitudes. À chaque nœud de l'arbre, nous supposons que le planificateur a accès à certaines connaissances sur le scénario en cours, ce qui réduit le niveau d'incertitude et permet le calcul de solutions moins pessimistes avec des garanties de robustesse. Cependant, l'obtention d'informations sur les paramètres incertains peut être coûteuse et le réordonnancement fréquent peut être perturbant. Nous avons d'abord défini formellement l'arbre de décision robuste et les concepts de raffinement de l'information dans le contexte des scénarios d'incertitude. Nous avons ensuite introduit le problème de partition robuste, le problème central de notre approche. Nous avons ensuite proposé des algorithmes pour résoudre ce problème et construire un tel arbre. Enfin, en nous concentrant sur un problème simple d'ordonnancement à une machine, nous avons fourni des comparaisons expérimentales mettant en évidence le potentiel de l'approche de l'arbre de décision par rapport aux algorithmes réactifs pour obtenir des solutions plus robustes avec moins de mises à jour d'informations et de changements d'ordonnancement. Au vu des résultats encourageants, nous pensons que cette méthode pourrait être utilisée dans des cas industriels. Aussi dans le chapitre suivant, nous étendons cette méthode à un problème industriel se formulant comme un problème d'ordonnancement de projet sous contrainte de ressources et modes multiples.

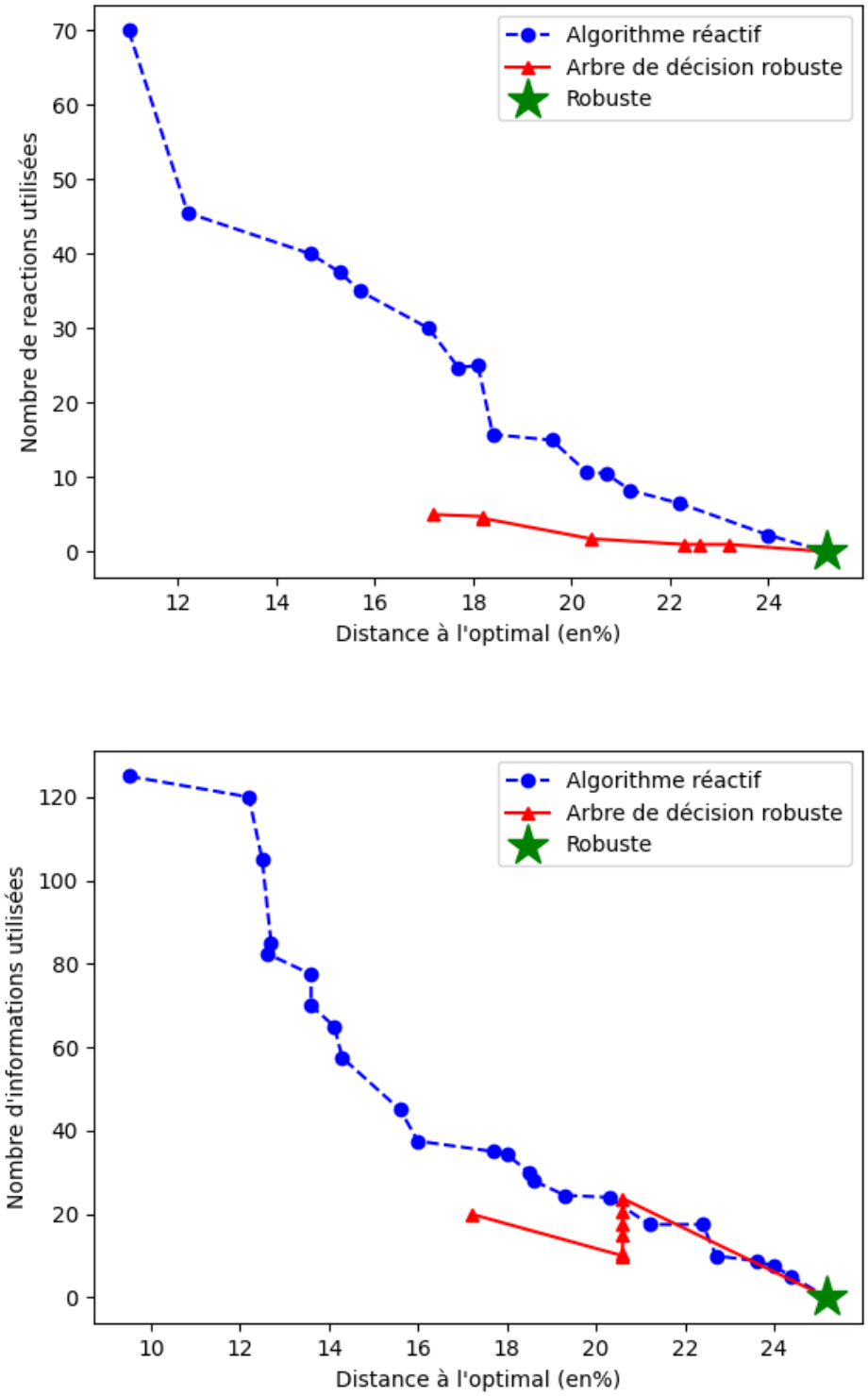



FIGURE 3.5 – Front de Pareto pour les deux critères sur les instances $A_{50}$

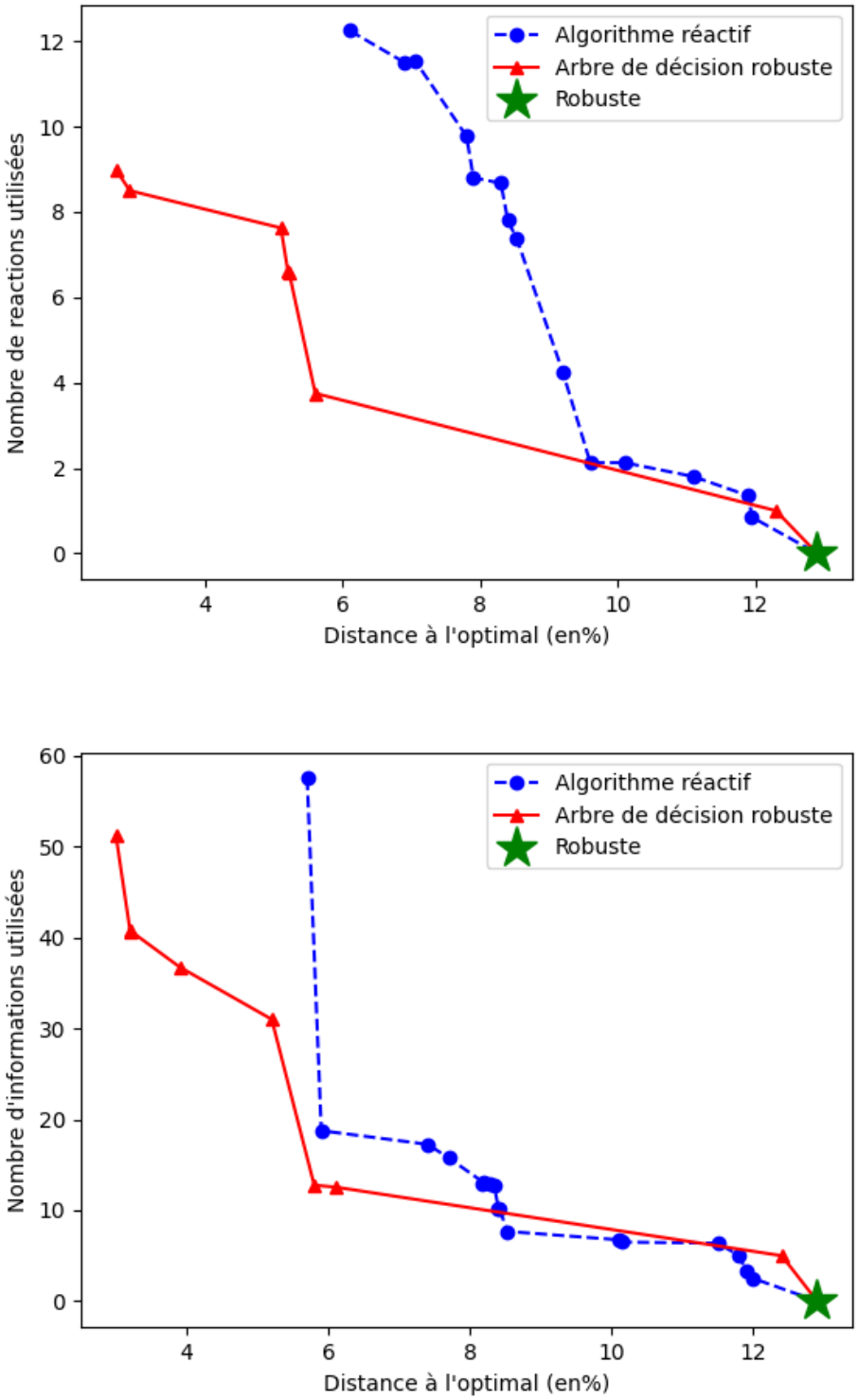



FIGURE 3.6 – Front de Pareto pour les deux critères sur les instances $B_{50}$

CHAPITRE 4

# Arbres de décision robustes pour les lignes d'assemblage aéronautiques

**Sommaire**



L'ensemble des résultats de ce chapitre fait l'objet d'un article soumis au journal *European Journal of Operational Research*. Une version libre de ce papier est disponible dans HAL (voir [Portoleau 2021d]). Ces travaux ont été également présentés aux conférences ROADEF 2021 [Portoleau 2021a] et EURO 2021 [Portoleau 2021c].

## 4.1 Introduction

Nous implémentons dans ce chapitre le modèle des arbres de décision robustes définis dans le chapitre 3 sur le problème d'ordonnancement de chaîne de montage aéronautique considéré dans le chapitre 2 en introduisant des incertitudes sur les durées des tâches. Nous comparons ensuite notre modèle à un algorithme proactif-réactif à l'aide d'expériences de calcul réalisées sur des instances de référence MMRCPSP de la PSPLIB [Kolisch 1997b] et les instances industrielles réelles du problème d'ordonnancement de chaîne de montage aéronautique.

Le plan de ce chapitre est le suivant. La section 2 rappelle le contexte industriel de notre approche. Dans la section 3, nous approfondissons la revue de la littérature concernant les techniques d'ordonnancement multi-étapes dont notre approche s'inspire. La section 4 introduit formellement le MMRCPSP, l'incertitude et la modélisation de l'information. Dans la section 5, nous présentons un modèle basé sur l'information du MMRCPSP et formulons le problème de partition robuste, qui est le problème central de notre approche. La section 6 introduit l'arbre de décision robuste. Dans la section 7, nous évaluons l'intérêt pratique de notre approche en la comparant à un algorithme d'ordonnancement proactif-réactif basé sur l'information.

Comme vu dans le chapitre 2, le problème consiste à construire un ordonnancement qui satisfait un makespan fixe tout en minimisant le pic d'utilisation des ressources, qui correspond au nombre d'opérateurs à employer pendant la période d'ordonnancement. Dans ce chapitre, nous considérons des aléas modérés, qui n'ont d'impact que sur la durée des activités. Étant donné un planning initial, toute augmentation de la durée des activités peut dégrader la qualité de ce planning. Si l'on connaît la durée maximale de chaque activité, un ordonnancement initial robuste peut être construit comme dans le schéma standard d'optimisation robuste, c'est-à-dire en recherchant une bonne performance sur le pire scénario, dans ce cas obtenu en fixant chaque activité à sa durée maximale.

Sur une chaîne de montage, le même problème d'ordonnancement avec de petites variations se répète à chaque fois (au fur et à mesure que des avions similaires sont assemblés) alors que les risques changent constamment, ce qui peut faire fluctuer fortement la qualité de l'ordonnancement robuste initial au cours de ces cycles de montage. De plus, surestimer les besoins de certaines ressources, comme cela est nécessaire dans l'optimisation robuste traditionnelle, peut s'avérer très coûteux. C'est le problème bien connu du conservatisme de l'optimisation robuste standard.

Une autre caractéristique importante du contexte industriel considéré est que les décideurs n'ont souvent accès aux informations sur le déroulement de la planification qu'à des moments spécifiques, par exemple, chaque soir avant un changement d'équipe, lorsque l'équipe sortante rapporte ses actions de la journée. À ce moment précis, les décideurs peuvent utiliser leur savoir-faire pour adapter rapidement le planning à la volée. Cependant, avoir accès à ces informations peut être coûteux, et toutes les informations peuvent ne pas être particulièrement pertinentes. De plus, le décideur peut ne pas vouloir changer le planning trop souvent pour assurer une certaine stabilité dans la planification des opérateurs. Pour ces raisons, on souhaite rechercher les informations les plus pertinentes, c'est-à-dire celles qui, si elles sont prises en compte pour la reprogrammation, amélioreraient le mieux les performances dans le pire des cas.

| Tâches | $i$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Durée maximale | $p_i^{\max}$ | 6 | 10 | 2 | 4 |
| Opérateur 1 | $b_{i,1}$ | 1 | 1 | 3 | 0 |
| Opérateur 2 | $b_{i,2}$ | 1 | 2 | 0 | 4 |

TABLE 4.1 – Une petite instance de RCPSP

**Exemple 6.** *Pour illustrer ce point, considérons une instance fictive d'un problème d'ordonnancement de projet (monomode) sous contraintes de ressources (RCPSP) : on nous donne un ensemble d'activités $\mathscr{J}$, un makespan $C_{\max}$ et deux points de décision $t_0 = 0$ et $t_1$. Il existe un ensemble de ressources renouvelables $\mathscr{R}$ de capacités limitées et chaque activité $i \in \mathscr{J}$ nécessite pendant son traitement une quantité donnée non négative $b_{i,k}$ de chaque ressource $k \in \mathscr{R}$ (voir les données d'instance dans le tableau 4.1). Dans cet exemple, nous avons quatre activités et deux ressources, chacune correspondant à un type d'opérateur (par exemple, électricien et mécanicien). L'activité 1 nécessite un opérateur de chaque type pendant son traitement, c'est-à-dire pendant au plus 6 heures. Notre objectif est de trouver un planning qui se termine avant $C_{\max}$ tout en minimisant la somme des utilisations maximales des ressources de $t_0$ à $t_1$ et de $t_1$ à la fin du planning. La raison d'être de cet objectif est que nous supposons que les opérateurs sont engagés en fonction de l'utilisation maximale des ressources du planning calculé pendant l'intervalle défini par deux points de temps de décision consécutifs. Considérons maintenant un makespan fixe $C_{\max} = 12$ et, en plus, les contraintes de précédence $1 \to 3$ et $1 \to 4$. Un planning $s$ est l'affectation d'heures de début entières $S_i$ dans $[0, C_{\max} - 1]$ à chaque activité $i \in \mathscr{J}$. Un scénario de durée est un vecteur $p = (p_i)_{i \in \mathscr{J}}$ avec $p_i \in [p_i^{\min}, p_i^{\max}] \cap \mathbb{N}$ où $p_i^{\min} = 1$ et $p_i^{\max}$ sont des paramètres connus. Un ordonnancement est réalisable pour un scénario $p$ s'il satisfait les contraintes de précédence $S_3 \geq S_1 + p_1$ et $S_4 \geq S_1 + p_1$. Pour un scénario $p$, l'utilisation de la ressource $k \in \mathscr{R}$ au temps $t \in [0, C_{\max} - 1]$ par l'ordonnancement $S$ est donnée par :*

$$R_{k,t} = \sum_{i \in \mathscr{J}} \sum_{t=S_i}^{S_i + p_i - 1} b_{i,k}$$

*Un ordonnancement optimal pour le scénario $p$ est celui qui minimise la fonction objectif*

$$z = \max_{t \in [0, t_1 - 1]} [R_{1,t} + R_{2,t}] + \max_{t \in [t_1, C_{\max} - 1]} [R_{1,t} + R_{2,t}]$$

*L'ordonnancement optimal $s^0$ pour le scénario $p = (p_i^{\max})_{i \in \mathscr{J}}$, avec une valeur objective de 13, est donné par les diagrammes de Gantt affichés sur les figures 4.1 et 4.2 :*

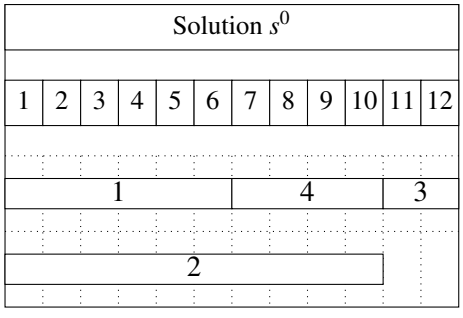



FIGURE 4.1 – Diagramme de Gantt de la solution $s^0$

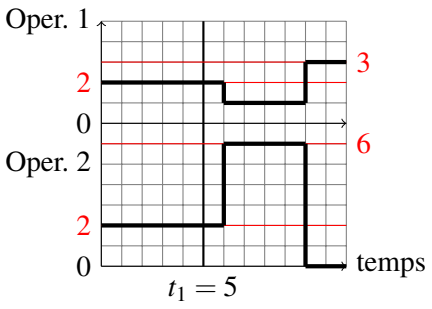



FIGURE 4.2 – Profil d'utilisation des ressources de la solution $s^0$

*Nous supposons que le planning $s^0$ a été calculé avant le temps $t_0 = 0$ où aucune*

*information sur l'état de la planification n'était disponible puisque le planning n'a pas été lancé. Le planning est suivi tel quel de $t_0$ à $t_1 - 1$ et les 2 opérateurs de chaque type sont engagés dans cet intervalle. Supposons maintenant qu'au temps $t_1 = 5$ le décideur ait l'opportunité de poser deux questions pour augmenter la connaissance des paramètres $p^{\min}$ et $p^{\max}$, mais se demande laquelle est la plus pertinente : A) "L'activité 1 est-elle déjà terminée ? (i.e. $p_1^{\max} \leq 5$)" ou B) "L'activité 2 durera-t-elle plus ou moins de 8 unités de temps ? (i.e. $p_2^{\min} \geq 8$ ou $p_2^{\max} \leq 8$)". Examinons les deux cas :*

- *Le décideur choisit de poser la question A). Alors, quelle que soit la réponse à cette question, nous ne pouvons pas calculer une meilleure solution, car même si la réponse est oui, cette nouvelle information permet seulement au décideur de déplacer les activités 3 et 4 vers la droite, ce qui n'améliore pas l'objectif.*
- *Le décideur choisit de poser la question B). Si la réponse à cette question est oui, le décideur peut passer à $t_1$ à la solution $s^1$ (voir Fig. 4.3 et 4.4), qui a une valeur objective de 12 ce qui en fait une solution strictement meilleure que $s^0$. Remarquons que pour mettre en œuvre une telle solution, l'ordonnancement (et l'utilisation des opérateurs) doit bien sûr être maintenu inchangé (gelé), c'est-à-dire dans $[t_0, t_1 - 1]$ indépendamment de l'information obtenue à $t_1$ sur ce qui s'est passé dans cet intervalle. C'est pourquoi l'horaire avant le temps $t_1 = 5$ est affiché en gris.*

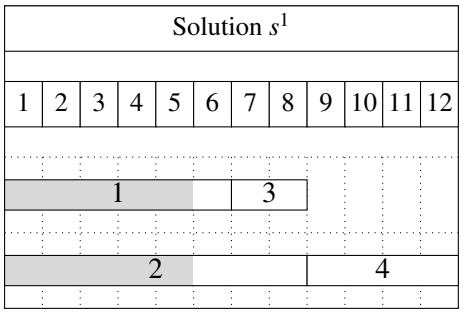



FIGURE 4.3 – Diagramme de Gantt de la solution $s^1$

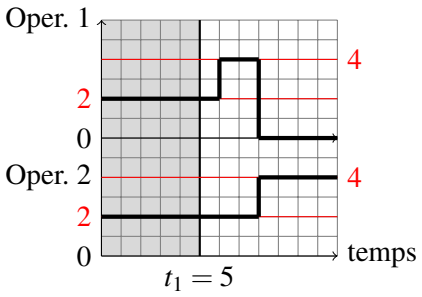



FIGURE 4.4 – Profil d'utilisation des ressources de la solution $s^1$

On voit bien dans cet exemple que la réponse à la question A) n'est d'aucune utilité, contrairement à la réponse à la question B). Du point de vue du décideur, ce type d'information est intéressant, car il suggère que s'il est possible de contrôler l'activité 2 en particulier pour qu'elle se déroule sans problème, alors il est possible d'améliorer le planning. Notons que nous nous intéressons ici uniquement à la sélection des questions à partir d'un ensemble fixe.

Dans ce cas, en sélectionnant à l'avance la question B) à poser à $t_1$, on peut définir un arbre de décision. Au nœud racine correspondant au temps $t_0$, les activités 1 et 2 sont lancées. Ce nœud a un nœud enfant unique correspondant à l'instant $t_1$. À partir de ce nœud, une branche correspondant à la réponse négative à la question B) continuera à suivre le programme $s^0$ tandis qu'une autre branche correspondant à la réponse positive à la question B) passera au programme $s^1$.

Dans l'hypothèse où l'on dispose à un moment donné du planning d'une information sur l'avancement de la planification et que l'on peut modifier le planning, un tel arbre de décision calculé à l'avance permet aux décideurs de retrouver rapidement des solutions de haute qualité tout en utilisant le moins d'information possible.

Dans la suite de ce chapitre, nous considérerons que la question posée par le décideur sur la durée d'une activité est : "Cette activité durera-t-elle moins de 80% de sa durée estimée ?". Le 80% est arbitraire et n'a pas d'impact majeur sur nos résultats expérimentaux. Une autre hypothèse arbitraire sur la nature des informations reçues avait été choisie au chapitre précédent (voir §3.2.2).

La méthode que nous présentons dans ce chapitre reprend l'idée des arbres de décision robustes présentés dans le chapitre précédent, que l'on peut voir comme un cas particulier d'optimisation robuste multi-étape basé sur le calcul de partition mais avec des différences notables que nous illustrons à nouveau à l'aide de l'exemple 6.

À l'instant de décision $t_1 = 5$, une approche d'optimisation robuste ajustable à plusieurs étapes se demanderait comment partitionner l'ensemble des scénarios de manière à proposer des réactions qui réduisent le nombre d'opérateurs requis dans le pire des cas. Supposons que le choix soit entre les partitions A) et B), comme déjà décrit. Aucune de ces partitions n'entraînerait une diminution du pire cas global. Dans le cas A), qui divise l'ensemble des scénarios en deux sous-ensembles définis par $p_1 \in [1,5]$ d'une part et $p_1 = 6$ d'autre part, nous avons vu que le pire cas restera inévitablement égal à 13 pour les deux sous-ensembles. Le cas B) divise l'ensemble des scénarios en deux sous-ensembles définis par $p_2 \in [1,8]$ d'une part et $p_2 \in [8,10]$ d'autre part. Même si le premier sous-ensemble réduit le nombre d'opérateurs le plus défavorable à 12, le second sous-ensemble inclut toujours le scénario le plus défavorable avec toutes les durées égales à leur valeur maximale, de sorte que l'ordonnancement global le plus défavorable reste égal à 13.

Cependant, la partition B) est clairement meilleure d'un point de vue pratique car le passage à $s^1$ dès que la réalisation de la durée se révèle -à $t_1$- appartenir au deuxième sous-ensemble permettra d'économiser des ressources. Pour cette raison, nous allons utiliser le critère présenté dans le chapitre précédent pour comparer les partitions de scénarios entre elles. Il n'est pas axé sur l'exploitation des informations obtenues pour abaisser le pire cas global mais plutôt sur l'identification des informations qui permettront de basculer de manière opportuniste vers une meilleure solution lorsque cela est possible et ainsi réduire le prix de la robustesse en pratique.

Deuxièmement, la notion de "coût de la connaissance" est parfois considérée dans la littérature, mais lorsque c'est le cas, elle est introduite dans la fonction objectif du problème de partitionnement comme une pénalité. Nous avons préféré l'ajouter comme deuxième objectif, d'un point de vue lexicographique. En effet, nous sommes plus intéressés à éviter de réagir à des informations inutiles qu'à considérer les coûts réels de l'information.

Enfin, dans la majorité des travaux, les problèmes incertains multi-étapes sont étudiés sous l'angle de la programmation mathématique. Dans ce chapitre, motivés par un contexte industriel spécifique -les chaînes de montage aéronautiques-, nous proposons une approche basée sur la programmation par contraintes que nous pouvons étendre naturellement pour intégrer le problème de sélection d'information.

Notre travail s'inspire en partie du problème d'ordonnancement de projet proactif et ré-

actif sous contrainte de ressources considéré dans [Davari 2017, Davari 2019] (voir §1.2.4) dans le contexte de durées incertaines. Les auteurs de ces travaux proposent de définir un planning de base et un ensemble de réactions face à une réalisation des durées. Une réaction est une transition d'un planning à un autre. Les durées des activités suivent une distribution discrète et les transitions ont lieu à l'intérieur d'un ensemble de calendriers précalculés. L'objectif est de minimiser le coût attendu des réactions compte tenu des distributions. Le coût mesure les coûts fixes des réactions ainsi que la distance entre la ligne de base et le programme réalisé. Dans ce document, nous supposons que les distributions de probabilité sur les durées des activités ne sont pas disponibles conformément aux applications aéronautiques ciblées, mais nous connaissons le pire scénario. Par conséquent, nous pouvons suivre un calendrier de base du pire des cas et réagir pour l'améliorer pour certaines réalisations en utilisant le moins d'information possible, minimisant ainsi indirectement la stabilité du calendrier.

Le modèle général que nous proposons a déjà été présenté, de manière formelle, dans le chapitre 2 et appliqué à un problème simple d'ordonnancement à une machine. Cependant, dans ce travail, nous avons traité uniquement le cas où le problème en question peut être résolu en temps polynomial, même dans sa version incertaine. Comme le MMRCPSP est un problème NP-difficile même dans sa forme déterministe, l'algorithme de base de l'approche est différent de celui présenté dans nos travaux précédents. Si l'idée principale de l'arbre de décision robuste reste la même, l'objectif du présent chapitre est de montrer comment il peut être appliqué à un problème industriel.

## 4.2 Incertitudes, informations et ordonnancement

### 4.2.1 Incertitude, robustesse et information

Dans ce chapitre et pour le problème considéré, nous supposons que la durée des activités est incertaine. Comme mentionné précédemment, le but de notre modèle est de prendre en compte les incertitudes petites et fréquentes, qui se produisent à chaque répétition du problème. De plus, comme la construction d'un modèle probabiliste précis nécessite une grande quantité de données, il est plus facile et plus pratique de demander au décideur des bornes sur la durée des activités. Pour ces raisons, nous considérons l'incertitude d'intervalle. Plus formellement, cela signifie que pour toute activité $i \in \mathscr{J}$ du problème, sa durée $p_i \in [p_i^{min}, p_i^{max}]$. Nous appelons toute réalisation de toutes les incertitudes un scénario. Nous désignons par $\Omega$ l'ensemble de tous les scénarios possibles et $\omega$ l'un quelconque de ses éléments. Nous pouvons alors écrire le critère de robustesse que nous considérons dans ce chapitre, introduit comme le critère de robustesse absolue dans [Kouvelis 1997] :

$$s^* = \arg\min_{s \in \mathscr{X}} \max_{\omega \in \Omega} f(\omega, s) \tag{4.1}$$

où $\mathscr{X}$ est l'ensemble des solutions, et $f(\omega, s)$ la valeur objective de la solution $s$ sous le scénario $\omega$. Nous nous référons à $s^*$ (resp. à la valeur maximale de $f(\omega, s^*)$) comme étant la solution minmax ou robuste (resp. la valeur minmax ou robuste) sur l'ensemble des

scénarios $\Omega$.
Nous pouvons remarquer que dans notre cas, la solution $s^*$ est la solution la plus robuste. On peut également observer que la fonction objectif que nous considérons est croissante avec la durée des activités. Ainsi, pour trouver une solution robuste pour le problème MMRCPSP avec incertitudes, il suffit de le résoudre dans son pire scénario, où toutes les durées sont à leur valeur maximale.

Maintenant que les incertitudes sont clairement énoncées, nous pouvons définir rigoureusement ce que nous appelons information. Il existe un ensemble $T$ de points de temps prédéfinis (par exemple les fins d'équipes), auxquels le décideur peut poser plusieurs questions, la réponse à une question apportant une information sur une activité. Soit $\mathscr{J}_q$ l'ensemble des activités qui sont en cours de traitement pendant l'intervalle $[t_q, t_q + \Delta] \subset \mathscr{T}$ dans le planning de base, c'est-à-dire

$$\mathscr{J}_q = \{i \in \mathscr{J} \,|S_i - \Delta \leq t_q < S_i + p_i\}$$

.

Dans le modèle d'information considéré, nous supposons que seules les questions sur les activités dans $\mathscr{J}_q$ sont autorisées. Le raisonnement derrière cela est que si une activité est terminée au temps $t_q$ dans le calendrier de base, elle est nécessairement terminée dans le scénario réalisé, car le pire cas a été considéré pour construire le calendrier. Si le début d'une activité n'est pas programmé dans l'intervalle $[t_q, t_q + \Delta]$, nous supposons que le décideur ne dispose pas d'un retour d'information suffisant de la chaîne de montage pour obtenir des informations fiables sur cette activité.

Dans notre modèle, avoir accès à une information sur une activité $i$ signifie que le décideur est capable de connaître la réponse à la question "L'activité $i$ dure-t-elle plus ou moins de $X_i$ unité de temps ?", avec $X_i \in [p_i^{min}, p_i^{max}]$. Par conséquent, une information nous permet de séparer en deux l'ensemble des scénarios : les scénarios tels que $p_i \in [p_i^{min}, X_i]$ et les scénarios tels que $p_i \in [X_i, p_i^{max}]$

Soit $K \subseteq \mathscr{J}_q$ un ensemble d'activités sur lesquelles des questions peuvent être posées à l'instant $t_q$. Le fait de disposer des réponses à $|K|$ questions, c'est-à-dire d'avoir accès à $|K|$ informations, permet au décideur de distinguer entre $2^{|K|}$ sous-ensembles de scénarios correspondant aux $2^{|K|}$ sous-ensembles possibles d'activités obtenant une réponse "oui". Soit $K_+ \subseteq K$ l'un des sous-ensembles possibles de réponses positives. Nous désignons par $\Omega_{K_+}$ le sous-ensemble de scénarios issus de l'utilisation des informations obtenues de $K_+$ pour réduire l'incertitude. Plus formellement,

$$\Omega_{K_+} = \prod_{i \in K_+} [p_i^{min}, X_i] \times \prod_{i \in \mathscr{J} \setminus K_+} [p_i^{min}, p_i^{max}]$$

Dans notre cas, la valeur objective la plus défavorable peut toujours être obtenue lorsque chaque activité a sa durée maximale, avoir pour une activité $i$ l'information que $p_i \in [X_i, p_i^{max}]$ ou n'avoir aucune information sur $i$ revient au même car cela ne change pas le pire cas. On pourrait noter qu'en utilisant ces notations, on a que si $K_+$ et $K'_+$ désignent deux ensembles d'informations sur les incertitudes sur un ensemble

de scénarios $\Omega$ alors $K_+ \subset K'_+ \Leftrightarrow \Omega_{K'_+} \subset \Omega_{K_+}$ et, plus généralement, $\Omega_{K_+ \cup K'_+} = \Omega_{K_+} \cap \Omega_{K'_+}$.

Dans les exemples et expérimentations, nous prendrons $\Delta = 0$ et $X_i = \lceil 0.8 p_i^{max} \rceil$.

**Exemple 7.** *Illustrons ces notions sur l'exemple 6 au temps $t_1 = 5$ et le planning de base $s^0$. Le décideur est autorisé à poser des questions sur $\mathscr{J}_1 = \{1,2\}$ car ces deux activités sont en cours de traitement au temps $t_1$ selon $s^0$. En prenant comme mentionné $X_i = \lceil 0.8 p_i^{\max} \rceil$, on obtient $X_1 = 5$ et $X_2 = 8$. Prenons $K = \{1,2\}$. En considérant les sous-ensembles possibles de réponses positives $K_+ = \{1\}$, $K'_+ = \{2\}$, $K''_+ = \{1,2\}$, on peut distinguer les 4 scénarios :*

- $\Omega_{\{1\}} = [1,5] \times [1,10]$
- $\Omega_{\{2\}} = [1,8] \times [1,8]$
- $\Omega_{\{1,2\}} = [1,5] \times [1,8]$
- $\Omega_{\emptyset} = \Omega = [1,8] \times [1,10]$

*On peut vérifier que $\Omega_{\{1,2\}} \subset \Omega_{\{1\}}$, $\Omega_{\{1,2\}} \subset \Omega_{\{2\}}$ et que $\Omega_{\{1,2\}} = \Omega_{\{1\}} \cap \Omega_{\{2\}}$.*

### 4.2.2 Problème d'ordonnancement : MMRCPSP robuste

Le problème que nous abordons dans ce chapitre concerne comme dans le chapitre 2 l'optimisation de l'utilisation des ressources pour l'ordonnancement des activités sur une ligne d'assemblage aéronautique. Nous rappelons que dans ce problème, chaque activité est soumise à des contraintes de précédence, de décalage temporel et possède plusieurs modes possibles, ce qui peut faire varier son utilisation des différentes ressources. Plusieurs ressources sont en jeu : les ressources des zones qui seront représentées par des contraintes cumulatives limitées, et les ressources humaines, qui correspondent à des opérateurs aux compétences différentes. Comme déjà expliqué, l'objectif est de minimiser la somme de l'utilisation maximale de chaque type d'opérateurs à l'intérieur de chaque intervalle défini par deux points de temps de décision consécutifs. Comme indiqué dans le chapitre 2, le problème industriel étudié peut être modélisé comme un problème d'ordonnancement de projet sous contrainte de ressources multi-modes (MMRCPSP) avec un objectif d'investissement des ressources, comme dans [Gerhards 2020].

Il existe un ensemble d'activités $\mathscr{J}$ et un ensemble de ressources $\mathscr{R}$. Chaque activité $i \in \mathscr{J}$ est associée à un ensemble $\mathscr{M}_i$ de modes tels qu'un mode $m \in \mathscr{M}_i$ pour une activité $i \in \mathscr{J}$ définit la durée $p_{i,m}$ de l'activité et la quantité $b_{i,k,m}$ que l'activité requiert sur la ressource $k \in \mathscr{R}$. Nous avons un horizon temporel discret $H$ avec des points particuliers dits de décision $T \subset H$ avec $H = \{0,\dots,|H|\}$ et $T = \{t_0 = 1,\dots,t_1,t_{|T|} = |H|\}$. L'ensemble des ressources $\mathscr{R}$ comprend deux sous-ensembles distincts : le sous-ensemble des ressources d'investissement $\mathscr{R}^I$, pour lesquelles la consommation de pointe doit être minimisée à l'intérieur de chaque intervalle de décision et le sous-ensemble des ressources renouvelables standard $\mathscr{R}^\rho$, chacune étant associée à une capacité $B_k$ qui ne peut être dépassée. Il existe un ensemble de contraintes de précédence $E$.

En définissant la variable de décision $S_i, \forall i \in \mathscr{J}$, qui donne l'heure de début de chaque activité $i \in \mathscr{J}$ et $m_i \in \mathscr{M}_i$, qui donne le mode choisi pour l'activité $i \in \mathscr{J}$, le MMRCPSP avec objectif d'investissement dans les ressources peut être formellement énoncé comme suit.

$$\min \quad z = \sum_{k \in \mathscr{R}^I} \sum_{q=1}^{|T|} \max_{\tau \in [t_{q-1}, t_q - 1]} R_{k,\tau} \tag{4.2}$$

$$\text{s. t.} \quad S_j - S_i \geq p_{i,m_i} \qquad \forall (i,j) \in E \tag{4.3}$$

$$R_{k,\tau} = \sum_{i \in \mathscr{J}, \tau \geq S_i \wedge \tau \leq S_i + p_{i,m_i} - 1} b_{i,k,m_i} \qquad \forall k \in \mathscr{R}, \forall \tau \in H \tag{4.4}$$

$$R_{k,\tau} \leq B_k \qquad \forall k \in \mathscr{R}^\rho, \forall \tau \in H \tag{4.5}$$

$$S_i \in H \qquad \forall i \in \mathscr{J} \tag{4.6}$$

$$m_i \in \mathscr{M}_i \qquad \forall i \in \mathscr{J} \tag{4.7}$$

L'objectif (4.2), à minimiser est égal à la somme sur chaque ressource des pics maximaux à l'intérieur de chaque intervalle. Les contraintes (4.3) sont des contraintes de précédence standard. Les contraintes (4.4) calculent, pour chaque ressource $k \in \mathscr{R}^I$ l'utilisation totale $R_{k,\tau}$ à chaque temps $\tau \in H$. Pour les ressources renouvelables standard, les contraintes (4.5) empêchent l'utilisation de la ressource de dépasser sa capacité. Les contraintes (4.6,4.7) donnent les domaines des variables.

A propos de la modélisation de l'incertitude et compte tenu du fait que les activités ont plusieurs modes, nous considérons maintenant que $p_{i,m}^{min}$ est la durée minimale de l'activité dans le mode $m$ tandis que $p_{i,m}^{max}$ est sa durée maximale dans le mode $m$. Par conséquent, l'ensemble des scénarios $\Omega$ est l'ensemble des matrices $(p_{i,m})_{j \in \mathscr{J}, m \in \mathscr{M}_i}$ où $p_{i,m} \in [p_{i,m}^{min}, p_{i,m}^{max}]$.

Considérons maintenant une instance de MMRCPSP $I_\Omega$ obtenue en fixant $p_{i,m} = p_{i,m}^{max}$, pour chaque activité $i \in \mathscr{J}$ et chaque mode $m \in \mathscr{M}_i$. Comme indiqué dans la section 4.2.1, la solution optimale du problème MMRCPSP déterministe avec $I_\Omega$ en entrée est la même que pour le problème robuste absolu (4.1) avec $\mathscr{X} = (4.3 - 4.7)$ sur l'instance $I$ soumise aux incertitudes $\Omega$. On désigne par ROBUSTMMRCPSP une procédure qui résout le problème en prenant en entrée l'instance $I_\Omega$ et en sortant la solution $s = \big((S_i)_{i \in \mathscr{J}}, (m_i)_{i \in \mathscr{J}}, z\big)$.

## 4.3 Partitionnement robuste de scénarios basé sur la sélection d'informations

### 4.3.1 Ordonnancement robuste avec sélection à information limitée

Comme expliqué dans la section **??**, le contexte industriel suppose que le décideur a accès à l'information -qui peut avoir un coût- pendant le planning, mais le problème de la sélection de l'information peut ne pas être trivial. Nous proposons une façon de formaliser le problème de sélection de l'information comme une extension du modèle MMRCPSP introduit dans la section précédente. Nous supposons que l'on nous donne une instance du MMRCPSP, un temps de décision $t_q$, et certaines caractéristiques $\bar{s}$ de la solution calculée au point de temps de décision précédent, l'ensemble des activités $K$ desquelles nous pouvons obtenir des informations au temps $t_q$, et un entier $Q$. L'objectif est de calculer une solution robuste avec une meilleure valeur de garantie dans le pire des cas que $\bar{s}$ en utili-

sant au maximum $Q$ informations de $K$, c'est-à-dire en construisant un sous-ensemble de réponses positives $K_+ \subseteq K$ avec $|K_+| \leq Q$. Pour y parvenir, nous introduisons un nouveau problème d'ordonnancement. Celui-ci étend le MMRCPSP comme suit.

Pour chaque activité $i$ dans $K$, nous définissons un ensemble de modes dupliqués $\mathscr{M}_i'$ tels que $m' \in \mathscr{M}_i'$ a la même exigence en ressources que le mode dupliqué $m \in \mathscr{M}_i$ mais une durée égale à $p_{i,m'} = X_i$, qui est la durée réduite du pire cas en cas de réponse positive à la question posée pour $i$. Par conséquent, choisir le mode $m'$ pour l'activité $i \in K$ signifie que $i \in K_+$.

Comme la solution précédente doit être fixée avant $t_q$, nous avons besoin à cet effet des heures de début et du coût d'investissement des ressources avant $t_q$, donnés par $\bar{s}$, où

$$\bar{s} = \left((\bar{S}_i)_{i \in \mathscr{J}}, \bar{R}, \bar{z}\right)$$

où $\bar{S}_i$ représente l'heure de début précédente de l'activité $i$, $\bar{R}$ représente la contribution à la fonction objectif de tous les intervalles dans $[t_0, t_q - 1]$, et $\bar{z}$ représente la valeur totale de la fonction objectif de la solution précédente. Le problème peut maintenant être modélisé comme le MMRCPSP suivant :

$$\begin{aligned}
\min \quad & leximin\,(z, |K_+|) && && (4.8)\\
\text{s. t.} \quad & (4.3 - 4.6) && &&\\
& z \geq \bar{R} + \sum_{k \in \mathscr{R}^I} \sum_{\rho=q}^{|T|} \max_{\tau \in [t_{\rho-1}, t_\rho - 1]} R_{k,\tau} && && (4.9)\\
& z \leq \bar{z} - 1 && && (4.10)\\
& K_+ = \{i \in K | m_i \in \mathscr{M}_i'\} && && (4.11)\\
& |K_+| \leq Q && && (4.12)\\
& S_i = \bar{S}_i && \forall i \in \mathscr{J}, \bar{S}_i \leq t_q - 1 && (4.13)\\
& S_i \geq t_q && \forall i \in \mathscr{J}, \bar{S}_i \geq t_q && (4.14)\\
& m_i \in \mathscr{M}_i \cup \mathscr{M}_i' && \forall i \in K && (4.15)\\
& m_i \in \mathscr{M}_i && \forall i \in \mathscr{J} \setminus K && (4.16)
\end{aligned}$$

Le but de l'objectif (4.8) est de choisir le plus petit sous-ensemble d'informations $|K_+|$ (c'est-à-dire le plus petit nombre de modes de duplication sélectionnés) qui produit la solution avec la meilleure valeur objective $z$. La contrainte (4.9) définit l'objectif principal $z$ avec une partie constante égale à $\bar{R}$ et une partie variable égale à l'investissement en ressources pour tous les intervalles dans $[t_q, H]$. La contrainte (4.10) indique que l'objectif le plus défavorable doit être amélioré en obtenant des informations de $K$. La contrainte (4.11) définit l'ensemble $K_+$ comme l'ensemble des modes de duplication sélectionnés. La contrainte (4.12) limite le nombre sélectionné de modes dupliqués, c'est-à-dire l'information $K_+$ utilisée à partir de $K$ sous la limite $Q$. Les contraintes (4.13) empêchent de modifier les heures de début des activités qui commencent avant le point de décision $t_q$. Les contraintes (4.14) empêchent qu'une activité qui n'a pas été planifiée avant $t_q$ dans $\bar{s}$ soit décalée avant le moment de décision $t_q$. Les contraintes (4.7) du MMRCPSP sont remplacées

par les contraintes (4.15) qui stipulent que les activités dans $K$ ont des modes dupliqués, et les contraintes (4.16) qui conservent l'ensemble des modes originaux des autres activités.

Une autre façon de modéliser la contrainte (4.10) est d'ajouter une ressource non renouvelable qui est utilisée exclusivement par les modes dans $\mathscr{M}'_i$, $i \in K$. Chaque mode $m$ requiert une unité de cette ressource, et seulement $Q$ de celle-ci est disponible. Dans les deux cas, nous pouvons noter que le problème n'est pas fondamentalement différent du MMRSPCP original et le prolonge naturellement.

Nous supposons que nous disposons d'un algorithme, appelé 1SUBSETSOLUTION, pour résoudre ce problème, en prenant comme entrée une instance de MMRCPSP $I_\Omega$ associée à un ensemble de scénarios $\Omega$, l'ensemble des activités $K$ disponibles pour les questions, le nombre maximal de questions $Q$, la solution précédente $\bar{s}$ et le point de temps $q$. L'algorithme produit la nouvelle solution $s$ et les informations utilisées $K_+$.

Un exemple de la manière d'appliquer la duplication de mode à une instance est donné dans l'exemple 8.

**Exemple 8.** *Nous considérons à nouveau l'instance de l'exemple 6. Nous rappelons qu'au temps* $t_1 = 5$ *le décideur a la possibilité de poser deux questions, sur la durée des activités* 1 *et* 2 *(*$K = \{1,2\}$*). Il s'ensuit que les activités* 1 *et* 2 *ont des modes dupliqués :*

$$\mathscr{M}_1 = \{1\}, \mathscr{M}'_1 = \{2\}, \mathscr{M}_2 = \{1\}, \mathscr{M}'_2 = \{2\}$$

*Le problème de la sélection de l'information peut être résolu en résolvant, à l'aide de la méthode décrite précédemment, l'instance suivante :*

| tâches | 1 | | 2 | | 3 | 4 |
|---|---|---|---|---|---|---|
| modes | 1 | 2 | 1 | 2 | 1 | 1 |
| Durée maximale | 6 | 5 | 10 | 8 | 2 | 4 |
| Opérateur 1 | 1 | 1 | 1 | 1 | 3 | 0 |
| Opérateur 2 | 1 | 1 | 2 | 2 | 0 | 4 |

*Si nous exécutons notre modèle avec cette instance, le solveur retournera la solution* $s^1$*, avec un total de* 1 *d'informations utilisées (*$K_+ = \{2\}$*)*

Dans la solution de ce problème, il y a deux choses intéressantes : le sous-ensemble d'informations qui induit un sous-ensemble de scénarios, et la solution qui est robuste pour ce sous-ensemble de scénarios.

### 4.3.2 Partition Robuste

L'algorithme qui vient d'être présenté permet de répondre au problème de sélection de l'information à un instant de décision donné.

En effet, en résolvant ce problème à l'instant $t_q$, on obtient une partition de l'ensemble des scénarios $\Omega$ en deux sous-ensembles $\Omega_{K+}$, sur lequel la solution obtenue $s$ minimise

l'objectif du pire cas étant donné qu'au plus $Q$ informations sont utilisées et $\Omega \setminus \Omega_{K+}$ qui contient encore le scénario global du pire cas sur lequel la solution précédente $\bar{s}$ minimise encore l'objectif du pire cas.

Cependant, on pourrait objecter qu'une réponse de cet algorithme est beaucoup trop optimiste, dans le sens où la solution proposée n'est réalisable que si la réponse à toutes les questions sélectionnées est "oui". Pour pallier ce problème, nous proposons d'utiliser cet algorithme plusieurs fois pour élargir la taille de la partition, en ajoutant à chaque fois de nouvelles contraintes empêchant l'utilisation d'un sous-ensemble de questions déjà sélectionnées. Chaque appel successif à cet algorithme permet de calculer un sous-ensemble différent de questions et de solutions. Supposons maintenant que l'algorithme précédent soit appelé $L-1$ fois, avec $L$ un entier supérieur à 1. Nous désignons par $(K_+^l, s^l)_{i=1}^{L-1}$ l'ensemble des informations et la solution robuste retournée par le $l$-ième appel de l'algorithme précédent. Soit $s^0$ la solution robuste initiale pour $\Omega$. Alors chaque scénario dans $\bigcup_{l=1}^{L-1} \Omega_{K_+^l}$ est couvert par au moins une solution de $\{s^l | l = 1, \ldots, L-1\}$. De plus, afin de s'assurer que le même sous-ensemble d'informations (ou tout ensemble dans lequel il est inclus) n'est pas sélectionné à chaque itération, les ensembles d'informations $K_+^l$ déjà utilisés sont stockés dans une liste $\mathscr{K}_+$.

La contrainte

$$\bigvee_{\kappa_+ \in \mathscr{K}_+} |\{i \in \kappa_+ | m_i \in \mathscr{M}_i'\}| \leq |\kappa_+| \tag{4.17}$$

est ajoutée au modèle (4.8–4.16) pour garantir cette propriété, ce qui signifie que la procédure de solution 1SUBSETSOLUTION prend aussi comme entrée l'ensemble $\mathscr{K}_+$.

Pour construire une partition de $\Omega$, nous devons ajuster la famille $(\Omega_{K_+^l})_l$ pour satisfaire ces conditions. Par construction, si $l' < l$, alors la solution $s_{l'}$ a une meilleure valeur objective dans le pire des cas que $s_l$. Alors la famille ordonnée du sous-ensemble $(\Omega_l)_{l=1}^{L-1}$ défini par

$$\Omega_l = \Omega_{K_+^l} \setminus \bigcup_{l'=1}^{l} \Omega_{K_+^{l'}} \forall lesl = 1, \ldots, L-1$$

garantit que pour tout $l \leq L-1$, chaque scénario dans $\Omega_l$ est couvert par exactement une solution –notamment $s^l$–, et qu'aucune autre solution $s^{l'}$ n'a une meilleure garantie prouvée dans le pire des cas que $s^l$. Enfin, nous pouvons étendre la famille $(\Omega_l)_l$ avec $\Omega_L = \Omega \setminus \bigcup_{i=1}^{L-1} \Omega_l$, et couvrir ses scénarios par la solution $s^0$, qui est la solution qui a été implémentée jusqu'au moment de la décision $t_q$. Or, la famille $(\Omega_m)_{i=1,\ldots,L}$ est clairement une partition de $\Omega$, et chaque scénario est couvert par une et une seule solution de $(s^l)_{l=0,\ldots,L-1}$. On peut noter que le paramètre $L$ limite la taille de la partition. Le pseudo-code de cet algorithme, renvoyant la partition $\mathscr{P}$ et l'ensemble associé de solutions robustes $\mathscr{S}$ calculées pour chaque sous-ensemble de scénarios dans la partition, est présenté dans l'Algorithme 6.

**Exemple 9.** *Considérons à nouveau l'instance de l'exemple courant, dans le même contexte que dans les exemples 6 et 8, mais cette fois, nous voulons calculer une partition robuste. Pour les besoins de l'exemple, nous fixons $L = 2$ et $Q = 1$. Nous commençons par résoudre 1SubsetSolution. Le solveur trouve une solution, et renvoie $(K_+^1 = \{2\}, s^1)$. Par*

*conséquent, le premier sous-ensemble de la partition $\mathscr{P}$ est $\Omega_{K_+^1} = [1,6] \times [1,8] \times [1,2] \times [1,4]$ et la solution qui le couvre est $s^1$. Puisque $L = 2$, nous devons compléter la partition avec le sous-ensemble de scénarios $\Omega \setminus \Omega_{K_1}$, et la couvrir avec la solution $s^0$ (la solution robuste initiale). Finalement, l'algorithme renvoie $\mathscr{P} = (\Omega_{K_1}, \Omega \setminus \Omega_{K_1})$, qui est clairement une partition de $\Omega$, et $\mathscr{S} = (s^1, s^0)$ dont on a prouvé qu'ils ont la meilleure garantie du pire cas sur les sous-ensembles de $\mathscr{P}$ dans le même ordre.*

**Algorithme 6** Partition Robuste

**Entrée:** une instance $I$, $\Omega$ un ensemble de scénarios, $s^0$ une solution robuste $\Omega$, $K$ un ensemble de tâches, un temps $t_q$, $\bar{R}$ le coût de la fonction objectif avant $t_q$ et deux entiers $L$ et $Q$.

$\mathscr{P} \leftarrow \{\}$, $\mathscr{S} \leftarrow \{\}$, $\mathscr{K}_+ \leftarrow \{\}$, $\bar{s} \leftarrow \left((s_i^0)_{i \in \mathscr{J}}, \bar{R}, z^0\right)$

**pour** $1 \leq l < L-1$ **faire**

  $s^l, K_+^l \leftarrow 1\text{SUBSETSOLUTION}(I_\Omega, K, Q, \mathscr{K}_+, \bar{s}, t_q)$

  **si** Pas de solution trouvée **alors**

    **break**

  **fin si**

  $\mathscr{S} \leftarrow \mathscr{S} \cup \{s^l\}$, $\mathscr{K}_+ \leftarrow \mathscr{K}_+ \cup \{K_+^l\}$, $\mathscr{P} \leftarrow \mathscr{P} \cup \{\Omega_{K_+^l} \setminus \bigcup_{\Omega_{K_+} \in \mathscr{P}} \Omega_{K_+}\}$

**fin pour**

$\mathscr{P} \leftarrow \mathscr{P} \cup \{\Omega \setminus \bigcup_{\Omega_{K_+} \in \mathscr{P}} \Omega_{K_+}\}$

$\mathscr{S} \leftarrow \mathscr{S} \cup \{s_0\}$

**retourne** $\mathscr{P}, \mathscr{S}$

## 4.4 Arbre de décision robuste

Dans cette section, nous présentons l'arbre de décision robuste. L'idée d'un tel arbre est qu'à chaque nœud, correspondant à un point de décision $t_q$, nous supposons que certaines informations sont accessibles. Avec ces informations, nous sommes capables de diviser l'ensemble des scénarios et de calculer une nouvelle solution robuste pour chaque sous-ensemble de la partition. La façon dont nous calculons une "bonne" partition est présentée dans la section 4.3.2. La figure 4.5 montre un exemple d'arbre de décision robuste pour l'exemple de la section 2. Nous supposons que les moments de décision, c'est-à-dire les moments où les informations sont disponibles, sont donnés. L'arbre doit être lu de haut en bas, et chaque moment correspond à un niveau dans l'arbre, de sorte que descendre dans l'arbre signifie avancer dans le temps. Cela le rend très facile à utiliser pour un décideur.

La planification commence avec la solution de la racine. A chaque moment de décision, le décideur pose aux opérateurs les questions (sélectionnées dans l'arbre) sur l'état de la planification. Ils sont alors en mesure de choisir la branche correspondant aux réponses aux questions, offrant une continuation robuste à la planification et menant à un autre nœud, et ainsi de suite. Comme on peut l'imaginer, puisque la construction

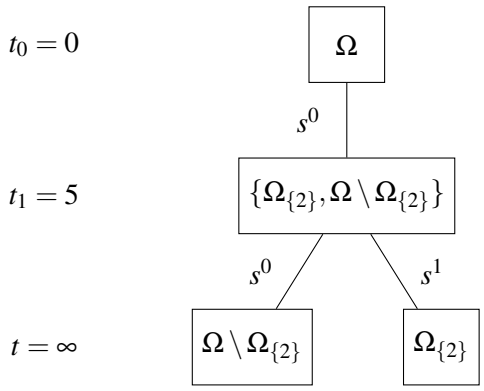



FIGURE 4.5 – Arbre de décision robuste de l'exemple 8

d'un tel arbre nécessite de résoudre plusieurs fois un problème NP-Difficile, cela prend beaucoup de temps. Mais comme les scénarios encapsulent toutes les incertitudes, l'arbre ne doit être calculé qu'une seule fois, dans une phase hors ligne. Une fois fait, l'arbre est réutilisable tant que les incertitudes, et les données du problème d'ordonnancement en jeu, restent inchangées. On peut noter que la solution initiale robuste se trouve toujours dans l'arbre, car le scénario le plus défavorable est toujours "contenu" dans une des branches de l'arbre, et est toujours réalisable, quel que soit le scénario actuel. De la même manière, la solution proposée après le choix d'une branche devient une nouvelle solution de référence, qui reste également faisable dans le sous-arbre correspondant. Ainsi, si plusieurs branches sont disponibles après un moment de décision, la solution locale robuste calculée au nœud de décision précédent sera la solution "par défaut". Ainsi, quel que soit le niveau dans l'arbre, il est nécessaire que les solutions proposées dans les différentes branches soient strictement plus intéressantes que la solution robuste locale. Dans notre cas, "plus intéressante" signifie avec une meilleure valeur de robustesse (i.e. une meilleure valeur minmax) comme expliqué dans la section 4.2. En procédant ainsi, il s'ensuit que, par construction, la valeur la plus défavorable ne peut que s'améliorer au fur et à mesure que l'on descend dans l'arbre. En somme, les arbres de décision robustes brisent le conservatisme inhérent aux approches robustes classiques tout en conservant des limites sur les scénarios les plus défavorables.

De plus, nous pouvons dériver de ces arbres une nouvelle façon de définir le degré de criticité d'une activité. Dans la définition classique de la planification de projet, une activité est critique si une augmentation de sa durée augmente nécessairement la durée totale du projet [Kelley Jr 1959]. Ici, nous pouvons adapter cette définition à notre contexte : une activité est critique si une diminution de sa durée permet le calcul d'un meilleur planning. Comme il est possible de visualiser à l'avance les questions à se poser au prochain moment de décision, il est intéressant pour le décideur de suivre - si possible - précisément les activités concernées. Si ces activités sont réalisées plus rapidement que l'estimation initiale, cela augmente les chances de se trouver dans un sous-ensemble de scénarios couverts par une branche de l'arbre qui améliore significativement le pire cas de la fonction objectif.

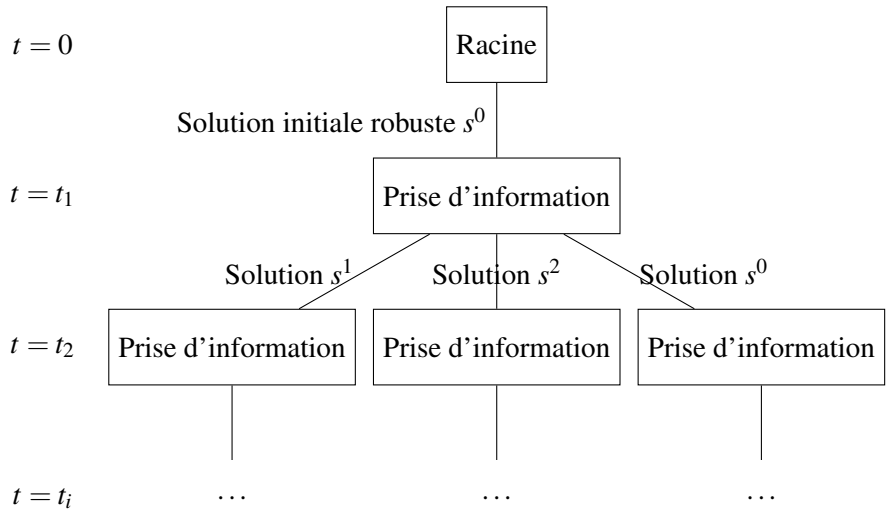



FIGURE 4.6 – Idée générale de l'arbre de décision robuste

Comme expliqué dans l'Algorithme 7, la construction de l'arbre se fait niveau par niveau, chaque niveau correspondant à un instant de décision $t_q$. L'algorithme de l'arbre de décision robuste prend comme paramètres la liste $(t_q) \in T$ des moments de décision, $Q$ le nombre maximal d'informations pouvant être utilisées à chaque nœud et $L$ la taille maximale de la partition calculée à chaque nœud en résolvant le problème de la partition robuste. Ces paramètres limitent la taille de l'arbre. La profondeur de l'arbre est $|T|$, et $L$ est le facteur de branchement maximal de l'arbre qui fait que le nombre total de nœuds dans l'arbre est limité par $\frac{L^{|T|+1}-1}{L-1}$, et le nombre total de feuilles est limité par $L^{|T|}$. De plus, on peut noter que puisque poser une question divise en deux l'ensemble des scénarios (et plus généralement si $m$ questions sont posées, l'ensemble des scénarios est divisé en $2^m$ sous-ensembles), la taille maximale de la partition $L$ ne peut être supérieure à $2^Q$.

**Exemple 10.** *Nous illustrons le processus de génération d'arbre de décision robuste sur un exemple plus large avec* $|\mathscr{J}| = 10$ *activités,* 2 *ressources d'investissement* $\mathscr{R}^I = \{1,2\}$ *et* 2 *ressources renouvelables* $\mathscr{R}^r ho = \{3,4\}$ *avec des capacités* $B = (20,19)$. *Enfin, nous avons les contraintes de précédence :*

$$E = \{(1,7),(1,10),(2,4),(2,5),(2,7),(3,5),(4,6),(4,9),(5,6),(6,8),(7,8),(7,9)\}.$$

*Les autres caractéristiques de l'instance sont affichées dans le tableau 4.2.*

*Au nœud racine, un planning robuste* $s^0$ *est calculé en prenant la durée maximale de chaque mode comme le pire scénario. Un calendrier de coût 74 est obtenu avec des heures de début d'activité* $(2,1,0,3,5,9,8,16,12,9)$ *et des modes* $(1,1,1,1,1,1,1,2,2,1)$. *Le diagramme de Gantt et les pics correspondants sur les deux ressources d'investissement sont présentés dans la figure 4.7. Par souci de concision, l'utilisation des ressources renouvelables n'est pas affichée.*

| Activity | mode | $p_{i,m}^{min}$ | $p_{i,m}^{max}$ | $b_{i,1,m}$ | $b_{i,2,m}$ | $b_{i,3,m}$ | $b_{i,4,m}$ |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 6 | 6 | 6 | 0 |
| | 2 | 2 | 4 | 4 | 3 | 5 | 0 |
| | 3 | 4 | 9 | 4 | 1 | 1 | 0 |
| 2 | 1 | 1 | 1 | 6 | 6 | 0 | 9 |
| | 2 | 3 | 6 | 5 | 6 | 0 | 7 |
| | 3 | 4 | 9 | 4 | 5 | 0 | 2 |
| 3 | 1 | 1 | 1 | 8 | 5 | 7 | 0 |
| | 2 | 3 | 6 | 5 | 4 | 5 | 0 |
| | 3 | 4 | 9 | 4 | 3 | 0 | 6 |
| 4 | 1 | 2 | 5 | 5 | 5 | 8 | 0 |
| | 2 | 4 | 8 | 5 | 4 | 0 | 3 |
| | 3 | 4 | 8 | 3 | 4 | 0 | 6 |
| 5 | 1 | 2 | 4 | 6 | 7 | 0 | 7 |
| | 2 | 2 | 5 | 6 | 5 | 7 | 0 |
| | 3 | 4 | 9 | 5 | 4 | 0 | 5 |
| 6 | 1 | 3 | 7 | 3 | 4 | 7 | 0 |
| | 2 | 4 | 9 | 3 | 3 | 5 | 0 |
| | 3 | 5 | 10 | 2 | 1 | 0 | 6 |
| 7 | 1 | 2 | 4 | 5 | 4 | 7 | 0 |
| | 2 | 3 | 6 | 5 | 4 | 0 | 7 |
| | 3 | 3 | 7 | 4 | 3 | 6 | 0 |
| 8 | 1 | 3 | 7 | 2 | 7 | 0 | 1 |
| | 2 | 3 | 7 | 3 | 6 | 4 | 0 |
| | 3 | 4 | 9 | 2 | 5 | 3 | 0 |
| 9 | 1 | 2 | 5 | 10 | 7 | 2 | 0 |
| | 2 | 2 | 5 | 10 | 7 | 0 | 6 |
| | 3 | 5 | 10 | 10 | 6 | 2 | 0 |
| 10 | 1 | 1 | 2 | 2 | 6 | 0 | 7 |
| | 2 | 4 | 8 | 2 | 5 | 0 | 4 |
| | 3 | 5 | 10 | 1 | 5 | 6 | 0 |

TABLE 4.2 – Une instance de MMRCPSP avec 10 tâches

**Algorithme 7** Arbre de décision robuste

**Entrée:** une instance $I_\Omega$, $\Omega$ un ensemble de scénarios, et deux entiers $L$ et $Q$.

$s \leftarrow$ ROBUSTRCPSP$(I_\Omega)$
Soit un arbre de décision robuste $\mathscr{T}$ avec un noeud racine etiqueté avec$(\Omega, s, 0)$
Initialiser la pile de noeud $\mathscr{Q} \leftarrow (\Omega, s, 0)$
**pour** $1 \leq q < |T|$ **faire**
  $\mathscr{Q}' \leftarrow \{\}$
  **tant que** $\mathscr{Q}$ n'est pas vide **faire**
    On retire le noeud $(\Omega, s, \bar{R})$ de la pile $\mathscr{Q}$
    $K \leftarrow \{j \in \mathscr{J} \,|S_i < t_q < S_i + p_{i,m_i}\}$
    $\bar{R}' \leftarrow \bar{R} + \sum_{j \in R^l} \max_{\tau=t_{q-1}}^{t_q-1} R_{k,\tau}$
    $\mathscr{P}, \mathscr{S} \leftarrow$ ROBUSTPARTITION$(I, \Omega, s, K, t_q, \bar{R}', L, Q)$
    **tant que** $\mathscr{P}$ n'est pas vide **faire**
      retirer l'ensemble de scenarios $\Omega'$ de $\mathscr{P}$ et la solution $s'$ de $\mathscr{S}$
      ajouter le noeud $(\Omega', s', \bar{R}')$ aux noeuds fils de $(\Omega, s, \bar{R})$ in $\mathscr{T}$
      ajouter $(\Omega', s', \bar{R}')$ à la queue $\mathscr{Q}'$
    **fin tant que**
  **fin tant que**
  $\mathscr{Q} \leftarrow \mathscr{Q}'$
**fin pour**
**retourne** $\mathscr{T}$

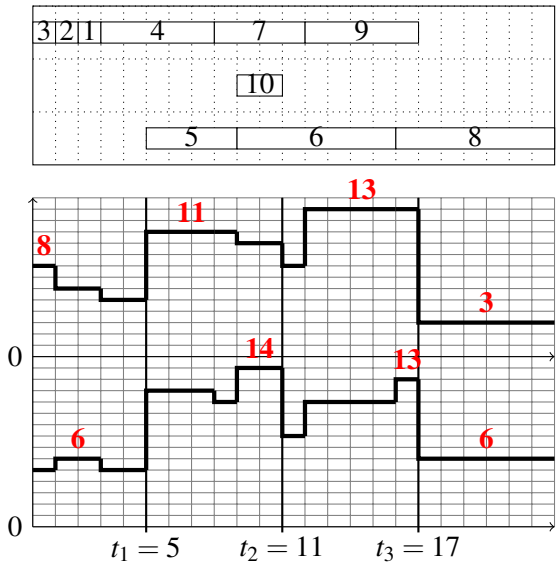



FIGURE 4.7 – Solution robuste $s^0$ avec une valeur objectif de 74 au noeud racine $\Omega$ ($t_0 = 0$)

*En sautant au premier point de décision $t_1 = 5$, l'ensemble des activités sur lesquelles on peut obtenir des informations est $K = 4,5$. En prenant à nouveau $Q = 1$ et $L = 2$, tous les modes de ces activités sont dupliqués et le problème de partition robuste est résolu. La meilleure façon trouvée d'améliorer la solution est d'obtenir des informations de l'en-*

*semble d'activités $K_+^1 = \{5\}$ avec une solution maximale réduite $p_{5,2}^{max} = 2$ dans le mode 2 (ensemble de scénarios $\Omega_5$), ce qui permet de passer à la solution $s^1$ de valeur 70 représentée dans la figure 4.8. L'affectation de mode mise à jour est $(1,1,1,1,2,1,1,1,1,1,1)$. Notez que le mode initial de l'activité 5 peut encore être modifié (du mode 1 au mode 2) puisque l'activité n'est pas encore lancée. Cela donne le nœud enfant gauche identifié par $\Omega_5$, $t_1 = 5$ et $s^1$. Comme $L = 2$, le nœud fils de droite considère tous les scénarios pour lesquels $p_{5,1}^{min} = 3$, $p_{5,2}^{min} = 3$ et $p_{5,3}^{min} = 6$, c'est-à-dire que l'activité 5 ne sera pas terminée à 80% de sa durée initiale prévue dans aucun de ses modes ($\Omega \setminus \Omega_{\{5\}}$), ce qui est couvert par la solution $s^0$.*

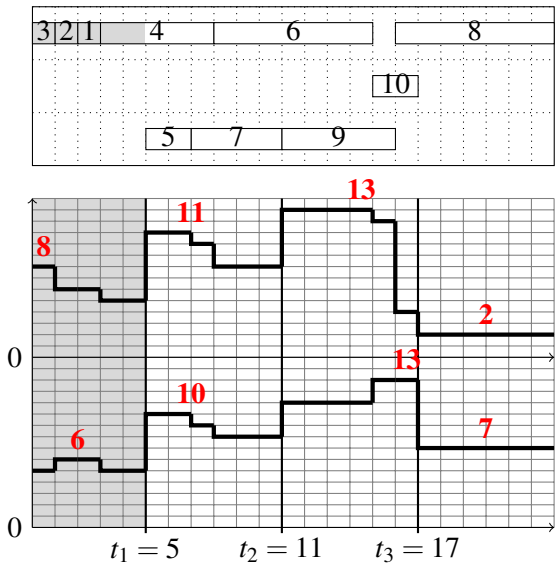



FIGURE 4.8 – Solution robuste $s^1$ avec une valeur objectif de 70 au noeud étiqueté avec $\Omega_5$ ($t_1 = 5$)

*Les nœuds de gauche et de droite sont poussés dans $\mathcal{Q}$ et le nœud ($\Omega_{\{5\}}$, $t_1$, $s^1$) est poussé pour le point de décision $t_2 = 11$. L'ensemble des activités sur lesquelles on peut obtenir des informations est $K = \{6,9\}$ et le problème de partition robuste sélectionne $K_+^1 = \{9\}$ avec une durée réduite de $p_{9,2}^{max}$ en mode 2 (ensemble de scénarios $\Omega_{\{5,9\}}$). Comme précédemment, nous avions $S_9 = t_2$ donc le mode de 9 peut être changé. La solution obtenue $s^2$ est affichée sur la figure 4.9. Les nouveaux modes assignés sont $(1,1,1,1,2,1,1,2,2,1)$. Cela donne le fils gauche $(\Omega_{\{5,9\}}, t_2, s^2)$. Le fils de droite considère les autres scénarios pour lesquels la durée de l'activité 9 est supérieure à 80% de sa durée initiale $(\Omega_{\{5\}} \setminus \Omega_{\{9\}}, t_2, s^1)$. Les deux noeuds sont poussés dans $\mathcal{Q}$.*

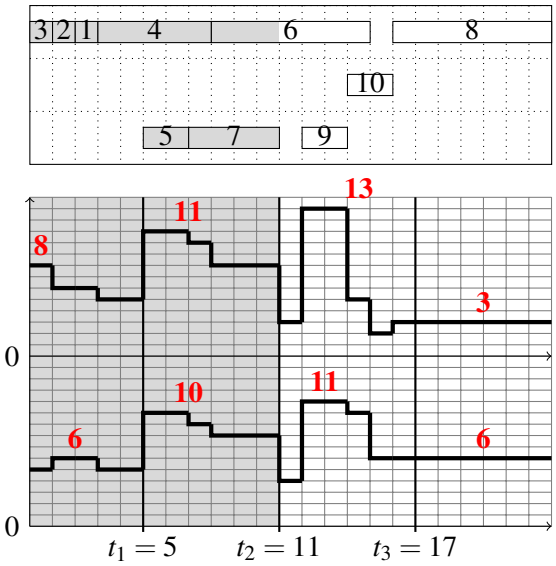



FIGURE 4.9 – Solution robuste $s^2$ avec une valeur objectif de 68 au noeud étiqueté avec $\Omega_{5,9}$ ($t_2 = 11$)

*Le processus est itéré, générant l'arbre de décision robuste affiché sur la figure 4.10. Lorsque le nœud* $(\Omega_{\{5,9\}}, t_2, s^2)$ *est extrait à* $t_3 = 17$*, aucune partition de scénario ne peut améliorer la solution actuelle car seule l'activité 8 commence son traitement à* $t_3$*. Ainsi, un seul enfant est généré. De même, pour tous les nœuds droits supprimés, aucune autre amélioration de la solution ne peut être obtenue. Finalement, l'arbre de décision robuste comporte 9 nœuds, permettant de passer à de meilleures solutions à* $t_1$ *et* $t_2$ *selon la partition du scénario* $\Omega_{\{5,9\}}, \Omega_{\{5\}} \setminus \Omega_{\{9\}}, \Omega \setminus \Omega_{\{5\}}$*. En utilisant uniquement les informations de 2 activités, l'arbre de décision robuste atteint, pour les partitions du scénario considéré, un gain allant jusqu'à 8,10% sur la valeur de la fonction objectif.*

## 4.5 Expérimentations

### 4.5.1 Instances

Afin de tester nos arbres de décision, nous avons construit nos propres instances basées sur les instances MMRCPSP de la PSPLIB [PSPLIB 2020]. Nous nous sommes intéressés en particulier aux instances J10 et J30. Ces deux ensembles diffèrent l'un de l'autre par deux paramètres, indiqués dans le tableau 4.3.

| Set | J10 | J30 |
|---|---|---|
| Nombre de tâches (N) | 10 | 30 |
| Complexité du graphe (C) | 1,8 | 1,5 |

TABLE 4.3 – Les instances J10 et J30

Le nombre de tâches est explicite, et la complexité du graphe est le nombre moyen

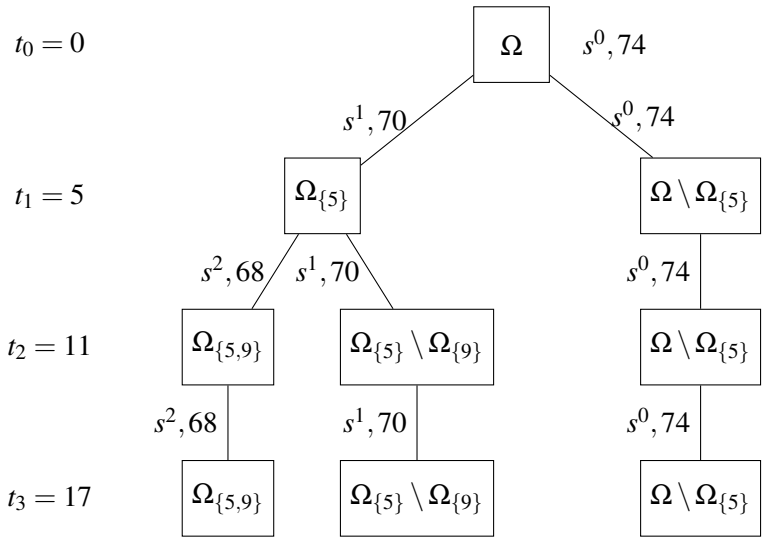



FIGURE 4.10 – Arbre de décision robuste pour l'exemple à 10 tâches

d'arcs non redondants par nœud dans le graphe de précédence. On peut noter que la complexité globale du problème n'est pas monotone par rapport à ce paramètre : lorsque ce paramètre est très élevé (très peu de solutions réalisables) ou très faible (beaucoup de solutions réalisables), les méthodes basées sur les arbres de recherche ont tendance à donner de bons résultats, alors qu'une valeur moyenne de ce paramètre tend à créer des instances difficiles. Tous les détails concernant les paramètres utilisés pour générer ces instances peuvent être trouvés dans [Kolisch 1995] et dans [PSPLIB 2020]. Pour construire nos instances, nous avons dû modifier légèrement celles de la PSPLIB. Ces instances considéraient deux types de ressources, renouvelables et non-renouvelables. Nous avons choisi de considérer les ressources renouvelables comme nos ressources $\mathscr{R}^P$, en gardant les mêmes capacités que dans l'instance originale. Quant aux ressources non renouvelables de l'instance originale, nous changeons leur sémantique et les considérons comme les ressources d'investissement $\mathscr{R}^I$ pour notre problème. L'idée derrière la structure des instances PSPLIB est que les activités ont plusieurs modes, et que plus un mode utilise la ressource non renouvelable, plus la durée de l'activité est courte. Ainsi, pour minimiser le makespan du projet, une des difficultés du problème est de choisir les activités qui seront exécutées avec un mode utilisant beaucoup de ressources non renouvelables. Dans notre cas, le makespan du projet est contraint. Nous utilisons donc les meilleures solutions récupérées depuis PSPLIB pour fixer notre makespan. Enfin, pour générer les intervalles d'incertitude, nous avons considéré que si la durée d'une activité $i$ était initialement $\hat{p}_i$, l'intervalle d'incertitude pour cette durée est $[0.6\hat{p}_i, \hat{p}_i]$. Ainsi, même dans le pire des cas où toutes les activités durent leur durée maximale, nous sommes certains qu'il existe une solution réalisable. Pour chaque instance, nous avons généré 100 scénarios, sous chacune des distributions de probabilité suivantes :

— Distribution uniforme (U) : tous les scénarios sont également possibles..
— Neutral Normal distribution (NN) : distribution normale avec les paramètres $\mu = 0.8\hat{p}_i, \sigma^2 = \frac{0.2\hat{p}_i}{3}$.

— Distribution normale optimiste (ON) : distribution normale avec les paramètres $\mu = 0.7\hat{p}_i, \sigma^2 = \frac{0.1\hat{p}_i}{3}$.
— Distribution normale pessimiste (PN) : distribution normale avec les paramètres $\mu = 0.9\hat{p}_i, \sigma^2 = \frac{0.1\hat{p}_i}{3}$.

L'intérêt de ces distributions est d'observer la qualité des arbres en fonction des tirages des scénarios. Ainsi, si les durées maximales des activités d'une instance sont sous-estimées, le tirage des scénarios en pratique ressemblera à notre distribution normale pessimiste. Inversement, si l'estimation de la durée maximale des activités est surestimée, les scénarios réels ressembleront à ceux générés selon la distribution normale optimiste.

### 4.5.2 Algorithme Réactif

De la même manière que dans le chapitre précédent, afin d'évaluer les performances de l'approche par arbre de décision, il nous a semblé pertinent de comparer les différents critères d'évaluation avec d'autres méthodes. Malheureusement, nous ne connaissons pas d'autres approches qui utilisent l'information de la même manière que nous. Nous avons donc décidé d'implémenter un algorithme réactif relativement simple, qui utiliserait l'information de la même manière. Comme nous voulons évaluer l'impact de la sélection des informations, cet algorithme est paramétré par $r_{info} \in \{0,1\}$ qui définit la proportion d'informations utilisées à chaque instant de décision. On constate qu'en choisissant $r_{info} = 0$, l'algorithme réactif suit simplement l'ordonnancement robuste initial. Le pseudo-code de l'algorithme est présenté dans l'algorithme 8.

**Algorithme 8** Algorithme réactif basé sur l'information

**Entrée:** une instance $I$, $\Omega$ un ensemble de scénarios, $\omega$ un scénario caché de $\Omega$ et $r_{info} \in \{0,1\}$ la proportion d'informations sélectionnées à chaque moment.

```
K ← {}
s_init ← RobustMMRCPSP(I_Ω)
pour 0 ≤ q < |T| faire
    K ← un ensemble d'informations disponibles au moment t_q
    K_+ ← un ensemble d'informations où 100 r_info % des informations de K ont été choisies au hasard
    Ω_q ← le plus grand sous-ensemble Ω_{K_+} tel que ω ∈ Ω_q
    solve : RobustMMRCPSP(I_{Ω_q})
    si Pas de solution trouvée alors
        s_q ← s_{q-1} or s_init if q = 0
    sinon
        s_q ← la solution trouvée par le solveur
    fin si
fin pour
retourne s_{|T|-1}
```

En principe, étant donné une instance et un scénario, l'algorithme simule le processus d'ordonnancement selon le scénario. À chaque moment de décision, $r_{info} * 100\%$ informa-

tions sont sélectionnées aléatoirement, ou en aveugle, parmi les informations disponibles à ce moment-là. Le scénario "répond" aux questions correspondantes, puis calcule une nouvelle solution robuste dans un temps limité, jusqu'au prochain moment de décision ou la fin de l'ordonnancement. Dans cette section, la notation $R_p$ désigne l'algorithme réactif basé sur l'information avec le paramètre $r_{info} = p$.

### 4.5.3 Détails d'implémentations

Dans ce qui précède, nous faisons souvent référence à la procédure de résolution du MMRCPSP avec l'objectif d'investissement en ressources. C'est le cas de la procédure ROBUSTMMRCPSP qui résout le MMRCPSP en fixant la durée de chaque activité à sa valeur maximale et de la procédure 1SUBSETSOLUTION qui résout le MMRCPSP avec des modes supplémentaires pour les activités qui peuvent être incluses dans l'ensemble d'informations $K_+$. Pour ces deux procédures, nous utilisons l'API C++ de IBM ILOG CP Optimizer v12.9, un solveur de programmation par contraintes adapté aux problèmes d'ordonnancement. Nous utilisons la contrainte d'ordonnancement global standard de CP Optimizer. Une description du modèle CP Optimizer pour le MMRCPSP avec l'objectif d'investissement des ressources peut être trouvée dans [Gerhards 2020, Borreguero 2021]. Les autres algorithmes (ROBUSTPARTITION, ROBUSTDECISIONTREE et INFORMATION-BASEDREACTIVEALGORITHM sont codés en C++.

Toutes les expériences ont été exécutées sur un processeur Intel Xeon E5-2695 v4 à 2,10 GHz. exécutant Linux Ubuntu 16.04.4.

### 4.5.4 Impact des paramètres dans le calcul de l'arbre

Comme indiqué dans la section 4.4 le calcul de l'arbre prend en entrée plusieurs paramètres, en plus de l'instance MMRCPSP, et ces paramètres ont un grand impact sur la taille de l'arbre. Compte tenu du nombre d'instances testées, nous avons arbitrairement fixé $L = 3$ et $|T| = 4$ pour l'expérimentation concernant les ensembles $J_{10}$ et $J_{30}$, afin que les arbres puissent être calculés dans un temps raisonnable. La durée maximale totale du projet $C_{\max}$ est divisée en cinq parties égales, ce qui fait la liste des moments de décision $T = [0.2C_{\max}, 0.4C_{\max}, 0.6C_{\max}, 0.8C_{\max}]$. Concernant le paramètre $Q$, le nombre maximum de questions posées à chaque nœud de l'arbre, nous avons remarqué qu'avec $Q > 3$ (resp. $Q > 5$) la plupart des informations ont été réellement utilisées pour les instances et les solutions retournées par les arbres ne s'améliorent plus, pour l'ensemble $J_{10}$ (resp. $J_{30}$). Ces résultats sont présentés dans la figure 4.11. Bien que cela puisse être surprenant, puisque la différence du paramètre C (voir Table 4.3) dans les générations des ensembles induit qu'en moyenne plus d'informations devraient être disponibles à un moment donné pour les instances $J_{10}$, cette quantité d'informations est contrebalancée par la différence du nombre d'activités entre les deux ensembles. Afin de rendre la qualité de la sélection de l'information plus facile à observer, le paramètre $Q$ est fixé à 2 (resp. 4) pour toutes les autres expérimentations impliquant l'ensemble $J_{10}$ (resp. $J_{30}$).

Quant aux algorithmes réactifs robustes, ils ont été exécutés avec $r_{info}$ en $\{0, 0.5, 1\}$,

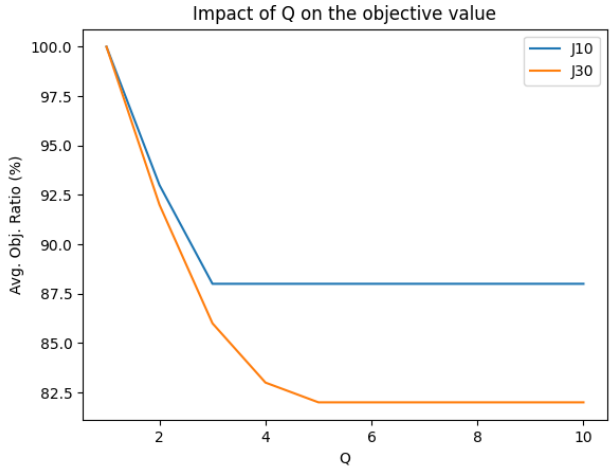



FIGURE 4.11 – Valeur objective moyenne normalisee en fonction de $Q$

sur les mêmes instances que les arbres de décision, avec les mêmes scénarios, les mêmes moments de décision $T$, et avec un temps de calcul à chaque moment d'une minute, et avec le planning courant comme point de départ du solveur. Nous désignerons par $R_0$, $R_{50}$, et $R_{100}$ chacun de ces algorithmes. On peut noter que puisque $R_0$ n'utilise aucune information, la solution retournée par cet algorithme est la solution initiale robuste.

### 4.5.5 Résultats expérimentaux

Nous étudions d'abord l'impact des distributions suivies par les scénarios sur la qualité des solutions proposées par les arbres de décision robustes. La figure 4.12 montre l'écart relatif de la valeur objective moyenne (sur tous les scénarios générés) entre la distribution uniforme et toutes les distributions normales pour les deux ensembles $J_{10}$ et $J_{30}$. Nous pouvons d'abord noter que dans tous les cas, les cases sont entièrement inférieures à 0, ce qui signifie que pour les deux ensembles, les arbres obtiennent de meilleurs résultats sur 75 % des instances lorsque les scénarios suivent une des lois normales que lorsqu'ils suivent la loi uniforme. Nous pouvons également remarquer que pour les deux ensembles, l'écart relatif moyen respecte l'ordre $ON < NN < PN$, ce qui n'est pas surprenant, puisque notre critère de partitionnement favorise l'optimisme dans la sélection de l'information.

Ensuite, nous examinons la quantité d'informations utilisées par chaque algorithme. Les résultats sont présentés dans la figure 4.13. Tout d'abord, on peut observer que les arbres utilisent beaucoup moins d'informations que $R_{100}$ pour les deux instances, et beaucoup plus que $R_0$ (qui par définition n'en utilise aucune). On peut également remarquer que globalement le nombre d'informations utilisées/disponibles est plus élevé pour les instances de l'ensemble $J_{30}$ que pour les instances de l'ensemble $J_{10}$. Ceci confirme l'intuition de l'étude de l'impact du paramètre $Q$ dans la section précédente. Ensuite, si l'on regarde la différence de la moyenne du nombre total d'informations utilisées par les

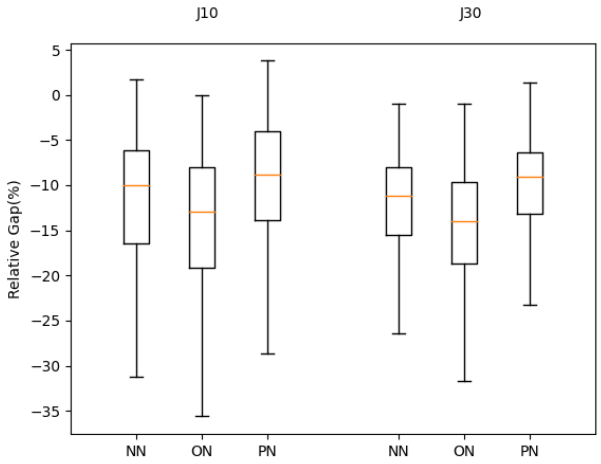



FIGURE 4.12 – Écart relatif normalisé de la valeur objective moyenne entre la distribution uniforme et toutes les autres distributions.

arbres par rapport à $R_{50}$, on constate que pour les instances de l'ensemble $J_{10}$, les arbres utilisent moins d'informations sur plus de 75% des instances. Par contre, pour les instances de l'ensemble $J_{30}$, ce nombre est assez similaire. Au vu de l'analyse de la qualité des solutions, on peut conclure que les arbres utilisent mieux l'information que les algorithmes réactifs, parvenant à faire mieux (en moyenne) même avec moins d'information.

Maintenant que nous savons que les arbres ont tendance à utiliser moins d'informations que l'algorithme $R_{50}$, nous pouvons évaluer la qualité de la solution offerte par les arbres de décision robustes par rapport à celles produites par les algorithmes réactifs. La figure 4.14 montre l'écart relatif de la valeur objective moyenne (sur tous les scénarios pour une instance donnée) de la solution renvoyée par les arbres et chaque algorithme réactif entre chaque instance pour les deux ensembles. Dans ce cas, une valeur positive signifie que l'arbre bat l'algorithme réactif correspondant, tandis qu'une valeur négative signifie le contraire. Tout d'abord, on peut noter que dans tous les cas, l'algorithme réactif utilisant 0 d'information ne bat jamais l'arbre de décision robuste sur la même instance. Le contraire peut être remarqué concernant les algorithmes réactifs utilisant toute l'information disponible : seulement dans les meilleurs cas, les arbres de décision robustes correspondent à l'algorithme réactif. On peut toutefois observer que l'algorithme $R_{100}$ ne bat la décision robuste que de quelques pourcents pour plus de 75% des instances, et que les arbres battent $R_{50}$ pour plus de 75% des instances.

Enfin, nous étudions le nombre de solutions retournées par les différents algorithmes. D'un point de vue pratique, il est important pour les opérateurs que les horaires varient le moins possible entre deux répétitions du problème. Deux solutions sont considérées comme différentes si l'ordre induit par les dates de début de chaque activité dans chaque solution est différent. La figure 4.15 montre le nombre moyen de solutions différentes trouvées par

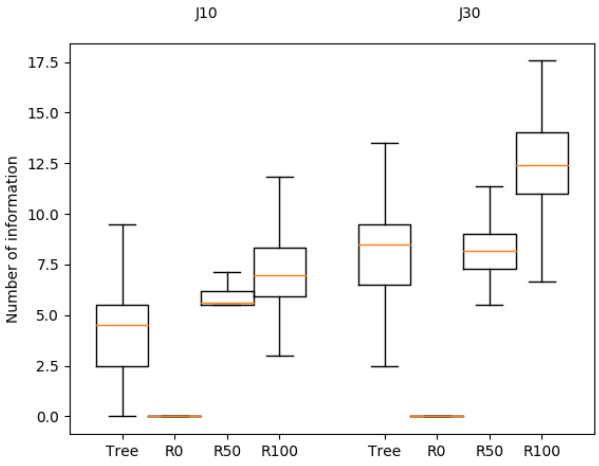



FIGURE 4.13 – Nombre total moyen d'informations utilisées par les arbres et les algorithmes réactifs.

chaque algorithme pour chaque instance sur tous les scénarios. Évidemment, puisque l'algorithme $R_0$ ne s'adapte pas au scénario, il renvoie la solution initiale robuste. Cependant, pour les deux ensembles, les arbres de décision robustes renvoient environ 10 solutions différentes par instance, alors que $R_{50}$ et $R_{100}$ renvoient environ 30 solutions différentes pour les instances de $J_{10}$, et environ 60 solutions pour les instances de $J_{30}$. Ainsi, du point de vue du nombre de solutions, les arbres sont préférables aux algorithmes réactifs.

### 4.5.6 Résultats sur les instances industrielles

Dans cette section, nous présentons des résultats expérimentaux sur deux instances industrielles fournies par Airbus [Borreguero 2015]. Ces instances sont désignées par instance *A* et instance *B*, avec respectivement 721 et 486 activités. Ces instances n'ont pas été données avec des incertitudes d'intervalle ou des scénarios, nous les avons donc générées de la même manière que celle expliquée dans la section 4.5.1. Pour ces deux instances, nous avons généré des arbres de décision robustes avec les paramètres suivants :

— Nombre de moments de décision $T = 10$.
— Nombre maximal d'informations à chaque instant $Q \in [1,..,10]$.
— Taille maximale des partitions à chaque instant $L \in [2,..,4]$.

En ce qui concerne les algorithmes réactifs, nous avons fait varier $r_{info}$ dans $[0, 0.1, ..., 1]$. Les résultats sont présentés dans les figures 4.16 et 4.17. Pour être plus précis, chaque point de ces figures est la valeur (valeur objective moyenne, nombre moyen d'informations par décision) pour une distribution de scénarios donnée et une paramétrisation pour un arbre (avec un couple $Q,L$) ou un algorithme réactif robuste (avec $r_{info}$). Parmi tous les points générés, nous avons conservé celui qui forme le front de Pareto. Dans ce cas, puisque nous voulons minimiser à la fois la valeur objective moyenne et le nombre moyen d'informations, les "meilleurs" points sont ceux qui se trouvent en bas à gauche de chaque figure. Il y a plusieurs choses à noter à propos de ces figures. Tout d'abord, l'impact des distributions

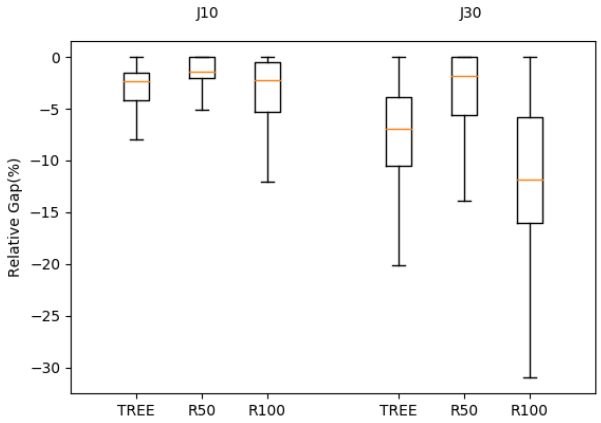



FIGURE 4.14 – Écart relatif (en pourcentage) de la valeur objective moyenne entre les solutions renvoyées par les arbres, $R_{50}$ $R_{100}$ et la solution robuste initiale renvoyée par $R_0$.

de scénarios semble être le même que celui noté dans la section 4.5.5. Deuxièmement, nous pouvons observer qu'il y a peu de points formant des fronts de Pareto, en particulier pour l'instance *A*. Une première raison est qu'il y a beaucoup de points qui se chevauchent : comme nous l'avons vu dans la section 4.5.4, augmenter le paramètre $Q$ n'améliore pas la qualité des solutions à partir d'un certain point. De plus, étant donné la taille des instances et le fait que les algorithmes ont été exécutés en un temps limité (60 sec par moment de décision), il arrive parfois qu'aucune solution ne soit calculée dans le temps imparti, laissant le nœud " vide " dans le sens où aucune question n'est posée, et la solution robuste précédente est automatiquement sélectionnée pour la suite de la construction de l'arbre. Néanmoins, on peut voir sur les figures que les points des arbres de décision sont généralement un peu plus à gauche que ceux des algorithmes réactifs sur la même distribution, à l'exception de l'instance *B*, où l'arbre utilise peu d'information.

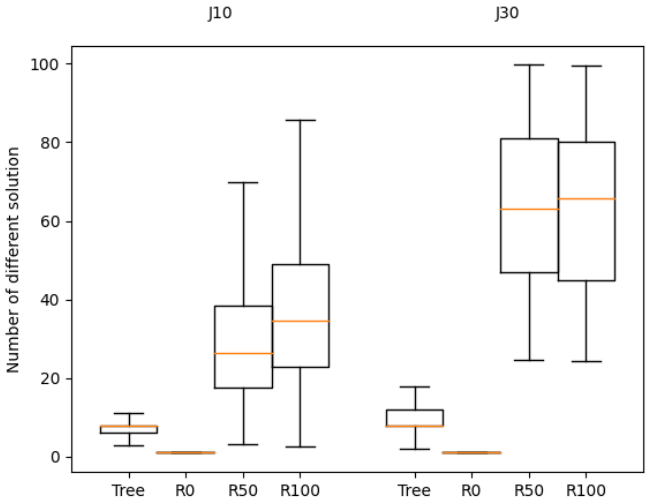



FIGURE 4.15 – Nombre de solutions différentes sur 100 scénarios pour chaque instance pour les deux ensembles.

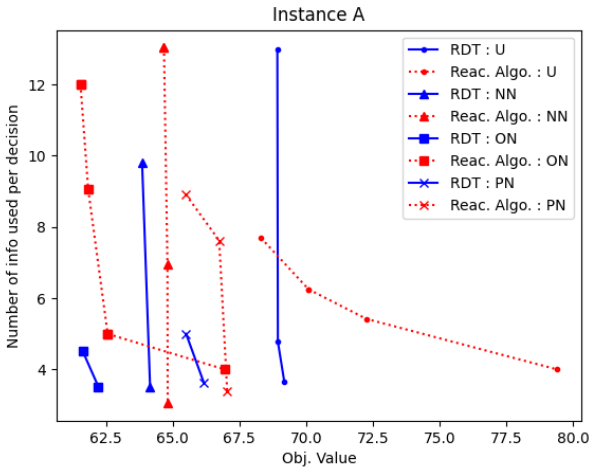



FIGURE 4.16 – Résultats expérimentaux sur les instances *A*

## 4.6 Conclusion

Dans ce travail, nous avons présenté un cadre basé sur des arbres de décision qui, en pratique, rompt le conservatisme inhérent aux approches robustes habituelles.
Nous montrons comment calculer de tels arbres dans le cas particulier du problème d'ordonnancement de lignes d'assemblage aéronautiques, modélisé comme un MMRSPSP. Nous avons ensuite évalué les arbres de décision robustes d'abord sur des instances de référence, puis sur des instances industrielles réelles fournies par Airbus Madrid en faisant

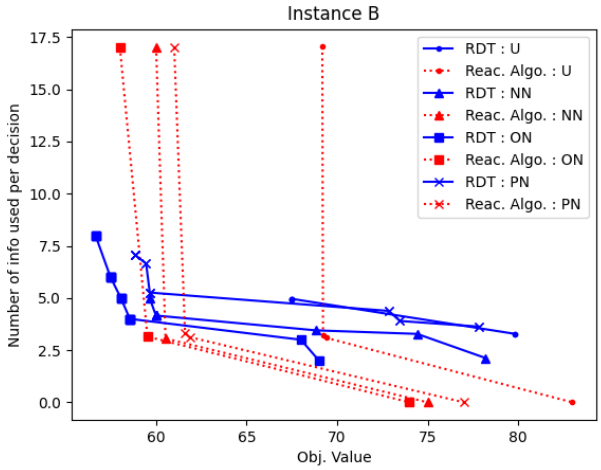



FIGURE 4.17 – Résultats expérimentaux sur les instances *B*

varier les paramètres de construction des arbres et les distributions selon lesquelles les scénarios sont tirés. Les résultats de ces expériences ont montré des résultats satisfaisants sur les instances de référence, et des résultats prometteurs sur les instances industrielles.
Un défaut majeur de notre approche est que de tels arbres prennent un temps de calcul très long (même s'il peut être contrôlé par des paramètres). Néanmoins, nous pensons que le temps de calcul peut être amélioré de manière significative, par exemple en fusionnant les nœuds équivalents de l'arbre au fur et à mesure qu'ils sont calculés. De plus, lorsque l'arbre est calculé, on peut imaginer utiliser des techniques de compilation de connaissances pour réduire sa taille.
On peut noter que le cadre proposé dans ce chapitre peut être étendu à n'importe quel problème dès lors qu'il comporte une composante temporelle. Cependant, toutes les fonctions objectives ne sont pas forcément adaptées. Les objectifs dont la valeur peut être fixée "tôt" dans la solution (par exemple un objectif $L_{max}$ pour les problèmes d'ordonnancement) seront plus difficiles à améliorer au fil du temps. Au contraire, pour les fonctions objectives basées sur des sommes et dont la valeur varie avec le temps (la plupart des fonctions objectives classiques pour les problèmes de dimensionnement de lots par exemple), l'utilisation d'arbres de décision robustes est tout à fait appropriée. De plus, on peut imaginer plusieurs façons d'adapter ce cadre au problème que l'on souhaite. Par exemple, nous avons choisi de commencer par une solution très -et trop- conservatrice et de l'améliorer au fur et à mesure. Cependant, on peut imaginer de construire l'arbre dans l'autre sens : partir d'une solution super-optimiste, et essayer de la dégrader le moins possible en choisissant les meilleures informations au fur et à mesure du déroulement du scénario. Intuitivement, on pourrait penser que cette approche serait plus appropriée dans les cas où les scénarios optimistes sont très probables.
En outre, les discussions avec Airbus nous ont contraints en matière d'accès à l'information. On pourrait étendre l'approche en supposant qu'il est possible d'avoir des informations sur le futur, et pas seulement sur la partie du scénario qui s'est déjà produite.

Cela aurait un sens pour les problèmes d'ordonnancement avec une incertitude sur la date d'échéance des activités, par exemple.

CHAPITRE 5

# Methode de Benders adverse étendue pour un problème de planification robuste

**Sommaire**



Les travaux présentés dans ce chapitre ont été présentés à la conférence ROADEF 2022 et ont été finalistes du prix du meilleur article étudiant [Portoleau 2022b].

## 5.1 Introduction et description du problème

Dans les deux chapitres précédents, nous avons proposé un modèle permettant d'approcher de manière robuste et optimiste des problèmes d'ordonnancement incertains, dans la mesure où un décideur a accès à des informations à propos du scénario alors que celui-ci est en train de se réaliser.

Dans ce chapitre, nous proposons une variante d'une méthode d'optimisation robuste appelée méthode de Benders adverse, et l'appliquons non pas à un problème d'ordonnancement mais à un problème de gestion de production. La méthode de Benders adverse (ou parfois méthode des plans coupants) a été introduite dans [Mutapcic 2009]. Cette approche se base notamment sur la résolution itérée d'un problème dit adverse, c'est-à-dire problème où l'on cherche à calculer un pire scénario pour une solution réalisable donnée. Une approche similaire appelée "algorithme de relaxation des scénarios" avait été proposée pour des problèmes d'optimisation robuste en version min-max regret dans [Assavapokee 2008]. La méthode de Benders adverse est indiquée lorsque ce problème adverse est facile à résoudre et dépend entre autres de la nature de l'ensemble des scénarios. Dans [Pätzold 2020], les auteurs présentent une variante de cette méthode où des pires scénarios sont calculés de manière approximative, et prouvent la convergence de leur méthode. L'intuition de la méthode que nous présentons dans ce chapitre, est que l'on cherche à calculer non pas un seul scénario pire cas mais un ensemble de pires scénarios, avec l'idée que l'on peut gagner du temps de calcul en réduisant le nombre d'itérations de l'algorithme. Nous présentons cette méthode pour un problème de dimensionnement de lots (*lot-sizing*) avec contraintes de capacité (*Capacited Lot Sizing Problem* – CLSP), sous budget d'incertitude portant sur la demande cumulée. Typiquement, pour modéliser ce problème, on se donne un horizon découpé en $T$ périodes de temps, et on associe à chaque période $t \leq T$ une demande $d_t$, et une production $x_t$. L'ensemble de ces variables de production constitue un plan de production. Le coût de production d'une période $t$ est un coût de lancement $c^P$. A ce coût, des coûts linéaires liés à chaque pas de temps $t$ et aux trois quantités suivantes sont ajoutées. Tout d'abord, si l'on produit plus que la demande, le surplus stocké sera noté $I_t$. Ensuite, la quantité effectivement vendue, qui satisfait au moins une partie de la demande est notée $s_t$. Et finalement, la quantité $B_t$ représente la part de la demande non satisfaite. Ces trois quantités sont pondérées, respectivement par le coût de stockage $c^I$, le prix de vente $b^P$ et le coût de réapprovisionnement $c^B$. On supposera que la quantité produite à chaque pas de temps est bornée, et donc que $X = (x_t)_{t \leq T}$ prend ses valeurs dans un polytope $\mathbb{X}$. On peut noter que ces différents coûts sont indépendants de la période de temps et que ce problème de lot sizing déterministe est NP-Difficile [Bitran 1982]. L'incertitude du problème porte sur la demande cumulée. Pour modéliser cela, on pose $X_t = \sum_{i=1}^{t} x_t$ et $D_t = \sum_{i=1}^{t} d_t$, qui représentent respectivement, la production cumulée jusqu'à la période $t$ et la demande cumulée jusqu'à $t$. Concernant les incertitudes des demandes cumulées, on suppose que chaque demande $D_t$ prend sa valeur dans un intervalle symétrique de la forme $[\hat{D}_t - \Delta_t, \hat{D}_t + \Delta_t]$, avec $\hat{D}_t \geq \Delta_t \geq 0$ où $\hat{D}_t$ est sa valeur nominale et $\Delta_t$ sa déviation maximale. On appelle scénario une réalisation de ces incertitudes et on note $\mathscr{U}$ un ensemble de scénarios. Dans la suite, on s'intéressera plus particulièrement à l'ensemble de scénarios continus suivant :

$\mathscr{U}$ est l'ensemble des vecteurs de demandes cumulées $\mathscr{D} = (D_t)_{t \leq T}$ vérifiant les

contraintes suivantes :

$$
\begin{aligned}
& D_t \leq D_{t+1} && t \leq T-1 \\
& D_t \geq \hat{D}_t - \Delta_t && t \leq T \\
& D_t \leq \hat{D}_t + \Delta_t && t \leq T \\
& \|\mathscr{D} - \hat{\mathscr{D}}\|_1 \leq \Gamma &&
\end{aligned}
$$

où $\Gamma$ est le budget d'incertitude. Pour résoudre ce problème de lot-sizing nous avons choisi d'utiliser le critère le robustesse absolue, ou *minmax*. Naturellement puisque le problème déterministe est déjà difficile, il le reste dans sa version incertaine.

## 5.2 Revue de littérature

Le problème de *lot-sizing* sous incertitude a vastement été étudié au cours des dernières décennies, et l'on se limitera dans cette section à des références où l'incertitude porte sur la demande. Tout d'abord, on trouve dans la littérature un certain nombre d'articles où l'incertitude du problème est décrite par des arbres de scénarios. C'est le cas de [Brandimarte 2006], dans lequel les auteurs considèrent une version du problème avec capacité multi-éléments. L'incertitude de la demande est explicitement modélisée par un arbre de scénarios, ce qui donne lieu à un modèle de programmation stochastique en nombres entiers mixtes à plusieurs étapes avec recours. Ils proposent une formulation du modèle basée sur la localisation des usines et une approche heuristique basée sur une recherche locale. Ils présentent des expérimentations numériques pour évaluer non seulement la viabilité de l'heuristique, mais aussi l'avantage (le cas échéant) du modèle de programmation stochastique par rapport au modèle déterministe considérablement plus simple basé sur l'espérance de la demande. Ils essayent différentes approches pour générer l'arbre de scénarios. Leurs résultats suggèrent qu'il existe une interaction entre différents leviers de gestion pour couvrir l'incertitude de la demande, c'est-à-dire les tampons de capacité réactive et les stocks de sécurité. Lorsque la capacité réactive est suffisante, la capacité du modèle stochastique à constituer des stocks de sécurité est de peu d'utilité. Lorsque la capacité est fortement limitée et que l'impact des temps de préparation est important, des avantages remarquables sont obtenus en modélisant explicitement l'incertitude. Avec des objectifs très similaires, dans [Hu 2016] les auteurs traitent d'un problème de *lot-sizing* multi-produits avec des configurations dépendantes de la séquence. Ils proposent un modèle de programmation stochastique en deux étapes pour minimiser les coûts totaux de production, de stock et de rupture de stock. La première étape décide de la production de base, y compris la quantité de production de chaque produit et la séquence de production, tandis que la deuxième étape se concentre sur les mises à jour possibles de la production de base, telles que la production d'heures supplémentaires. L'objectif est de trouver la meilleure séquence de quantités de production sous une demande aléatoire avec des commandes en attente autorisées. Là aussi, l'incertitude est explicitement représentée par un arbre de scénarios puis par la sélection des scénarios les plus représentatifs afin d'obtenir un sous-ensemble plus petit tout en préservant les propriétés essentielles. Le temps de préparation et le coût de prépara-

tion dépendent tous deux du produit. Une étude de cas pour une entreprise de fabrication d'équipements de freinage a été menée pour illustrer et valider le modèle. Leurs résultats montrent que le modèle stochastique est plus performant que le modèle déterministe, surtout lorsque les ressources de production sont suffisantes. Dans [Quezada 2022], un état de l'art des approches de *lot-sizing* stochastique sans contraintes de capacité avec arbres de scenarios et une amélioration de la méthode *stochastic dual dynamic integer programming algorithm* de [Zou 2019] est proposée pour résoudre ce problème.

Dans [Ben-Tal 2005], les auteurs proposent de résoudre le problème d'optimisation robuste pour un problème de *lot-sizing* avec un seul produit et des incertitudes de forme convexe en se limitant à des recours affines, comme détaillé au Chapitre 1. Dans [Santos 2018], une borne inférieure pour un problème de *lot-sizing* robuste sans contraintes de capacité avec budget d'incertitude sur la demande est obtenue de manière efficace par relaxation des contraintes de non anticipation. Finalement dans [Alem 2018] les auteurs présentent une étude du *General Lot-Sizing and Scheduling Problem* (GLSP) –qui combine un problème de *lot-sizing* et un problème d'ordonnancement– sous demande incertaine à l'aide d'une optimisation robuste à base de budget d'incertitude et d'un modèle de programmation stochastique à deux étapes avec recours. Ils développent une procédure systématique basée sur la simulation de Monte Carlo pour comparer les deux modèles en termes de robustesse et de temps de calcul. Les expérimentations numériques couvrent différentes caractéristiques d'instances, en tenant en compte des budgets d'incertitude et des niveaux de variabilité pour le modèle d'optimisation robuste, ainsi que d'un nombre croissant de scénarios et de fonctions de distribution de probabilité pour le modèle de programmation stochastique. Une autre approche robuste est proposée dans [Roels 2006] où les auteurs étudient une version min-max regret du problème. En particulier, ils s'intéressent au problème avec des informations limitées sur la distribution de la demande. En ce sens, ils quantifient la valeur de l'information supplémentaires et mettent en avant que le fait de savoir que la distribution de la demande a une forme régulière est généralement plus utile que de connaître sa moyenne et sa variance. Cependant, il n'existe, à notre connaissance, pas de travaux -hormi [Guillaume 2020] sur lequel nous nous appuyons dans ce chapitre- traitant du cas particulier où l'incertitude porte sur la demande cumulée.

## 5.3 Méthode de Benders adverse (BA)

Dans cette section, nous décrivons précisément la méthode de Benders adverse qui permet de résoudre des problèmes d'optimisation robuste avec le critère de robustesse absolue, et l'appliquons au problème de *lot-sizing* avec incertitude sur la demande. Le principe de la méthode de Benders adverse est le suivant. L'idée est de résoudre le problème **minmax** (ou problème maître), d'abord sur un seul scénario (la demande nominale $\hat{\mathscr{D}}$), puis en augmentant le nombre de scénarios considérés. Pour calculer le prochain scénario $\mathscr{D}'$ à ajouter au problème maître, on effectue une phase adverse, c'est-à-dire que l'on cherche, étant donnée la solution particulière $S$ du problème calculée à l'itération précédente, à calculer le pire scénario pour cette solution. Ce problème est appelé problème adverse (que l'on notera **Adv**). Une fois ce scénario calculé, il est ajouté à la liste des scénarios du problème

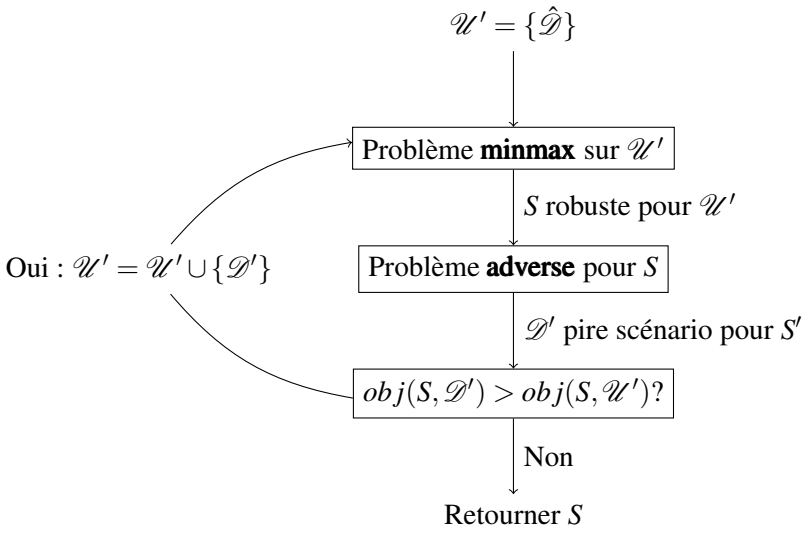



FIGURE 5.1 – Schéma de la méthode de Benders adverse

maître. On réitère alors jusqu'à ce que le critère d'arrêt soit atteint, à savoir quand le scénario obtenu à l'issue de la résolution du problème adverse ne permet pas de dégrader la valeur objectif du problème maître de l'itération en cours. Quand cela se produit c'est que la dernière solution calculée en résolvant le problème **minmax** est la solution robuste recherchée. On peut s'en convaincre en remarquant que par définition, la valeur objectif d'une solution robuste d'un problème **minmax**, ne peut être dégradée par aucun scénario. Le schéma montré sur la figure 5.1 donne une représentation visuelle de cet algorithme.

L'utilisation de cette méthode est recommandée lorsque le problème d'optimisation robuste sur l'ensemble des scénarios est trop difficile pour s'y attaquer frontalement, ce qui peut être dû à la forme de l'ensemble des scénarios considérés, ou à leur nombre en cas d'ensemble de scénarios discrets, et lorsque le problème adverse est relativement facile, ce qui, comme on le verra, est le cas pour notre problème. Pour appliquer cette méthode à notre problème, il "suffit" donc d'être capable de résoudre le problème **minmax** pour une liste de scénarios donnée, et de résoudre le problème **Adv** pour une solution du problème donnée.

### 5.3.1 Le problème minmax

En ce qui concerne le problème **minmax**, on peut le modéliser par le programme linéaire en variables mixtes avec les variables suivantes. Pour une liste de scénario $U \subseteq \mathscr{U}$ et un horizon de temps $T$ on a, pour $1 \leq t \leq T$ et $1 \leq l \leq |U|$, $X_t$ la production cumulée jusqu'au temps $t$, $y_t$ la variable booléenne qui vaut 1 si une production et effectuée au temps $t$ et 0 sinon, $I_{t,l}$ la quantité de produit stockée au temps $t$ dans le scénario $l$, $B_{t,l}$ la quantité de produit non vendue au temps $t$ dans le scénario $l$ et $s_{t,l}$ la quantité de produit vendue au temps $t$ dans le scénario $l$. On peut remarquer que seules les deux premières familles

de variables sont indépendantes de la liste de scénarios. On peut alors écrire le programme linéaire en variables mixtes suivant :

$$
\begin{aligned}
&\min z \\
&\text{s.t.} \sum_{t=1}^{T} (c^I I_{t,l} + c^B B_{t,l} + c^P y_t - b^P s_{t,l}) \leq z && l \leq |U| && (5.1)\\
&B_{t,l} + X_t = D_t^l + I_{t,l} && 1 \leq t \leq T, l \leq |U| && (5.2)\\
&\sum_{i=1}^{t} s_{i,l} = D_t^l - B_{t,l} && 1 \leq t \leq T, l \leq |U| && (5.3)\\
&X_t - X_{t-1} \leq M y_t && 2 \leq t \leq T && (5.4)\\
&X_1 \leq M y_1 && && (5.5)\\
&B_{t,l}, I_{t,l}, s_{t,l} \geq 0 && 1 \leq t \leq T, l \leq |U| && \\
&X \in \mathbb{X} && && \\
&y_t \in \{0,1\} && 1 \leq t \leq T &&
\end{aligned}
$$

avec $M = D_T + \Gamma$ et $z$ une variable auxiliaire et $D_t^l$ la demande cumulée au temps $t$ dans le $l$-ième scénario de la liste $U$. On notera ce programme linéaire $\mathscr{P}_U$. Les contraintes 5.2 et 5.3 garantissent la cohérence des variables $X$, $B$, $I$ et $s$ à chaque pas de temps et pour chaque scénario. On s'assure également de la correction des valeurs des variables $y$ à l'aide des contraintes 5.4 et 5.5. Le "big $M$" choisi, à savoir $D_T + \Gamma$ est bien une borne supérieure pour les variables $X$ puisqu'il s'agit de la demande maximale possible. Notons que la contrainte $X \in \mathbb{X}$, où $\mathbb{X}$ est un polytope de dimension $T$ permet de représenter des contraintes de capacité ou bien d'autre relations linéaires entre les variables de production de différentes périodes. Finalement, pour un $l$ donné, la quantité à gauche dans l'inégalité 5.1 vaut la valeur de la fonction objectif de la solution $X$ dans le $l$-ième scénario de la liste $U$. De cette manière, puisque l'on cherche à minimiser $z$ qui est un majorant de chacune de ces quantités, on résout bien le problème de la forme $\min_{X \in \mathbb{X}} \max_{u \in U}$ souhaitée.

Cette formulation est compacte pour un nombre de scénarios fixé et possède $O(T|U|)$ contraintes, $O(T|U|)$ variables continues et $O(T)$ variables entières. De plus, puisqu'elle s'appuie sur une liste de scénarios, elle a l'avantage de ne pas dépendre de la forme de l'ensemble de scénarios. On peut cependant remarquer qu'elle fait intervenir des contraintes de type "big $M$" qui ont tendance à poser des difficultés aux solveurs, en fonction du choix de $M$. Dans notre cas, notre choix dépend de $\Gamma$ ce qui implique que l'on peut s'attendre à ce que, pour une même instance et une même liste de scénario, une augmentation du budget d'incertitude ralentissent la résolution du problème.

### 5.3.2 Le problème **Adv**

Quant au problème **Adv**, dans [Guillaume 2020], les auteurs ont montré que ce problème était NP-Difficile au sens faible en le réduisant à une variante du problème sac-à-dos (*knapsack*), et ont proposé un algorithme de programmation dynamique de complexité pseudo-polynomiale pour le résoudre. L'idée de cet algorithme est de calculer un plus long

chemin dans un graphe partitionné construit à partir d'un plan de production $X$ de la manière suivante.

**Définition 5** (Graphe de budget)**.** *On note ce graphe $\mathscr{G} = (V,A)$. L'ensemble des noeuds $V$ partitionné en $T+2$ niveaux et, mis à part le premier et le dernier niveau qui contiennent un unique noeud chacun, chaque niveau contient $\Gamma+1$ noeuds. On note respectivement $v_0$ et $v_{T+1}$ les noeuds du premier et du dernier niveau, et $v_{t,i}$ pour $1 \leq t \leq T$ et $0 \leq i \leq \Gamma$ tous les autres noeuds du graphe. L'ensemble des arcs $A$ est construit de sorte que l'arc de $v_{t-1,i}$ vers $v_{t,j}$ existe si et seulement si $i \leq j$ et $j-i \leq \Delta_t$ pour chaque $t$ (ou niveau) compris entre $1$ et $T$, et un arc de $v_{t,i}$ vers $v_{t+1,i}$ est défini pour $i$ compris entre $0$ et $\Gamma$. Ce graphe est tel que chaque chemin de $v_0$ vers $v_{T+1}$ représente un scénario de $\mathscr{U}$, et que si un noeud $v_{t,i}$ est sur ce chemin, alors $i$ unités de budget ont été consommées entre la première période et la période $t$. Ensuite, on définit les coûts des arcs de sorte qu'ils représentent chacun la contribution à la fonction objectif de la décision de faire varier la demande à une période donnée pour une solution $X$ donnée. Plus formellement, on définit les coûts des arcs par :*

$$c_{t-1,i,t,j} = \begin{cases} \max\{c^I(X_t - (\hat{D}_t - (j-i))), c^B(\hat{D}_t + (j-i) - X_t)\} + c^P y_t & \text{si } 1 \leq t \leq T-1, \\ \max\{c^I(X_t - (\hat{D}_t - (j-i))) - b^P(\hat{D}_t - (j-i)), & \\ c^B(\hat{D}_t + (j-i) - X_t) - b^P X_t\} + c^P y_t & \text{si } t = T, \\ 0 & \text{si } t = T+1 \end{cases}$$

*où $c_{t-1,i,t,j}$ est le coût de l'arc entre les noeuds $v_{t-1,i}$ et $v_{t,j}$ lorsque cet arc existe, et $y_t = 0$ si $X_{t-1} = X_t$ et $1$ sinon.*

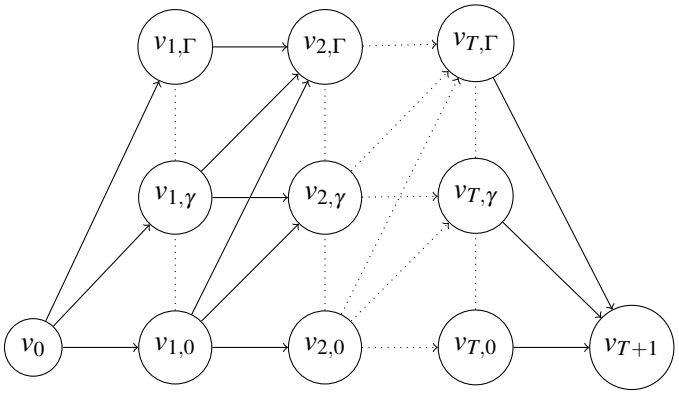



FIGURE 5.2 – Schéma générique du graphe de budget

Sauf précision contraire, dans la suite de ce chapitre, lorsqu'on parlera de "chemin" dans un graphe de budget, on parlera par abus de langage uniquement des chemins entre les noeuds $v_0$ et $v_{T+1}$.

On peut montrer que sous l'hypothèse que toutes les données du problème sont entières, l'ensemble des chemins dans ce graphe est en bijection avec l'ensemble des scénarios restreint aux valeurs entières $\mathscr{U} \cap \mathbb{Z}^T$. Pour reconstruire un scénario $D = (D_t)_{t \leq T}$ à partir d'un chemin du graphe de budget, on pose, pour chaque arc $(v_{t-1,i}, v_{t,j})$ sur le chemin

$D_t = \hat{D}_t + s(t)(j-i)$ avec, pour $1 \leq t \leq T-1$, $s(t) = 1$ si $c^B(\hat{D}_t + (j-i) - X_t) \geq c^I(X_t - (\hat{D}_t - (j-i)))$ et $s(t) = -1$ sinon, et, pour $t = T$ $s(t) = 1$, si $c^B(\hat{D}_t + (j-i) - X_t) - b^P X_t \geq c^I(X_t - (\hat{D}_t - (j-i))) - b^P(\hat{D}_t - (j-i))$ et $s(t) = -1$ sinon. Réciproquement pour un scénario entier $D = (D_t)_{t \leq T}$ donné, on note $\gamma_t$ la quantité de budget d'incertitude utilisée à la période $t$ dans $D$, à savoir $|\hat{D}_t - D_t|$. Alors par construction le chemin passant par les arcs $(v_{t-1,i}, v_{t,j})$ pour $1 \leq t \leq T+1$ et $0 \leq i, j \leq \Gamma$ tels que $|j-i| = \gamma(t)$ existe dans $\mathscr{G}$.

Puisqu'un chemin dans le graphe représente un scénario unique, le graphe $\mathscr{G}$ représente un ensemble de scénarios que l'on note $\mathscr{U}_{\mathscr{G}}$. On a alors $\mathscr{U} \cap \mathbb{Z}^T = \mathscr{U}_{\mathscr{G}}$. Pour une solution $X$ donnée, et donc un coût des arcs de $\mathscr{G}$ donné, on peut montrer (voir [Guillaume 2020]) que par définition du coût des arcs, le coût d'un chemin est égal à la valeur objectif de $X$ dans le scénario représenté par ce chemin. Ainsi, toujours sous l'hypothèse de données entières, il est montré dans [Guillaume 2020]) que le plus long chemin dans le graphe de budget et le coût de ce chemin représentent respectivement le pire scénario dans l'ensemble continu $\mathbb{U}$ pour une solution $X$ et la valeur objectif de cette solution sur ce pire scénario. Ainsi, la résolution du problème **Adv** revient à résoudre un problème de calcul de plus long chemin dans un graphe orienté sans cycle, ce qui peut être résolu dans notre cas en $O(T\Gamma^2)$ par un algorithme de programmation dynamique. On peut ensuite retrouver précisément le scénario par *backtracking*.

### 5.3.3 Algorithme de Benders adverse pour le lot-sizing incertain

Maintenant que nous avons présenté comment résoudre les problèmes **minmax** et **Adv**, on peut résoudre notre problème en utilisant la méthode de Benders adverse en suivant l'algorithme 9 qui précise le schéma de la figure 5.1 présenté au début de cette section.

**Algorithme 9** Benders adverse

$U \leftarrow \{\hat{\mathscr{D}}\}$
$(X, v) \leftarrow$ solution optimale et valeur objectif optimale de $\mathscr{P}_U$
$(\mathscr{D}, v') \leftarrow$ pire scénario et sa valeur objectif du problème **Adv** pour la solution $X$
**tant que** $v \neq v'$ **faire**
  $v \leftarrow v'$
  $U \leftarrow U \cup \{\mathscr{D}\}$
  $(X, v) \leftarrow$ solution optimale et valeur objectif optimale de $\mathscr{P}_U$
  $(\mathscr{D}, v') \leftarrow$ pire scénario et sa valeur objectif du problème **Adv** pour la solution $X$
**fin tant que**
**retourne** $X$

## 5.4 Méthode de Benders adverse étendue (BAE)

Dans cette section, nous proposons maintenant une variante de la méthode de Benders adverse présentée dans la section précédente. L'idée de cette variante est qu'à chaque phase adverse, on ne renvoie pas seulement le pire scénario pour une solution donnée, mais un ensemble de mauvais scénarios. Cependant, le problème maître $\mathscr{P}_U$, défini pour une liste

de scénarios $U$, passe difficilement à l'échelle lorsque le nombre de scénarios explose. Pour pallier cela, on propose de réécrire le problème maître en se basant cette fois non pas sur une liste de scénarios, mais sur le graphe de budget introduit dans la section précédente.

### 5.4.1 Graphe partiel de budget

Une manière de voir le graphe de budget est de le considérer comme une représentation relativement compacte de l'ensemble des scénarios. En se basant sur la définition du graphe de budget du chapitre précédent, on définit un graphe partiel de budget, qui représente un sous-ensemble de scénarios, de la manière suivante.

**Définition 6** (Graphe partiel de budget)**.** *Soit $\mathscr{G} = (V,A)$ un graphe de budget défini suivant la définition 5. On dit que $\mathscr{G}' = (V,A')$ est un graphe partiel de budget de $\mathscr{G}$ si $A' \subset A$ et pour tout arc $a \in A'$, il existe un chemin de $v_0$ vers $v_{T+1}$ passant par l'arc $a$. Les poids des arcs de ce graphe sont calculés de la même manière que pour le graphe de budget défini précédemment. On a alors $\mathscr{U}_{\mathscr{G}'} \subset \mathscr{U}_{\mathscr{G}}$.*

Nous allons utiliser ce type de graphe pour redéfinir le problème maître.

### 5.4.2 Problème maître

Tout comme le problème maître de la méthode de Benders adverse classique se basait sur l'approximation de l'ensemble de scénarios $\mathscr{U}$ par une liste de scénarios, nous proposons cette fois un problème maître où l'on approxime $\mathscr{U}$ par un sous-ensemble de scénarios représenté par un graphe partiel de budget. Pour résoudre ce nouveau problème maître, on utilise une formulation en programme linéaire en variables mixtes qui étant donné un sous-graphe de budget $\mathscr{G}'$, calcule le planning robuste $X$, par rapport à l'ensemble des scénarios représentés par $\mathscr{G}'$. L'idée de cette formulation est de considérer notre problème comme un problème de plus long chemin, en termes de potentiels, dans le graphe $\mathscr{G}' = (V,A')$ avec des arcs dont le poids dépend des variables de décisions, à savoir $X_t$ et $y_t$ qui représentent encore une fois respectivement la production cumulée jusqu'à un instant $t$ et la décision binaire de produire à cet instant ou non. Les variables de potentiels associées à chaque noeud du graphe, notées $\pi_{t,i}$ pour $v_{t,i} \in V$ sont égales au coût du plus long chemin du noeud $v_0$ jusqu'au noeud $v_{t,i}$. Pour des questions de lisibilité, un arc $(v_{t-1,i}, v_{t,j})$ sera noté $a_{t,i,j}$. On

notera cette formulation $\mathscr{P}'_{\mathscr{G}'}$ et elle s'écrit de la manière suivante :

$$
\begin{aligned}
&\min \pi_{T+1} \\
&\text{s.t. } \pi_{t_j} - \pi_{(t-1)_i} \geq c^I(X_t - (\hat{D}_t - (j-i))) + c^P y_t && t \in [1, T-1], a_{t,i,j} \in A' && (5.6)\\
&\quad \pi_{t_j} - \pi_{(t-1)_i} \geq c^B(\hat{D}_t + (j-i) - X_t) + c^P y_t && t \in [1, T-1], a_{t,i,j} \in A' && (5.7)\\
&\quad \pi_{t_j} - \pi_{(t-1)_i} \geq c^I(X_t - (\hat{D}_t - (j-i))) + c^P y_t \\
&\quad - b^P(\hat{D}_t - (j-i)) && t = T, a_{t,i,j} \in A' && (5.8)\\
&\quad \pi_{t_j} - \pi_{(t-1)_i} \geq c^B(\hat{D}_t + (j-i) - X_t) + c^P y_t - b^P X_t && t = T, a_{t,i,j} \in A' && (5.9)\\
&\quad \pi_{T+1} - \pi_{T,i} \geq 0 && 0 \leq i \leq \Gamma && (5.10)\\
&\quad \pi_0 = 0, \pi_v \in \mathbb{R} && v \in V && (5.11)\\
&\quad X_t - X_{t-1} \leq M y_t && 2 \leq t \leq T && (5.12)\\
&\quad X_1 \leq M y_1 && && (5.13)\\
&\quad y_t \in \{0,1\} && 1 \leq t \leq T \\
&\quad X \in \mathbb{X}
\end{aligned}
$$

avec $M = D_T + \Gamma$. Les termes à droite des inégalités des contraintes de 5.6 à 5.11 représentent les coûts des arcs $a_{t,i,j}$ lorsqu'ils existent et bornent les valeurs des variables $\pi$. Comme précédemment, les contraintes 5.12 et 5.13 assurent la cohérence entre les variables $X$ et $y$. Cette formulation comporte $O(|A'|+\Gamma)$ contraintes, $O(|A'|+\Gamma)$ variables continues et $O(T)$ variables entières. Elle a donc l'avantage d'être compacte alors qu'elle permet de résoudre le problème **minmax** pour un nombre possiblement exponentiel de scénarios, ce que la formulation précédente du problème ne permettait pas de faire. On peut remarquer que la taille de l'ensemble des arcs $A'$ est bornée par un $O(T\Gamma^2)$. Cependant, si l'on a choisi d'exprimer le nombre de contraintes et de variables en fonction de $|A'|$ c'est parce que l'on essayera en pratique de jouer sur cette valeur pour utiliser au mieux notre méthode.

### 5.4.3 Sous-problème (**Adv**)

On peut maintenant considérer le problème de la phase adverse, qui cette fois consiste, étant donné un plan de production $X$, à calculer un graphe partiel de budget $\mathscr{G}'$ qui représente un sous-ensemble de mauvais scénarios pour $X$. Pour faire cela, on calcule dans un premier temps par programmation dynamique -de la même manière que pour la méthode de Benders adverse classique présentée dans la section précédente- le plus long chemin entre chaque noeud et le noeud $v_0$ du graphe de budget $\mathscr{G} = (V,A)$. Comme dans la formulation PLNE de la sous-section précédente, on appelle $\pi_{t,i}$ ce coût pour le noeud $v_{t,i} \in V$. On sélectionne ensuite les arcs du graphe partiel de budget par backtracking. On part du noeud $v_{T+1}$ et on considère l'ensemble des noeuds de la $T$-ieme couche qui participent activement au plus long chemin, c'est à dire tels que $\pi_{T+1} = \pi_{T,i} + c_{T,i,T+1}$. On en sélectionne un sous-ensemble, et pour chacun des noeuds sélectionnés on ajoute l'arc dont il est l'origine au graphe partiel. On réitère en remontant le graphe couche par couche jusqu'au noeud $v_0$. Ainsi, le nombre d'arcs du graphe partiel dépend de la manière dont on a sélectionné les noeuds à chaque étape du *backtraking*. Les méthodes que nous avons implémentées pour

sélectionner les noeuds déterminant les arcs du graphe partiel de budget sont détaillées dans la section 5.5 de ce chapitre. L'algorithme 10 donne le pseudo code qui permet de résoudre le problème **Adv** à partir d'un graphe de budget dont le coûts des arcs est obtenu à partir d'une solution $X$ donnée.

**Algorithme 10 Adv** (BAE)

**Entrée:** Un graphe de budget $\mathscr{G} = (V, A)$
  $\pi_0 \leftarrow 0$
  **pour** $t = 1 \rightarrow T+1$ **faire**
    **pour** $j = 0 \rightarrow \Gamma+1$ **faire**
      $\pi_{t,j} \leftarrow \max\{\pi_{t-1,i} + c_{t-1,i,t,j} | (v_{t-1,i}, v_{t,j}) \in A\}$
    **fin pour**
  **fin pour**
  $A' = \{\}$
  $v_{T+1} \leftarrow$ marqué
  **pour** $t = T \rightarrow 1$ **faire**
    $A'_t = \{\}$
    **pour** $j = 0 \rightarrow \Gamma+1$ **faire**
      **si** $t = T$ **et** $\pi_{T+1} = \pi_{T,j} + c_{T,j,T+1}$ **alors**
        $A'_t \leftarrow A'_t \cup \{(v_{T,j}, v_{T+1})\}$
      **fin si**
      $A'_t \leftarrow$ Selection($A'_t$)
      **pour** $(v, v') \in A'_t$ **faire**
        $v \leftarrow$ marqué
      **fin pour**
      **si** $v_{t,j}$ est marqué **alors**
        **pour** $i = 0 \rightarrow \Gamma+1$ **faire**
          **si** $\pi_{t-1,i} + c_{t-1,i,t,j} = \pi_{t,j}$ **alors**
            $A'_t \leftarrow A'_t \cup \{(v_{t-1,i}, v_{t,j})\}$
          **fin si**
        **fin pour**
        $A'_t \leftarrow$ Selection($A'_t$)
        **pour** $(v, v') \in A'_t$ **faire**
          $v \leftarrow$ marqué
        **fin pour**
      **fin si**
    **fin pour**
  **fin pour**
  **retourne** $\mathscr{G}' = (V, A'), \pi_{T+1}$

### 5.4.4 Fusion de graphes partiels de budget

Avant de pouvoir adapter cette étape de phase adverse pour une approche de Benders adverse, il reste une dernière étape. En effet dans l'approche classique, on ajoute les scénarios un à un dans une liste. Dans notre cas, cela signifie que l'on a besoin d'une

opération d'union ou de fusion de graphes. On propose donc la fonction de fusion de graphes suivante : $Merge((V,A'),(V,A'')) = (V,A' \cup A'')$. On peut cependant remarquer que l'ensemble des scénarios représentés par la fusion de deux graphes partiels de budget peut-être plus grand que la somme des tailles des sous-ensembles de scénarios de chacun des deux graphes partiels. Plus formellement, $|\mathscr{U}_{Merge(\mathscr{G}',\mathscr{G}'')}| \geq |\mathscr{U}_{\mathscr{G}'}| + |\mathscr{U}_{\mathscr{G}''}|$. On peut s'en convaincre simplement en regardant la figure 5.3. Sur cette figure est représenté un graphe partiel de budget avec deux chemins. Le premier, $c^1 = v_0 \to v_{1,1} \to v_{2,1} \to v_{3,1} \to v_4$ représenté avec des tirés, et le second $c^2 = v_0 \to v_{1,0} \to v_{2,1} \to v_{3,2} \to v_4$ représenté avec des pointillés. Supposons que ce graphe soit la fusion de deux graphes partiels de budget qui contenaient tous les deux un unique chemin, respectivement $c^1$ et $c^2$. On peut voir que le résultat de cette fusion contient deux chemins supplémentaires, à savoir $v_0 \to v_{1,1} \to v_{2,1} \to v_{3,2} \to v_4$ et $v_0 \to v_{1,0} \to v_{2,1} \to v_{3,1} \to v_4$. Malgré cela, en utilisant cette opération, on ne change ni la correction ni la terminaison de la méthode de Benders adverse, l'important étant que le graphe contienne au moins le pire scénario pour la solution de l'itération en cours. Cependant, l'existence de ce phénomène pousse à être prudent sur la taille des sous-graphes de budget issus de la phase adverse : si l'on prend des sous-graphes trop grands on risque de se retrouver au bout de quelques itérations à avoir un graphe partiel de budget qui est en réalité le graphe de budget en entier.

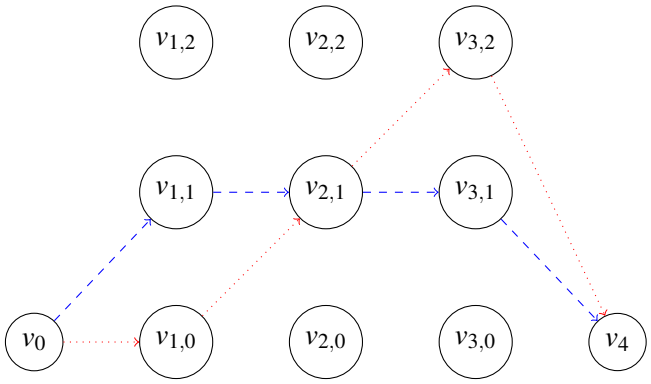



FIGURE 5.3 – Exemple de fusion de graphes partiels de budget

### 5.4.5 Algorithme de Benders adverse étendue pour le lot-sizing incertain

On peut maintenant présenter l'algorithme de Benders adverse étendu, qui est décrit dans l'algorithme 11. De la même manière que l'on initialisait la liste de scénarios du Benders adverse classique avec le scénario nominal $\hat{D}$, on initialise cet algorithme avec le graphe partiel de budget qui ne représente que le scénario nominal, c'est-à-dire un unique chemin $v_0 \to v_{1,0} \to v_{2,0} \to \cdots \to v_{T,0} \to v_{T+1}$, que l'on note $\hat{\mathscr{G}}$.

**Algorithme 11** Benders adverse étendu

$\mathscr{G}' \leftarrow \hat{\mathscr{G}}$
$(X, v) \leftarrow$ solution optimale et valeur objectif optimale de $\mathscr{P}'_{\mathscr{G}'}$
$(\mathscr{G}'', v') \leftarrow$ graphe partiel de budget et son plus long chemin, solution de **Adv** pour $X$
**tant que** $v \neq v'$ **faire**
  $v \leftarrow v'$
  $\mathscr{G}' \leftarrow Merge(\mathscr{G}', \mathscr{G}'')$
  $(X, v) \leftarrow$ solution optimale et valeur objectif optimale de $\mathscr{P}'_{\mathscr{G}'}$
  $(\mathscr{G}'', v') \leftarrow$ graphe partiel de budget et son plus long chemin, solution de **Adv** pour $X$
**fin tant que**
**retourne** $X$

## 5.5 Expérimentations numériques

### 5.5.1 Description des instances

Afin de comparer les différentes méthodes, nous avons sélectionné des instances de **csplib.org**. Ces instances comportent entre 200 et 500 périodes de temps et décrivent un problème de lot-sizing multi-produit avec un coût de stockage variant avec le temps. Pour adapter ces instances à notre problème, nous avons tout d'abord agrégé tous les produits et leurs demandes associées en un seul produit. Pour les différents coûts de la fonction objectif, le coût de stockage $c^I$ de la première période de l'instance initiale a été choisi puis étendu à toutes les périodes. Tous les autres coûts ($c^B$, $c^P$, et $b^P$) ont été tirés aléatoirement en suivant une loi uniforme dans l'intervalle $[10, 20]$. En ce qui concerne l'ensemble des scénarios $\mathscr{U}$, celui-ci dépend des valeurs $\delta_t$ qui sont, pour chaque pas de temps $t$ l'écart maximal possible à la valeur nominale $\hat{D}_t$. Ils ont chacun été tirés uniformément dans l'intervalle $[0, \hat{D}_t]$. Finalement, On définit le polytope $\mathbb{X}$ avec des contraintes de capacité $X_t \leq \mathbb{X}_t, \forall t \leq T$ en tirant uniformément les valeurs $\mathbb{X}_t$ dans l'intervalle $[\frac{\hat{D}_{t+1}-\hat{D}_t}{2}, \frac{3(\hat{D}_{t+1}-\hat{D}_t)}{2}]$.

### 5.5.2 Sélection des noeuds

Comme expliqué précédemment la méthode de Benders adverse étendue nécessite, lors de la résolution du problème **Adv** d'une méthode de sélection des arcs qui constitueront le graphe partiel de budget renvoyé par l'algorithme 10. Lors de nos expérimentations, nous avons utilisé deux méthodes de sélection différentes :

- La sélection *SelectAll* consiste à choisir tous les arcs entrants du nœud examiné dont les noeuds d'origine appartiennent à un des plus longs chemins dans le graphe partiel de budget. Autrement dit, en utilisant cette méthode dans l'algorithme 10, les chemins contenus dans le graphe partiel obtenu sont tous les plus longs chemins de ce graphe. Ils constituent l'ensemble de tous les pires scénarios entiers pour la solution courante.
- La sélection *SelectFew* consiste à choisir systématiquement un premier arc et éventuellement un autre avec une probabilité égale à $\frac{1}{T}$ parmi tous les arcs entrant dont les noeuds d'origines appartiennent à un des plus longs chemins dans le graphe

partiel de budget.

Comparativement à la méthode *SelectAll*, le graphe partiel de budget obtenu en utilisant la méthode *SelectFew* représente aussi un sous-ensemble de scénarios pire cas pour la solution courante, mais de taille beaucoup plus réduite. La raison à cela est qu'il existe pour nos instances un grand nombre de pire scénarios équivalents. Expérimentalement, nous avons pu remarquer qu'après plusieurs itérations -et opérations de fusions- le nombre d'arcs du graphe partiel de budget obtenu avec la méthode *SelectAll* avait tendance à exploser rapidement, rendant très vite ce graphe partiel très dense et difficile à exploiter. Ce phénomène explique en partie les résultats expérimentaux de la section qui suit.

### 5.5.3 Résultats expérimentaux

Les différents programmes linéaires ont été résolus avec CPLEX 12.9 et tous les calculs ont été effectués avec un processeur Intel Xeon CPU E5-2695 v4 2.10GHz sous Linux Ubuntu 16.04.4. Pour ces expérimentations, nous avons considéré deux paramètres. Tout d'abord le budget d'incertitude $\Gamma$ qui prend ses valeurs dans l'ensemble $\{1, 20, 40, 60, 80, 100\}$, ainsi que notre méthode de sélection. Nous avons au total trois méthodes :

*BA* : la méthode de Benders adverse standard.

$BAE_{all}$ : la méthode de Benders adverse étendue, en utilisant la méthode de sélection *SelectAll*.

$BAE_{few}$ : la méthode de Benders adverse étendue, en utilisant la méthode de sélection *SelectFew*.

Les deux critères qui sont observés sont le nombre d'itérations de l'algorithme et le temps de calcul pour chacune des méthodes et chacun des $\Gamma$. Les résultats sont affichés dans la table 5.1 avec à gauche dans chaque cellule le nombre d'itérations moyen, et à droite le temps de calcul moyen de ces méthodes sur l'ensemble des instances. Pour chaque $\Gamma$, les valeurs des cellules avec le plus petit nombre d'itération moyen ou le plus petit temps de calcul moyen ont été mises en gras.

| $\Gamma$ \ Méthode | *BA* | | $BAE_{all}$ | | $BAE_{few}$ | |
|---|---|---|---|---|---|---|
| | iter. | temps | iter. | temps | iter. | temps |
| 1 | 3.27 | 1.63 | **1.82** | **0.19** | **1.82** | **0.19** |
| 20 | 6.18 | 10.34 | **1.82** | 7.82 | 3.82 | **1.79** |
| 40 | 4.81 | 12.62 | **1.81** | 29.36 | 3.90 | **6.29** |
| 60 | 6.81 | 21.18 | **1.90** | 62.2 | 5.72 | **18.30** |
| 80 | 8.09 | 42.89 | **1.90** | 90.4 | 4.09 | **21.14** |
| 100 | 5.81 | **28.45** | **2.0** | 134 | 5.18 | 44.0 |

TABLE 5.1 – Nombre d'itérations et temps d'exécution (en secondes) moyen pour chacune des trois méthodes.

Tout d'abord, on peut observer que quand le budget d'incertitude $\Gamma$ vaut 1 les résultats de $BAE_{all}$ et $BAE_{few}$ sont similaires. On peut expliquer cela en remarquant qu'avec un budget d'incertitude aussi faible, le nombre de scénarios est également petit –en $O(T)$– et que,

dans ce cas, les graphes partiels de budget obtenus pendant les phases adverses ne doivent pas beaucoup varier en fonction de la méthode de sélection utilisée. Sans surprise, la méthode $BAE_{all}$ est meilleure que les deux autres quand on regarde le nombre d'itérations en moyenne, et ce pour toutes les valeurs de $\Gamma$. Comme expliqué précédemment, avec la méthode de sélection $SelectAll$, on se retrouve en pratique avec un graphe partiel de budget qui recouvre quasiment le graphe de budget total du problème après seulement une ou deux itérations, c'est-à-dire que quasiment l'intégralité des scénarios de $\mathscr{U}_{\mathscr{G}}$ est contenue dans le graphe partiel de budget. C'est pour cela que cette méthode converge en aussi peu d'itérations. En revanche, cela se répercute également sur le temps de calcul. La taille de la formulation du problème adverse de la méthode BAE dépend fortement de la taille du graphe partiel de budget qui, lorsque presque tous les arcs sont présents, contient $O(T\Gamma^2)$ arcs. Il était donc prévisible que le temps de calcul augmente très fortement lorsque $\Gamma$ augmente également. En ce qui concerne la méthode $BAE_{few}$ on peut observer que son nombre d'itérations est toujours supérieur à celui de $BAE_{all}$ et inférieur à celui de *BA* et ce quelle que soit la valeur de $\Gamma$. Encore une fois cela n'est pas surprenant, car c'est précisément l'idée de *BAE* que d'ajouter au problème maître un ensemble de scénarios plutôt qu'un seul à chaque itération de l'algorithme. On peut également observer que lorsque $\Gamma$ vaut 1, 20, 40, 60 et 80 les temps de calcul de $BAE_{few}$ sont significativement plus bas que ceux de $BAE_{all}$, et toujours en dessous de ceux de *BA*. C'est n'est en revanche pas le cas quand $\Gamma = 100$, où *BA* est plus rapide que *BAE*. Cela peut s'expliquer par le fait qu'à partir d'une certaine valeur de $\Gamma$, le nombre de scénarios représenté par le graphe partiel de budget - même ceux produits en utilisant $SelectFew$- reste trop important et ralentisse le problème maître. Cependant les résultats suggèrent qu'en contrôlant correctement la quantité de scénarios ajoutée à chaque itération de *BAE*, il est possible d'améliorer *BA*, et parfois de manière importante.

## 5.6 Conclusion et perspectives

Pour conclure, dans ce chapitre nous avons présenté une variante de la méthode de Benders adverse pour résoudre un problème d'optimisation robuste pour le lot-sizing avec budget d'incertitudes sur la demande cumulée. Les résultats expérimentaux sont encourageants et indiquent que notre variante peut être plus intéressante que l'approche classique si le budget d'incertitude n'est pas trop élevé. Nous pensons qu'il est possible d'améliorer les résultats en contrôlant encore plus finement la taille du graphe partiel de budget calculé pendant la phase adverse de la méthode de Benders adverse étendue. Cela pourrait par exemple être fait en utilisant une méthode de sélection dynamique qui ferait varier les probabilités de sélection des prédécesseurs en fonction du nombre d'arcs qui a déjà été sélectionnés et de l'avancement du *backtracking*. Nous pensons également qu'il est possible d'améliorer les résultats en passant par des approches de compilation de connaissances, afin de trouver un langage de représentation des scénarios un peu plus compact que les graphes partiels de budget, et d'adapter notre problème maître afin de gagner en temps de calcul lors de sa résolution.

# Conclusion

Dans cette thèse, nous avons proposé de nouveaux outils et méthodologies à la fois pour l'aide à la décision pour des cas industriels pratiques et pour la résolution théorique de certains problèmes d'optimisation robuste.

Dans le chapitre 1, une revue de littérature est présentée, dans laquelle nous introduisons les concepts et contextes théoriques dans lesquels se placent les travaux présentés dans ce manuscrit. Une première contribution sur l'analyse de criticité d'un PERT généralisé avec longueurs des arcs incertaines est présentée.

Nous avons présenté dans le chapitre 2 une méthode de voisinage étendue basée sur l'analyse des pics de consommation qui améliore significativement les performances du modèle de programmation par contraintes résolu par le solveur CP Optimizer ainsi que l'approche de [Gerhards 2020] sur des jeux de données issus de la littérature. Par ailleurs, les résultats dominent également ceux de l'heuristique actuellement utilisée par Airbus.

Dans le chapitre 3, nous avons introduit les arbres de décisions robustes, et les avons utilisé pour approcher problème d'ordonnancement à une machine dans lequel on cherche à minimiser le retard maximal. En comparant les arbres de décision robustes à un algorithme naïf qui réagit de manière aveugle à l'arrivée de nouvelles informations, on montre que les arbres -et la sélection d'informations pertinentes- permettent à la fois de calculer un nombre raisonnable de solutions adaptées à des sous-ensembles de scénarios, et proposent une manière stable de passer d'une solution à une autre, notamment lorsque les intervalles d'incertitudes sont larges.

Ensuite, dans le chapitre 4 nous avons proposé une manière d'appliquer le concept d'arbres de décision robustes introduit dans le chapitre 3 au problème d'ordonnancement de ligne d'assemblage aéronautiques présenté dans le chapitre 2. Une fois encore, on compare favorablement les arbres de décisions robustes avec un algorithme réactif cette fois sur des instances issues de la littérature, où l'on génère des scénarios suivant plusieurs lois de probabilité et avec plusieurs jeux de paramètres, et sur des instances réelles fournies par Airbus.

Finalement, le chapitre 5 a porté quant à lui sur un problème de planification de production avec budget d'incertitude pour lequel nous avons proposé une variante de la méthode de Benders adverse. Nous concluons avec des résultats expérimentaux montrant que notre variante semble plus efficace que la méthode de Benders adverse classique pour résoudre ce problème, à la fois du point de vue du nombre d'itérations que du temps de calcul quand le budget d'incertitude n'est pas trop élevé.

## Perspectives

Les contributions de cette thèse ouvrent la voie à de nombreuses perspectives.

Nous présentons dans ce qui suit les travaux en cours ainsi que les futures ouvertures que nous envisageons d'investiguer.

Dans le premier chapitre nous proposons quelques résultats de complexité sur le calcul des dates de départ au plus tôt, au plus tard et la marge de tâches dans un contexte de contraintes de précédences généralisées avec de l'incertitude portant sur les délais entre les tâches. Nous avons montré que le temps de calcul pour $est^{min}$, $est^{max}$ et $lst^{min}$ était polynomial. Cependant la complexité de calcul de $lst^{max}$ et $f^{max}$ reste encore ouverte et semble être une perspective de recherche théorique intéressante.

Concernant les arbres de décision robustes présentés dans les chapitres 3 et 4, nous aimerions les mettre en avant dans le cadre du projet ANR *PER4MANCE*. Tout d'abord, il serait intéressant de voir comment ils pourraient être mis en place en pratique sur une ligne d'assemblage aéronautique, d'abord de manière expérimentale et d'observer leurs effets et impacts de manière concrète. Ensuite, j'ai évoqué dans l'introduction un autre doctorant qui lui s'intéresse plus à des problématiques de re-optimisation en cas d'aléa à fort impact et non prévu sur une ligne d'assemblage. Nous avons pour projet de mettre en place un outil plus général, permettant de lier ses travaux et les miens, permettant de prendre en compte n'importe quel type d'incertitudes.

Finalement, les travaux présentés dans le chapitre 5 sont encore préliminaires, et nous pensons qu'il est possible d'améliorer encore les résultats obtenus en maîtrisant encore plus finement les sous-ensembles de scénarios calculés lors de la phase adverse de notre méthode. Nous envisageons également d'appliquer cette approche à d'autres problèmes d'optimisation robuste avec budget d'incertitude, et de généraliser le concept de graphe de budget partiel.

# Bibliographie


[Al-Fawzan 2005] Mohammad A Al-Fawzan et Mohamed Haouari. A bi-objective model for robust resource-constrained project scheduling. International Journal of production economics, vol. 96, no. 2, pages 175–187, 2005. (Cité en page 14.)

[Alem 2018] Douglas Alem, Eduardo Curcio, Pedro Amorim et Bernardo Almada-Lobo. A computational study of the general lot-sizing and scheduling model under demand uncertainty via robust and stochastic approaches. Computers & Operations Research, vol. 90, pages 125–141, 2018. (Cité en page 92.)

[Aloulou 2008] Mohamed Ali Aloulou et Federico Della Croce. Complexity of single machine scheduling problems under scenario-based uncertainty. Operations Research Letters, vol. 36, no. 3, pages 338–342, 2008. (Cité en pages 12 et 43.)

[Andres 2008] Carlos Andres, Cristobal Miralles et Rafael Pastor. Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. European Journal of Operational Research, vol. 187, no. 3, pages 1212–1223, 2008. (Cité en page 22.)

[Arkhipov 2018] Dmitry Arkhipov, Olga Battaïa, Julien Cegarra et Alexander Lazarev. Operator assignment problem in aircraft assembly lines : a new planning approach taking into account economic and ergonomic constraints. Procedia CIRP, vol. 76, pages 63–66, 2018. 7th CIRP Conference on Assembly Technologies and Systems (CATS 2018). (Cité en page 39.)

[Artigues 2013a] C. Artigues et E. Hébrard. MIP Relaxation and Large Neighborhood Search for a Multi-Mode Resource-Constrained Multi-Project Scheduling Problem. Dans 6th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA), pages 814–819, Ghent, Belgium, 2013. (Cité en page 39.)

[Artigues 2013b] Christian Artigues, Roel Leus et Fabrice Talla Nobibon. Robust optimization for resource-constrained project scheduling with uncertain activity durations. Flexible Services and Manufacturing Journal, vol. 25, no. 1, pages 175–205, 2013. (Cité en page 13.)

[Assavapokee 2008] Tiravat Assavapokee, Matthew J Realff, Jane C Ammons et I-Hsuan Hong. Scenario relaxation algorithm for finite scenario-based min–max regret and min–max relative regret robust optimization. Computers & operations research, vol. 35, no. 6, pages 2093–2102, 2008. (Cité en pages 13 et 90.)

[Atamtürk 2007] Alper Atamtürk et Muhong Zhang. Two-stage robust network flow and design under demand uncertainty. Operations Research, vol. 55, no. 4, pages 662–673, 2007. (Cité en page 12.)

[Ayoub 2016] Josette Ayoub et Michael Poss. Decomposition for adjustable robust linear optimization subject to uncertainty polytope. Computational Management Science, vol. 13, no. 2, pages 219–239, 2016. (Cité en page 12.)

[Ben-Tal 2004] Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer et Arkadi Nemirovski. Adjustable robust solutions of uncertain linear programs. Mathematical programming, vol. 99, no. 2, pages 351–376, 2004. (Cité en page 11.)

[Ben-Tal 2005] Aharon Ben-Tal, Boaz Golany, Arkadi Nemirovski et Jean-Philippe Vial. Retailer-supplier flexible commitments contracts : A robust optimization approach. Manufacturing & Service Operations Management, vol. 7, no. 3, pages 248–271, 2005. (Cité en page 92.)

[Bertsimas 2004] Dimitris Bertsimas et Melvyn Sim. The price of robustness. Operations research, vol. 52, no. 1, pages 35–53, 2004. (Cité en page 10.)

[Bertsimas 2010] Dimitris Bertsimas et Constantine Caramanis. Finite adaptability in multistage linear optimization. IEEE Transactions on Automatic Control, vol. 55, no. 12, pages 2751–2766, 2010. (Cité en page 11.)

[Bertsimas 2011] Dimitris Bertsimas, David B Brown et Constantine Caramanis. Theory and applications of robust optimization. SIAM review, vol. 53, no. 3, pages 464–501, 2011. (Cité en page 41.)

[Bertsimas 2012] Dimitris Bertsimas, Eugene Litvinov, Xu Andy Sun, Jinye Zhao et Tongxin Zheng. Adaptive robust optimization for the security constrained unit commitment problem. IEEE transactions on power systems, vol. 28, no. 1, pages 52–63, 2012. (Cité en page 12.)

[Bitran 1982] Gabriel R Bitran et Horacio H Yanasse. Computational complexity of the capacitated lot size problem. Management Science, vol. 28, no. 10, pages 1174–1186, 1982. (Cité en page 90.)

[Borreguero 2015] T Borreguero, F Mas, JL Menéndez et MA Barreda. Enhanced assembly line balancing and scheduling methodology for the aeronautical industry. Procedia engineering, vol. 132, pages 990–997, 2015. (Cité en pages 22 et 83.)

[Borreguero 2021] Tamara Borreguero, Tom Portoleau, Alvaro García Sánchez, Miguel Ortega Mier et Pierre Lopez. Exact and heuristic methods for an aeronautical assembly line time-constrained scheduling problem with multiple modes and a resource leveling objective. Rapport technique, 2021. Technical report – hal-03344445. (Cité en pages 2, 21, 26, 35, 36 et 80.)

[Bougeret 2019] Marin Bougeret, Artur Alves Pessoa et Michael Poss. Robust scheduling with budgeted uncertainty. Discrete applied mathematics, vol. 261, pages 93–107, 2019. (Cité en page 12.)

[Brandimarte 2006] Paolo Brandimarte. Multi-item capacitated lot-sizing with demand uncertainty. International Journal of Production Research, vol. 44, no. 15, pages 2997–3022, 2006. (Cité en page 91.)

[Bruni 2018] Maria Elena Bruni, L Di Puglia Pugliese, Patrizia Beraldi et Francesca Guerriero. A computational study of exact approaches for the adjustable robust resource-constrained project scheduling problem. Computers & Operations Research, vol. 99, pages 178–190, 2018. (Cité en page 14.)

[Calafiore 2005] Giuseppe Calafiore et Marco C Campi. On two-stage portfolio allocation problems with affine recourse. Dans Proceedings of the 44th IEEE Conference on Decision and Control, pages 8042–8047. IEEE, 2005. (Cité en page 12.)

[Chtourou 2008] Hédi Chtourou et Mohamed Haouari. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. Computers & industrial engineering, vol. 55, no. 1, pages 183–194, 2008. (Cité en page 14.)

[Coelho 2011] José Coelho et Mario Vanhoucke. Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. European Journal of Operational Research, vol. 213, no. 1, pages 73–82, 2011. (Cité en page 8.)

[Cohen 2021] Izack Cohen, Krzysztof Postek et Shimrit Shtern. An adaptive robust optimization model for parallel machine scheduling. arXiv preprint arXiv :2102.08677, 2021. (Cité en page 12.)

[Coughlan 2015] Eamonn T Coughlan, Marco E Lübbecke et Jens Schulz. A branch-price-and-cut algorithm for multi-mode resource leveling. European Journal of Operational Research, vol. 245, no. 1, pages 70–80, 2015. (Cité en page 8.)

[Davari 2017] Morteza Davari et Erik Demeulemeester. The proactive and reactive resource-constrained project scheduling problem. Journal of Scheduling, pages 1–27, 2017. (Cité en pages 13 et 64.)

[Davari 2019] Morteza Davari et Erik Demeulemeester. Important classes of reactions for the proactive and reactive resource-constrained project scheduling problem. Annals of Operations Research, vol. 274, no. 1-2, pages 187–210, 2019. (Cité en pages 13 et 64.)

[De Reyck 1999] Bert De Reyck et Willy Herroelen. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research, vol. 119, no. 2, pages 538–556, 1999. (Cité en page 8.)

[Dearden 2003] Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David E Smith et Rich Washington. Incremental contingency planning. Dans ICAPS-03 Workshop on Planning under Uncertainty, 2003. (Cité en page 13.)

[Demeulemeester 1995] Erik Demeulemeester. Minimizing resource availability costs in time-limited project networks. Management Science, vol. 41, no. 10, pages 1590–1598, 1995. (Cité en page 8.)

[Demeulemeester 2002] Erik Demeulemeester et Willy Herroelen. Project scheduling : A research handbook. International Series in Operations Research & Management Science. Springer, 01 2002. (Cité en page 8.)

[Drexl 1993] Andreas Drexl et Juergen Gruenewald. Non-preemptive multi-mode resource-constrained project scheduling. IIE Transactions, vol. 25, no. 5, pages 74–81, 1993. (Cité en page 8.)

[Drummond 1994] Mark Drummond, John Bresina et Keith Swanson. Just-in-case scheduling. Dans AAAI, volume 94, pages 1098–1104, 1994. (Cité en page 13.)

[Fortin 2010] Jérôme Fortin, Paweł Zieliński, Didier Dubois et Hélène Fargier. Criticality analysis of activity networks under interval uncertainty. Journal of Scheduling, vol. 13, no. 6, pages 609–627, 2010. (Cité en pages 15, 17 et 19.)

[Gerhards 2020] Patrick Gerhards. The multi-mode resource investment problem : a benchmark library and a computational study of lower and upper bounds. OR Spectrum, vol. 42, no. 4, pages 901–933, 2020. (Cité en pages 8, 38, 39, 66, 80 et 105.)

[Godard 2005] Daniel Godard, Philippe Laborie et Wim Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling. Dans ICAPS, volume 5, pages 81–89, 2005. (Cité en page 39.)

[Guillaume 2020] Romain Guillaume, Adam Kasperski et Paweł Zieliński. Production planning under demand uncertainty : a budgeted uncertainty approach. Dans Operations Research Proceedings 2019, pages 431–437. Springer, 2020. (Cité en pages 92, 94 et 96.)

[Hanasusanto 2015] Grani A Hanasusanto, Daniel Kuhn et Wolfram Wiesemann. K-adaptability in two-stage robust binary programming. Operations Research, vol. 63, no. 4, pages 877–891, 2015. (Cité en page 11.)

[Hartmann 2022] Sönke Hartmann et Dirk Briskorn. An updated survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research, vol. 297, no. 1, pages 1–14, 2022. (Cité en page 7.)

[Hu 2016] Zhengyang Hu et Guiping Hu. A two-stage stochastic programming model for lot-sizing and scheduling under uncertainty. International Journal of Production Economics, vol. 180, pages 198–207, 2016. (Cité en page 91.)

[Jackson 1955] James R Jackson. Scheduling a production line to minimize maximum tardiness. Rapport technique, CALIFORNIA UNIV LOS ANGELES NUMERICAL ANALYSIS RESEARCH, 1955. (Cité en page 6.)

[Jackson 1956] James R Jacksonet al. An extension of Johnson's results on job IDT scheduling. Naval Research Logistics Quarterly, vol. 3, no. 3, pages 201–203, 1956. (Cité en page 6.)

[Johnson 1954] Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. Naval research logistics quarterly, vol. 1, no. 1, pages 61–68, 1954. (Cité en page 6.)

[Kagermann 2013] H. Kagermann et W. Wahlster. Recommendations for implementing the strategic initiative Industry 4.0 : Securing the future of German manufacturing industry. Final report of the Industry 4.0 Working Group, 2013. (Cité en page 22.)

[Kelley Jr 1959] James E Kelley Jr et Morgan R Walker. Critical-path planning and scheduling. Dans Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference, pages 160–173, 1959. (Cité en page 72.)

[Kolisch 1995] Rainer Kolisch, Arno Sprecher et Andreas Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. Management science, vol. 41, no. 10, pages 1693–1703, 1995. (Cité en page 78.)

[Kolisch 1997a] Rainer Kolisch et Andreas Drexl. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions, vol. 29, no. 11, pages 987–999, 1997. (Cité en page 25.)

[Kolisch 1997b] Rainer Kolisch et Arno Sprecher. PSPLIB-a project scheduling problem library : OR software-ORSEP Operations research software exchange program. European Journal of Operational Research, vol. 96, no. 1, pages 205–216, 1997. (Cité en page 59.)

[Kouvelis 1997] P. Kouvelis et G. Yu. Robust discrete optimization and its applications. Kluwer Academic Publishers, 1997. (Cité en pages 9, 46 et 64.)

[Laborie 2007] Philippe Laborie et Daniel Godard. Self-adapting large neighborhood search : Application to single-mode scheduling problems. Proceedings MISTA-07, Paris, vol. 8, 2007. (Cité en page 29.)

[Laborie 2018] Philippe Laborie, Jérôme Rogerie, Paul Shaw et Petr Vilím. IBM ILOG CP Optimizer for scheduling. Constraints, vol. 23, no. 2, pages 210–250, 2018. (Cité en page 27.)

[Mani 2006] Murari Mani, Ashish K Sing et Michael Orshansky. Joint design-time and post-silicon minimization of parametric yield loss using adjustable robust optimization. Dans Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design, pages 19–26, 2006. (Cité en page 12.)

[Mas 2014] Fernando Mas, José Luis Menéndez, Manuel Oliva, Javier Servan, Rebeca Arista et Carmelo Del Valle. Design within Complex Environments : Collaborative Engineering in the aerospace industry. Dans Information System Development, pages 197–205. Springer, 2014. (Cité en page 22.)

[Mas 2015] F. Mas, R. Arista, M. Oliva, B. Hiebert, I. Gilkerson et J. Ríos. A review of PLM Impact on US and EU Aerospace Industry. Procedia Engineering, vol. 132, pages 1053–1060, 2015. (Cité en page 22.)

[McNaughton 1959] Robert McNaughton. Scheduling with deadlines and loss functions. Management science, vol. 6, no. 1, pages 1–12, 1959. (Cité en page 6.)

[Meuleau 2002] Nicolas Meuleau et David E Smith. Optimal limited contingency planning. Dans Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence, pages 417–426. Morgan Kaufmann Publishers Inc., 2002. (Cité en page 13.)

[Minoux 2007] Michel Minoux. Models and algorithms for robust PERT scheduling with time-dependent task durations. Vietnam J. Math, vol. 35, no. 4, pages 387–398, 2007. (Cité en page 15.)

[Mogaadi 2016] Hayet Mogaadi et Besma Fayech Chaar. Scenario-based evolutionary approach for robust RCPSP. Dans Proceedings of the Second International Afro-European Conference for Industrial Advancement AECIA 2015, pages 45–55. Springer, 2016. (Cité en page 13.)

[Morris 1987] Peter WG Morris et George H Hough. The anatomy of major projects : A study of the reality of project management. 1987. (Cité en page 5.)

[Mudchanatongsuk 2005] S Mudchanatongsuk et F Ordonez. and J. Liu, Robust Solutions For Network Design Under Transportation Cost And Demand Uncertainty. Rapport technique, USC ISE Working paper 2005–05, 2005. (Cité en page 12.)

[Mutapcic 2009] Almir Mutapcic et Stephen Boyd. Cutting-set methods for robust convex optimization with pessimizing oracles. Optimization Methods and Software, vol. 24, no. 3, pages 381–406, 2009. (Cité en page 90.)

[Nikulin 2006] Yury Nikulin. Robustness in combinatorial optimization and scheduling theory : An extended annotated bibliography. Rapport technique, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 2006. (Cité en page 13.)

[Ordónez 2007] Fernando Ordónez et Jiamin Zhao. Robust capacity expansion of network flows. Networks : An International Journal, vol. 50, no. 2, pages 136–145, 2007. (Cité en page 12.)

[Palacio 2017] Juan D Palacio et Olga L Larrea. A lexicographic approach to the robust resource-constrained project scheduling problem. International Transactions in Operational Research, vol. 24, no. 1-2, pages 143–157, 2017. (Cité en page 14.)

[Palpant 2004] Mireille Palpant, Christian Artigues et Philippe Michelon. LSSPER : Solving the resource-constrained project scheduling problem with large neighbourhood search. Annals of Operations Research, vol. 131, no. 1-4, pages 237–257, 2004. (Cité en pages 28 et 29.)

[Pätzold 2020] Julius Pätzold et Anita Schöbel. Approximate cutting plane approaches for exact solutions to robust optimization problems. European Journal of Operational Research, vol. 284, no. 1, pages 20–30, 2020. (Cité en page 90.)

[Pinedo 2012] Michael L Pinedo. Scheduling, volume 29. Springer, 2012. (Cité en page 6.)

[Portoleau 2020a] Tom Portoleau, Christian Artigues et Romain Guillaume. Arbres de décision robustes pour l'ordonnancement proactif/reactif sous incertitude. Dans 21ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Montpellier, France, 2020. (Cité en pages 2 et 41.)

[Portoleau 2020b] Tom Portoleau, Christian Artigues et Romain Guillaume. Robust Predictive-Reactive Scheduling : an Information-Based Decision Tree Model. Dans International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, pages 479–492. Springer, 2020. (Cité en pages 2 et 41.)

[Portoleau 2021a] Tom Portoleau, Christian Artigues et Romain Guillaume. Arbres de décision robustes pour le RCPSP multi-mode. Dans 22ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Mulhouse (en distanciel), France, 2021. (Cité en pages 3 et 59.)

[Portoleau 2021b] Tom Portoleau, Christian Artigues et Romain Guillaume. Decision trees for robust scheduling. Dans 17th Internatonal Workshop on Project Manage-

ment and Scheduling, pages 265–268, Toulouse (full online), France, 2021. (Cité en pages 2 et 41.)

[Portoleau 2021c] Tom Portoleau, Christian Artigues et Romain Guillaume. Robust Decision Tree for MMRCPSP. Dans 31st European Conference on Operational Research (EURO, Athens, Greece, 2021. (Cité en pages 3 et 59.)

[Portoleau 2021d] Tom Portoleau, Christian Artigues et Romain Guillaume. Robust decision trees for the multi-mode project scheduling problem with a resource investment objective and uncertain activity duration. Rapport technique, LAAS report 22002, hal-03502505, 2021. (Cité en pages 3 et 59.)

[Portoleau 2022a] Tom Portoleau, Christian Artigues, Tamara Borreguero Sanchidriàn, Alvaro García Sánchez, Miguel Ortega Mier et Pierre Lopez. Large neighborhood search for a multi-mode resource constrained scheduling problem with resource leveling objective. Dans 18th Internatonal Workshop on Project Management and Scheduling, 2022. (Cité en pages 2 et 21.)

[Portoleau 2022b] Tom Portoleau, Romain Guillaume et Christian Artigues. Une variante de la méthode de Benders adverse pour le problème de lot-sizing robuste avec budget d'incertitude. Dans 23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Lyon, France, 2022. 6 pages. (Cité en pages 3 et 89.)

[Postek 2016] Krzysztof Postek et Dick den Hertog. Multistage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set. INFORMS Journal on Computing, vol. 28, no. 3, pages 553–574, 2016. (Cité en page 11.)

[Postek 2019] Krzysztof Postek, Dick Den Hertog, Jarl Kind et Chris Pustjens. Adjustable robust strategies for flood protection. Omega, vol. 82, pages 142–154, 2019. (Cité en page 12.)

[Potts 2009] Chris N Potts et Vitaly A Strusevich. Fifty years of scheduling : a survey of milestones. Journal of the Operational Research Society, vol. 60, no. 1, pages S41–S68, 2009. (Cité en page 6.)

[PSPLIB 2020] PSPLIB. Project scheduling problem library. http ://www.om-db.wi.tum.de/psplib/main.html, 2020. (Cité en pages 77 et 78.)

[Quezada 2022] Franco Quezada, Céline Gicquel et Safia Kedad-Sidhoum. Combining polyhedral approaches and stochastic dual dynamic integer programming for solving the uncapacitated lot-sizing problem under uncertainty. INFORMS Journal on Computing, vol. 34, no. 2, pages 1024–1041, 2022. (Cité en page 92.)

[Rajendran 1999] Chandrasekharan Rajendran et Oliver Holthaus. A comparative study of dispatching rules in dynamic flowshops and jobshops. European Journal of Operational Research, vol. 116, no. 1, pages 156 – 170, 1999. (Cité en page 13.)

[Roels 2006] Guillaume Roels et Georgia Perakis. The price of information : Inventory management with limited information about demand. Manufacturing & Service Operations Management, vol. 8, no. 1, pages 98–117, 2006. (Cité en page 92.)

[Sabuncuoğlu 2008] İhsan Sabuncuoğlu, Yasin Gocgun et Erdal Erel. Backtracking and exchange of information : Methods to enhance a beam search algorithm for assembly line scheduling. European Journal of Operational Research, vol. 186, no. 3, pages 915–930, 2008. (Cité en page 23.)

[Santos 2018] Marcio Costa Santos, Michael Poss et Dritan Nace. A perfect information lower bound for robust lot-sizing problems. Annals of Operations Research, vol. 271, no. 2, pages 887–913, 2018. (Cité en page 92.)

[Schläpfer 2014] Ralf C Schläpfer, Markus Koch et Philipp Merkofer. Industry 4.0. Challenges and solutions for the digital transformation and use of exponential technologies. Rapport technique, Deloitte, Zurich, 2014. (Cité en page 22.)

[Schwindt 2015] Christoph Schwindt et J. Zimmermann. Handbook on project management and scheduling vol.1. Springer, 01 2015. (Cité en page 8.)

[Shahanaghi 2010] Kamran Shahanaghi, Abdolmajid Yolmeh et Unes Bahalke. Scheduling and balancing assembly lines with the task deterioration effect. Proceedings of the Institution of Mechanical Engineers, Part B : Journal of Engineering Manufacture, vol. 224, no. 7, pages 1145–1153, 2010. (Cité en page 22.)

[Silva 2018] Marco Silva, Michael Poss et Nelson Maculan. Solving the bifurcated and nonbifurcated robust network loading problem with k-adaptive routing. Networks, vol. 72, no. 1, pages 151–170, 2018. (Cité en page 12.)

[Sivasankaran 2014] Panneerselvam Sivasankaran et P Shahabudeen. Literature review of assembly line balancing problems. The International Journal of Advanced Manufacturing Technology, vol. 73, no. 9, pages 1665–1694, 2014. (Cité en page 22.)

[Smith 1956] IWE Smith. Various optimizers for single stage production. Nat. Res. Logist. Q. 3, vol. 5946, 1956. (Cité en page 6.)

[Snyder 1987] James R Snyder. Modern Project Management : how Did We Get Here–where Do We Go ? 1987. (Cité en page 5.)

[Van de Vonder 2007] Stijn Van de Vonder, Erik Demeulemeester et Willy Herroelen. A classification of predictive-reactive project scheduling procedures. Journal of scheduling, vol. 10, no. 3, pages 195–207, 2007. (Cité en page 13.)

[Węglarz 2011] Jan Węglarz, Joanna Józefowska, Marek Mika et Grzegorz Waligóra. Project scheduling with finite or infinite number of activity processing modes – A survey. European Journal of Operational Research, vol. 208, no. 3, pages 177–205, 2011. (Cité en page 8.)

[Zhang 2000] Yuanhui Zhang, Peter B Luh, Kiyoshi Yoneda, Toshiyuki Kano et Yuji Kyoya. Mixed-model assembly line scheduling using the Lagrangian relaxation technique. Iie Transactions, vol. 32, no. 2, pages 125–134, 2000. (Cité en page 23.)

[Zou 2019] Jikai Zou, Shabbir Ahmed et Xu Andy Sun. Stochastic dual dynamic integer programming. Mathematical Programming, vol. 175, no. 1, pages 461–502, 2019. (Cité en page 92.)

# Annexe

Cet Annexe contient trois publications auxquelles j'ai contribué pendant la durée de la thèse mais qui ne sont pas en lien avec celle-ci :

— Hugo Gilbert, Tom Portoleau, Olivier Spanjaard. Beyond Pairwise Comparisons in Social Choice : A Setwise Kemeny Aggregation Problem. In. Thirty-Fourth AAAI Conference on Artificial Intelligence. AAAI 2020, New York, NY, USA, AAAI Press p. 1982-1989 (2020).

— Valentin Antuori, Tom Portoleau, Louis Rivière, Emmanuel Hebrard. On How Turing and Singleton Arc Consistency Broke the Enigma Code. In Laurent D. Michel, editor, 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France. LIPIcs 210, Schloss Dagstuhl - Leibniz-Zentrum für Informatik p. 13 :1-13 :16 (2021).

— Hugo Gilbert, Tom Portoleau, Olivier Spanjaard : Beyond pairwise comparisons in social choice : A setwise Kemeny aggregation problem. Theoretical Computer Science 904 : p. 27-47 (2022)

# Beyond Pairwise Comparisons in Social Choice: A Setwise Kemeny Aggregation Problem

**Hugo Gilbert**
Gran Sasso Science Institute
67100 L'Aquila, Italy
hugo.gilbert@gssi.it

**Tom Portoleau**
LAAS-CNRS, IRIT-CNRS
Université de Toulouse
31400 Toulouse, France
tom.portoleau@laas.fr

**Olivier Spanjaard**
Sorbonne Université
CNRS, LIP6,
75005 Paris, France
olivier.spanjaard@lip6.fr

**Abstract**

In this paper, we advocate the use of setwise contests for aggregating a set of input rankings into an output ranking. We propose a generalization of the Kemeny rule where one minimizes the number of $k$-wise disagreements instead of pairwise disagreements (one counts 1 disagreement each time the top choice in a subset of alternatives of cardinality at most $k$ differs between an input ranking and the output ranking). After an algorithmic study of this $k$-wise Kemeny aggregation problem, we introduce a $k$-wise counterpart of the majority graph. It reveals useful to divide the aggregation problem into several sub-problems. We conclude with numerical tests.

## 1 Introduction

Rank aggregation aims at producing a single ranking from a collection of rankings of a fixed set of alternatives. In social choice theory (e.g., Moulin 1991), where the alternatives are candidates to an election and each ranking represents the preferences of a voter, aggregation rules are called *Social Welfare Functions* (SWFs). Apart from social choice, rank aggregation has proved useful in many applications, including preference learning (Cheng and Hüllermeier 2009; Clémençon, Korba, and Sibony 2018), collaborative filtering (Wang et al. 2014), genetic map creation (Jackson, Schnable, and Aluru 2008), similarity search in databases systems (Fagin, Kumar, and Sivakumar 2003) and design of web search engines (Altman and Tennenholtz 2008; Dwork et al. 2001). In the following, we use interchangeably the terms "input rankings" and "preferences", "output ranking" and "consensus ranking", as well as "alternatives" and "'candidates".

The well-known Arrow's impossibility theorem states that there exists no aggregation rule satisfying a small set of desirable properties (Arrow 1950). In the absense of an "ideal" rule, various aggregation rules have been proposed and studied. Following Fishburn's classification (1977), we can distinguish between the SWFs for which the output ranking can be computed from the *majority graph* alone, those for which the output ranking can be computed from the *weighted majority graph* alone, and all other SWFs[1]. The majority graph is obtained from the input rankings by defining one vertex per alternative $c$ and by adding an edge from $c$ to $c'$ if $c$ is preferred to $c'$ in a strict majority of input rankings. In the weighted majority graph, each edge is weighted by the majority margin. The many SWFs that rely on these graphs alone take therefore only pairwise comparisons into account to determine an output ranking. For a compendium of these SWFs, we refer to the book by Brandt et al. (2016).

The importance of this class of SWFs can be explained by their connection with the *Condorcet consistency* property, stating: if there is a Condorcet winner (i.e., an alternative with outgoing edges to every other ones in the majority graph), then it should be ranked first in the output ranking. Nevertheless, as shown by Baldiga and Green (2013), the lack of Condorcet consistency is not necessarily a bad thing, because this property may come into contradiction with the objective of maximizing voters' agreement with the output ranking. The following example illustrates this point.

**Example 1** (Baldiga and Green, 2013)**.** *Consider an election with 100 voters and 3 candidates $c_1, c_2, c_3$, where 49 voters have preferences $c_1 \succ c_2 \succ c_3$, 48 have preferences $c_3 \succ c_2 \succ c_1$ and 3 have preferences $c_2 \succ c_3 \succ c_1$. Candidate $c_2$ is the Condorcet winner, but is the top choice of only 3 voters. In contrast, candidate $c_1$ is in slight minority against $c_2$ and $c_3$, but $c_1$ is the top choice of 49 voters. This massive gain in agreement may justify to put $c_1$ instead of $c_2$ in first position of the output ranking.*

Following Baldiga and Green (2013), we propose to handle this tension between the pairwise comparisons (leading to ranking $c_2$ first) and the plurality choice (leading to ranking $c_1$ first) by using SWFs that take into account not only pairwise comparisons but *setwise* contests. More precisely, given input rankings on a set $C$ of candidates and $k \in \{2, \ldots, |C|\}$, the idea is to consider the plurality score of each candidate $c$ for each subset $S \subseteq C$ such that $2 \leq |S| \leq k$, where the plurality score of $c$ for $S$ is the number of voters for which $c$ is the top choice in $S$. The results of setwise con-



[1] Fishburn's classification actually applies to social *choice* functions, which prescribe a subset of winning alternatives from a collection of rankings, but the extension to SWFs is straightforward.

Table 1: Results of setwise contests in Example 1.

| set | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|
| $\{c_1, c_2\}$ | 49 | 51 | – |
| $\{c_1, c_3\}$ | 49 | – | 51 |
| $\{c_2, c_3\}$ | – | 52 | 48 |
| $\{c_1, c_2, c_3\}$ | 49 | 3 | 48 |

tests for the preferences of Example 1 are given in Table 1 for $k=3$. Note that the three top rows obviously encode the same information as the weighted majority graph while the bottom row makes it possible to detect the tension between the pairwise comparisons and the plurality choice.

One can then define a new class of SWFs, those that rely on the results of setwise contests alone to determine an output ranking. The many works that have been carried out regarding voting rules based on the (weighted) majority graph can be revisited in this broader setting. This line of research has already been investigated by Lu and Boutilier (2010) and Baldiga and Green (2013). However, note that both of these works consider a setting where candidates may become unavailable after voters express their preferences. We do not make this assumption. We indeed believe that this new class of SWFs makes sense in the standard setting where the set of candidates is known and deterministic, as it amounts to generate an output ranking by examining the choices that are made by the voters on subsets of candidates of various sizes (while usually only pairwise choices are considered).

A natural SWF in this class consists in determining an output ranking that minimizes the number of disagreements with the results of setwise contests for sets of cardinality at most $k$. This is a $k$-wise generalization of the Kemeny rule, obtained as a special case for $k=2$. We recall that the Kemeny rule consists in producing a ranking that minimizes the number of *pairwise* disagreements (Kemeny 1959).

**Example 2.** *Let us come back to Example 1 and assume that we use the 3-wise Kemeny rule. Consider the output ranking* $r = c_1 \succ c_2 \succ c_3$. *For set* $S = \{c_1, c_2\}$, *the number of disagreements with the results of setwise contests is* 51 *because* $c_2$ *is the top choice in* $S$ *for* 51 *voters (see Table 1) while it is* $c_1$ *for* $r$. *Similarly, the number of disagreements induced by* $\{c_1, c_3\}$, $\{c_2, c_3\}$ *and* $\{c_1, c_2, c_3\}$ *are respectively* 51, 48 *and* 3+48. *The total number of disagreements is thus* 51+51+48+3+48 = 201. *This is actually the minimum number of disagreements that can be achieved for these input rankings, which makes* $r$ *the* 3*-wise Kemeny ranking.*

The purpose of this paper is to study the $k$-wise Kemeny aggregation problem. Section 2 formally defines the problem and reports on related work. Section 3 is devoted to some axiomatic considerations of the corresponding voting rule, and to an algorithmic study of the problem. We then investigate a $k$-wise variant of the majority graph in Section 4. We prove that determining this graph is easy for $k=3$ but becomes NP-hard for $k>3$, and we show how to use it in a preprocessing step to speed up the computation of the output ranking. Numerical tests are presented in Section 5. Due to lack of space, missing proofs can be found in the long version of the paper (Gilbert, Portoleau, and Spanjaard 2019).

## 2 Preliminaries

Adopting the terminology of social choice theory, we consider an election with a set $V$ of $n$ voters and a set $C$ of $m$ candidates. Each voter $v$ has a complete and transitive preference order $r_v$ over candidates (also called ranking). The collection of these rankings defines a preference profile $\mathcal{P}$.

### Notations and Definitions

Let us introduce some notations related to rankings. We denote by $\mathcal{R}(C)$ the set of $m!$ rankings over $C$. Given a ranking $r$ and two candidates $c$ and $c'$, we write $c \succ_r c'$ if $c$ is in a higher position than $c'$ in $r$. Given a ranking $r$ and a candidate $c$, $\mathtt{rk}(c, r)$ denotes the rank of $c$ in $r$. For instance, $\mathtt{rk}(c, r_v) = 1$ if $c$ is the preferred candidate of voter $v$ (the candidate ranked highest in $r_v$). Given a ranking $r$ and a set $S \subseteq C$, we define $r_S$ as the restriction of $r$ to $S$ and $t_r(S)$ as the top choice (i.e., preferred candidate) in $S$ according to $r$. Similarly, given a preference profile $\mathcal{P}$ and a set $S \subseteq C$, we define $\mathcal{P}_S$ as the restriction of $\mathcal{P}$ to $S$. Lastly, we denote by $\mathtt{tail}_k(r)$ (resp. $\mathtt{head}_k(r)$) the subranking compounded of the $k$ least (resp. most) preferred candidates in $r$.

We are interested in SWFs which, given a preference profile $\mathcal{P}$, should return a consensus ranking which yields a suitable compromise between the preferences in $\mathcal{P}$. One of the most well-known SWFs is the *Kemeny rule*, which selects a ranking $r$ with minimal Kendall tau distance to $\mathcal{P}$. Denoting by $\delta_{\text{KT}}(r, r')$ the Kendall tau distance between rankings $r$ and $r'$, the distance $\delta_{\text{KT}}(r, \mathcal{P})$ between a ranking $r$ and a profile $\mathcal{P}$ reads as:

$$\delta_{\text{KT}}(r, \mathcal{P}) = \sum_{r' \in \mathcal{P}} \delta_{\text{KT}}(r, r')$$

$$\text{where } \delta_{\text{KT}}(r, r') = \sum_{\{c,c'\} \subseteq C} \mathbb{1}_{t_r(\{c,c'\}) \neq t_{r'}(\{c,c'\})}$$

Stated differently, $\delta_{\text{KT}}$ measures the distance between two rankings by the number of pairwise disagreements between them. The distance between a ranking and a preference profile is then obtained by summation.

However, the Kendall tau distance only takes into account pairwise comparisons, which may entail counterintuitive results as illustrated by Example 1. To address this issue, the Kendall tau distance can be generalized to take into consideration disagreements on sets of cardinal greater than two. Given a set $S \subseteq C$ and $t \leq m$, we denote by $\Delta^t(S)$ the set of subsets of $S$ of cardinal lower than or equal to $t$, i.e., $\Delta^t(S) = \{S' \subseteq S \text{ s.t. } |S'| \leq t\}$. When $S$ is not specified, it is assumed to be $C$, i.e., $\Delta^t = \Delta^t(C)$. For $k \geq 2$, the $k$-wise Kendall tau distance $\delta^k_{\text{KT}}$ between $r$ and $r'$ is defined by:

$$\delta^k_{\text{KT}}(r, r') = \sum_{S \in \Delta^k} \mathbb{1}_{t_r(S) \neq t_{r'}(S)}$$

In other words, $\delta^k_{\text{KT}}$ measures the distance between two rankings by the number of top-choice disagreements on sets of cardinal lower than or equal to $k$.

Note that $\delta^k_{\text{KT}}$ has all the properties of a distance: non-negativity, identity of indiscernibles, symmetry and triangle inequality. Secondly, as mentioned in the introduction, we

have $\delta^2_{\text{KT}} = \delta_{\text{KT}}$. Thirdly and maybe most importantly, we point out that the distances induced by $\delta^k_{\text{KT}}(r, r')$ can be computed in $O(m^3)$ by using the following formula:

$$\delta^k_{\text{KT}}(r, r') = \sum_{\{c,c'\}\subseteq C} \mathbb{1}_{c \succ_r c'} \mathbb{1}_{c' \succ_{r'} c} |\Delta^{k-2}(B_c(r) \cap B_{c'}(r'))|$$

$$= \sum_{\{c,c'\}\subseteq C} \mathbb{1}_{c \succ_r c'} \mathbb{1}_{c' \succ_{r'} c} \sum_{i=0}^{k-2} \binom{|B_c(r) \cap B_{c'}(r')|}{i} \quad (1)$$

where $B_c(r) = \{x \in C \text{ s.t. } c \succ_r x\}$ is the set of candidates that are ranked below $c$ in $r$. Let us give some intuition for this formula. For any pair $\{c, c'\}$ of candidates such that $c \succ_r c'$ and $c' \succ_{r'} c$, we count the number of sets in $\Delta^k$ on which there is a disagreement because the top choice is $c$ for $r$ while it is $c'$ for $r'$. Such sets are of the form $S \cup \{c, c'\}$, where $S \in \Delta^{k-2}(B_c(r) \cap B_{c'}(r'))$, otherwise $c$ and $c'$ would not be the top choices. Hence the formula.

The distance $\delta^k_{\text{KT}}$ induces a new SWF, the $k$-wise Kemeny rule, which, given a profile $\mathcal{P}$, returns a ranking $r$ with minimal distance $\delta^k_{\text{KT}}$ to $\mathcal{P}$, where:

$$\delta^k_{\text{KT}}(r, \mathcal{P}) = \sum_{r' \in \mathcal{P}} \delta^k_{\text{KT}}(r, r')$$

Note that this coincides with the rule we used in the introduction, by commutativity of addition:

$$\sum_{r' \in \mathcal{P}} \sum_{S \in \Delta^k} \mathbb{1}_{t_r(S) \neq t_{r'}(S)} = \sum_{S \in \Delta^k} \sum_{r' \in \mathcal{P}} \mathbb{1}_{t_r(S) \neq t_{r'}(S)}$$

Determining a consensus ranking for this rule defines the $k$-wise Kemeny Aggregation Problem ($k$-KAP for short).

**k-WISE KEMENY AGGREGATION PROBLEM**
*INSTANCE:* A profile $\mathcal{P}$ with $n$ voters and $m$ candidates.
*SOLUTION:* A ranking $r$ of the $m$ candidates.
*MEASURE:* $\delta^k_{\text{KT}}(r, \mathcal{P})$ to minimize.

## Related Work

Several other variants of the Kemeny rule have been proposed in the literature, either to obtain generalizations able to deal with partial or weak orders (Dwork et al. 2001; Zwicker 2018), to penalize more some pairwise disagreements than others (Kumar and Vassilvitskii 2010), or to account for candidates that may become unavailable after voters express their preferences (Baldiga and Green 2013; Lu and Boutilier 2010).

Indeed, despite its popularity, the Kemeny rule has received several criticisms. One of them is that the Kendall tau distance counts equally the disagreements on every pair of candidates. This property is undesirable in many settings. For instance, with a web search engine, a disagreement on a pair of web pages with high positions in the considered rankings should have a higher cost than a disagreement on pairs of web pages with lower ones. This drawback motivated the introduction of weighted Kendall tau distances by Kumar and Vassilvitskii (2010). A more thorough comparison between our work and theirs can be found in the extended version of the paper (Gilbert, Portoleau, and Spanjaard 2019).

Let us illustrate with the following example, that the $k$-wise Kendall tau distance is also well suited to penalize more the disagreements involving alternatives at the top of the input rankings.

**Example 3.** *Consider rankings $r_1, r_2, r_3$ defined by $c_1 \succ_{r_1} c_2 \succ_{r_1} c_3$, $c_1 \succ_{r_2} c_3 \succ_{r_2} c_2$, and $c_2 \succ_{r_3} c_1 \succ_{r_3} c_3$. We have $\delta_{\text{KT}}(r_1, r_2) = \delta_{\text{KT}}(r_1, r_3) = 1$ while $\delta^3_{\text{KT}}(r_1, r_2) = 1 < 2 = \delta^3_{\text{KT}}(r_1, r_3)$ because $r_1$ and $r_3$ disagree on both subsets $\{c_1, c_2\}$ and $\{c_1, c_2, c_3\}$. Put another way, $\delta^3_{\text{KT}}(r_1, r_3) > \delta^3_{\text{KT}}(r_1, r_2)$ because $r_1$ and $r_3$ disagree on their top-ranked alternatives whereas $r_1$ and $r_2$ disagree on the alternatives ranked in the last places.*

The two works closest to ours are related to another extension of the Kemeny rule. This extension considers a setting in which, besides the fact that voters have preferences over a set $C$, the election will in fact occur on a subset $S \subseteq C$ drawn according to a probability distribution (Baldiga and Green 2013; Lu and Boutilier 2010). The optimization problem considered is then to find a consensus ranking $r$ which minimizes, in expectation, the number of voters' disagreements with the chosen candidate in $S$ (a voter $v$ disagrees if $t_{r_v}(S) \neq t_r(S)$). The differences between the work of Baldiga and Green (2013) and the one of Lu and Boutilier (2010) is then twofold. Firstly, while Baldiga and Green mostly focused on the axiomatic properties of this aggregation procedure, the work of Lu and Boutilier has more of an algorithmic flavor. Secondly, while Baldiga and Green mostly study a setting in which the probability $\mathbb{P}(S)$ of $S$ is only dependent on its cardinality (i.e., $\mathbb{P}(S)$ is only a function of $|S|$), Lu and Boutilier study a setting that can be viewed as a special case of the former, where each candidate is absent of $S$ independently of the others with a probability $p$ (i.e., $\mathbb{P}(S) = p^{|C \setminus S|}(1-p)^{|S|}$). The Kemeny aggregation problem can be formulated in both settings, either by defining $\mathbb{P}(S) = 0$ for $|S| \geq 3$, or by defining a probability $p$ that is "sufficiently high" w.r.t. the size of the instance (Lu and Boutilier 2010). Lu and Boutilier conjectured that the determination of a consensus ranking is NP-hard in their setting, designed an exact method based on mathematical programming, two approximation greedy algorithms and a PTAS.

Our model can be seen as a special case of the model of Baldiga and Green where the set $S$ is drawn uniformly at random within the set of subsets of $C$ of cardinal smaller than or equal to a given constant $k \geq 2$. While it cannot be casted in the specific setting studied by Lu and Boutilier, our model is closely related and may be used to obtain new insights on their work.

# 3 Aggregation with the $k$-wise Kemeny Rule

In this section, we investigate the axiomatic properties of the $k$-wise Kemeny rule, and then we turn to the algorithmic study of $k$-KAP.

## Axiomatic Properties of the $k$-wise Kemeny Rule

Several properties of the $k$-wise Kemeny rule have already been studied by Baldiga and Green (2013), because their setting includes the $k$-wise Kemeny rule as a special case.

Among other things, they showed that the rule is not *Condorcet consistent*. That is to say, a Condorcet winner may not be ranked first in any consensus ranking even when one exists, as illustrated by Example 2.

The authors also show that the $k$-wise Kemeny rule is neutral, i.e., all candidates are treated equally, and that for $k \geq 3$ it is different from any positional method or any method that uses only the pairwise majority margins (among which is the standard Kemeny rule). We provide here some additional properties satisfied by the $k$-wise Kemeny rule:

- *Monotonicity*: up-ranking cannot harm a winner; down-ranking cannot enable a loser to win.
- *Unanimity*: if all voters rank $c$ before $c'$, then $c$ is ranked before $c'$ in any consensus ranking.
- *Reinforcement*: let $\mathcal{R}^*_{\mathcal{P}}$ and $\mathcal{R}^*_{\mathcal{P}'}$ denote the sets of consensus rankings for preference profiles $\mathcal{P}$ and $\mathcal{P}'$ respectively. If $\mathcal{R}^*_{\mathcal{P}} \cap \mathcal{R}^*_{\mathcal{P}'} \neq \emptyset$ and $\mathcal{P}''$ is the profile obtained by concatenating $\mathcal{P}$ and $\mathcal{P}'$, then $\mathcal{R}^*_{\mathcal{P}''} = \mathcal{R}^*_{\mathcal{P}} \cap \mathcal{R}^*_{\mathcal{P}'}$.

Besides, the $k$-wise Kemeny rule does not satisfy *Independence of irrelevant alternatives*, i.e., the relative positions of two candidates in a consensus ranking can depend on the presence of other candidates. Let us illustrate this point with the following example.

**Example 4.** *Considering the preference profile from Example 1, the only consensus ranking for $\delta^3_{\mathtt{KT}}$ is $c_1 \succ c_2 \succ c_3$. Yet, without $c_3$ the only consensus ranking would be $c_2 \succ c_1$.*

Lastly, note that there exists a noise model such that the $k$-wise Kemeny rule can be interpreted as a maximum likelihood estimator (Conitzer, Rognlie, and Xia 2009). In this view of voting, one assumes that there exists a "correct" ranking $r$, and each vote corresponds to a noisy perception of this correct ranking. Consider the conditional probability measure $\mathbb{P}$ on $\mathcal{R}(C)$ defined by $\mathbb{P}(r'|r) \propto e^{-\delta^k_{\mathtt{KT}}(r,r')}$. It is easy to convince oneself that the $k$-wise Kemeny rule returns a ranking $r^*$ that maximizes $\mathbb{P}(\mathcal{P}\,|r^*) = \prod_{r' \in \mathcal{P}} \mathbb{P}(r'|r^*)$ and is thus a maximum likelihood estimate of $r$.

## Computational Complexity of $k$-KAP

We now turn to the algorithmic study of $k$-KAP. After providing a hardness result, we will design an efficient Fixed Parameter Tractable (FPT) algorithm for parameter $m$.

While $k$-KAP is obviously NP-hard for $k = 2$ as it then corresponds to determining a consensus ranking w.r.t. the Kemeny rule, we strengthen this result by showing that it is also NP-hard for *any* constant value $k \geq 3$. The proof, uses a reduction from 2-KAP.

**Theorem 1.** *For any constant $k \geq 3$, $k$-KAP is NP-hard, even if the number of voters equals 4 or if the average range of candidates equals 2 (where the range of a candidate $c$ is defined by $\max_{r \in \mathcal{P}} \mathtt{rk}(c, r) - \min_{r \in \mathcal{P}} \mathtt{rk}(c, r) + 1$ and the average is taken over all candidates).*

Despite this result, $k$-KAP is obviously FPT w.r.t. the number $m$ of candidates, by simply trying the $m!$ rankings in $\mathcal{R}(C)$. We now design a dynamic programming procedure which significantly improves this time complexity.

**Proposition 1.** *If $r^*$ is an optimal ranking for $k$-KAP, then $\delta^k_{\mathtt{KT}}(r^*, \mathcal{P}) = d^k_{\mathtt{KT}}(C)$, where, for any subset $S \subseteq C$, $d^k_{\mathtt{KT}}(S)$ is defined by the recursive relation:*

$$d^k_{\mathtt{KT}}(S) = \min_{c \in S}[d^k_{\mathtt{KT}}(S \setminus \{c\}) + \sum_{r \in \mathcal{P}_S} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S| - \mathtt{rk}(c', r) - 1}{i}] \quad (2)$$

$$d^k_{\mathtt{KT}}(\emptyset) = 0.$$

*Proof.* Given $S \subseteq C$ and $c \in S$, let us define $\mathcal{R}_c(S)$ as $\{r \in \mathcal{R}(S) \text{ s.t. } t_r(S) = c\}$. The set $\Delta^k(S)$ can be partitioned into $\Delta^k_c(S) = \{S' \subseteq \Delta^k(S) \text{ s.t. } c \in S'\}$ and $\Delta^k_{\bar{c}}(S) = \{S' \subseteq \Delta^k(S) \text{ s.t. } c \notin S'\} = \Delta^k(S \setminus \{c\})$. Given a preference profile $\mathcal{P}$ over $C$ and a ranking $\hat{r} \in \mathcal{R}_c(S)$, the summation defining $\delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_S)$ breaks down as follows:

$$\delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_S) = \sum_{r \in \mathcal{P}_S} \sum_{S' \in \Delta^k(S)} \mathbb{1}_{t_{\hat{r}}(S') \neq t_r(S')}$$

$$= \delta^k_{\mathtt{KT}}(\hat{r}_{S \setminus \{c\}}, \mathcal{P}_{S \setminus \{c\}}) + \sum_{r \in \mathcal{P}_S} \sum_{S' \in \Delta^k_c(S)} \mathbb{1}_{t_{\hat{r}}(S') \neq t_r(S')}. \quad (3)$$

Using the same reasoning as in Equation 1 on page 3, the second summand in Equation 3 can be rewritten as follows:

$$\sum_{r \in \mathcal{P}_S} \sum_{c' \in S} \mathbb{1}_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|B_c(\hat{r}) \cap B_{c'}(r)|}{i}$$

because $t_{\hat{r}}(S') = c$ for all $S' \in \Delta^k_c(S)$. Note that $B_c(\hat{r}) = S \setminus \{c\}$ and $B_{c'}(r) = \{c'' \in S \text{ s.t. } c' \succ_r c''\} \subseteq S$, thus $|B_c(\hat{r}) \cap B_{c'}(r)| = |S| - \mathtt{rk}(c', r) - 1$. Hence, $\delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_S)$ is equal to:

$$\delta^k_{\mathtt{KT}}(\hat{r}_{S \setminus \{c\}}, \mathcal{P}_{S \setminus \{c\}}) + \sum_{r \in \mathcal{P}_S} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S| - \mathtt{rk}(c', r) - 1}{i} \quad (4)$$

Consider now a ranking $r^* \in \mathcal{R}(S)$ such that $\delta^k_{\mathtt{KT}}(r^*, \mathcal{P}_S) = \min_{r \in \mathcal{R}(S)} \delta^k_{\mathtt{KT}}(r, \mathcal{P}_S)$. We have:

$$\delta^k_{\mathtt{KT}}(r^*, \mathcal{P}_S) = \min_{c \in S} \min_{\hat{r} \in \mathcal{R}_c(S)} \delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_S)$$

$$= \min_{c \in S} \Big( (\min_{\hat{r} \in \mathcal{R}(S \setminus \{c\})} \delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_{S \setminus \{c\}})) + \sum_{r \in \mathcal{P}_S} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S| - \mathtt{rk}(c', r) - 1}{i} \Big)$$

because the second summand in Equation 4 does not depend on $\hat{r}$ (it only depends on $c$, which is the argument of the first min operator). If one denotes $\min_{r \in \mathcal{R}(S)} \delta^k_{\mathtt{KT}}(r, \mathcal{P}_S)$ by $d^k_{\mathtt{KT}}(S)$, one obtains Equation 2. This concludes the proof. □

A candidate $c \in S$ that realizes the minimum in Equation 2 can be ranked in first position in an optimal ranking for $\mathcal{P}_S$. Once $d^k_{\mathtt{KT}}(S)$ is computed for each $S \subseteq C$, a ranking $r^*$ achieving the optimal value $d^k_{\mathtt{KT}}(C)$ can thus be determined recursively starting from $S = C$. The complexity of the induced dynamic programming method is $O(2^m m^2 n)$

as there are $2^m$ subsets $S \subseteq C$ to consider and each value $d^k_{\mathtt{KT}}(S)$ is computed in $O(m^2n)$ by Equation 2. The min operation is indeed performed on $m$ values and the sum $\sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S|-\mathtt{rk}(c',r)-1}{i}$ is computed incrementally in $O(m)$, which entails an $O(mn)$ complexity for the second summand in Equation 2 (the $n$ factor is due to the sum over all $r \in \mathcal{P}_S$). The computation of binomial coefficients $\binom{p}{i}$ for $i \in \{0, \ldots, k-2\}$ and $p \in \{i, \ldots, m-2\}$ is performed in $O(mk)$ in a preliminary step thanks to Pascal's formula.

## 4 The $k$-Wise Majority Digraph

We now propose and investigate a $k$-wise counterpart of the pairwise majority digraph, that will be used in a preprocessing procedure for $k$-KAP.

As stated in the introduction, the pairwise Kemeny rule is strongly related to the *pairwise majority digraph*. We denote by $\mathcal{G}_\mathcal{P}$ the pairwise majority digraph associated to profile $\mathcal{P}$. We recall that in this digraph, there is one vertex per candidate, and there is an arc from candidate $c$ to candidate $c'$ if a strict majority of voters prefers $c$ to $c'$. In the weighted pairwise majority digraph, each arc $(c, c')$ is weighted by $w_\mathcal{P}(c, c') := |\{r \in \mathcal{P} \text{ s.t. } c \succ_r c'\}| - |\{r \in \mathcal{P} \text{ s.t. } c' \succ_r c\}|$.

**Example 5.** *Consider a profile $\mathcal{P}$ with 10 voters and 6 candidates such that:*
*– 4 voters have preferences $c_1 \succ c_2 \succ c_4 \succ c_3 \succ c_5 \succ c_6$;*
*– 4 voters have preferences $c_1 \succ c_3 \succ c_2 \succ c_4 \succ c_5 \succ c_6$;*
*– 1 voter has preferences $c_6 \succ c_1 \succ c_2 \succ c_4 \succ c_3 \succ c_5$;*
*– 1 voter has preferences $c_6 \succ c_1 \succ c_4 \succ c_3 \succ c_2 \succ c_5$.*
*The pairwise majority digraph $\mathcal{G}_\mathcal{P}$ is on the left of Figure 1.*

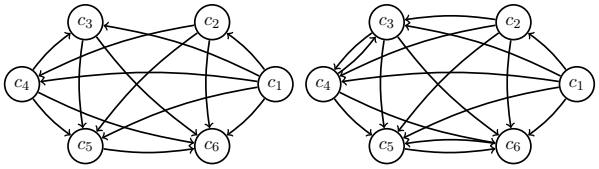



Figure 1: $k$-wise majority digraph in Example 5 for $k = 2$ (left) and $k = 3$ (right).

From $\mathcal{G}_\mathcal{P}$, we can define a set of *consistent* rankings:

**Definition 1.** *Let $\mathcal{G}$ be a digraph whose vertices correspond to the candidates in $C$. Let $B_1(\mathcal{G}), \ldots, B_{t(\mathcal{G})}(\mathcal{G})$ be the subsets of $C$ corresponding to the Strongly Connected Components (SCCs) of $\mathcal{G}$, and $\mathcal{O}(\mathcal{G})$ denote the set of linear orders $<_\mathcal{G}$ on $\{1, \ldots, t(\mathcal{G})\}$ such that if there exists an arc $(c, c')$ from $c \in B_i(\mathcal{G})$ to $c' \in B_j(\mathcal{G})$ then $i <_\mathcal{G} j$. Given $<_\mathcal{G} \in \mathcal{O}(\mathcal{G})$, we say that a ranking $r$ is* consistent *with $<_\mathcal{G}$ if the candidates in $B_i$ are ranked before the ones of $B_j$ when $i <_\mathcal{G} j$.*

The following result states that, for any $<_{\mathcal{G}_\mathcal{P}} \in \mathcal{O}(\mathcal{G}_\mathcal{P})$, there exists a consensus ranking for $\delta_{\mathtt{KT}}$ among the rankings consistent with $<_{\mathcal{G}_\mathcal{P}}$.

**Theorem 2** (Theorem 16 in reference Charon and Hudry, 2010)**.** *Let $\mathcal{P}$ be a profile over $C$ and assume that the SCCs of $\mathcal{G}_\mathcal{P}$ are numbered according to a linear order $<_{\mathcal{G}_\mathcal{P}} \in \mathcal{O}(\mathcal{G}_\mathcal{P})$. Consider the ranking $r^*$, consistent with $<_{\mathcal{G}_\mathcal{P}}$, obtained by the concatenation of rankings $r^*_1, \ldots, r^*_{t(\mathcal{G}_\mathcal{P})}$ where $\delta_{\mathtt{KT}}(r^*_i, \mathcal{P}_{B_i(\mathcal{G}_\mathcal{P})}) = \min_{r \in \mathcal{R}(B_i(\mathcal{G}_\mathcal{P}))} \delta_{\mathtt{KT}}(r, \mathcal{P}_{B_i(\mathcal{G}_\mathcal{P})})$. We have:*

$$\delta_{\mathtt{KT}}(r^*, \mathcal{P}) = \min_{r \in \mathcal{R}(C)} \delta_{\mathtt{KT}}(r, \mathcal{P})$$

*That is, $r^*$ is a consensus ranking according to the Kemeny rule. Furthermore, if $\mathcal{O}(\mathcal{G}_\mathcal{P}) = \{<_{\mathcal{G}_\mathcal{P}}\}$ and $w_\mathcal{P}(c, c') > 0$ for all $c \in B_i(\mathcal{G}_\mathcal{P})$ and $c' \in B_j(\mathcal{G}_\mathcal{P})$ when $i <_{\mathcal{G}_\mathcal{P}} j$, then all consensus rankings are consistent with $<_{\mathcal{G}_\mathcal{P}}$.*

This result does not hold anymore if one uses $\delta^k_{\mathtt{KT}}$ (with $k \geq 3$) instead of $\delta_{\mathtt{KT}}$, as shown by the following example.

**Example 6.** *Let us denote by $\mathcal{P}$ the profile of Example 1. The pairwise majority digraph $\mathcal{G}_\mathcal{P}$ has three SCCs $B_1(\mathcal{G}_\mathcal{P}) = \{c_2\}$, $B_2(\mathcal{G}_\mathcal{P}) = \{c_3\}$ and $B_3(\mathcal{G}_\mathcal{P}) = \{c_1\}$. In this example, $\mathcal{O}(\mathcal{G}_\mathcal{P}) = \{<_{\mathcal{G}_\mathcal{P}}\}$ where $1 <_{\mathcal{G}_\mathcal{P}} 2 <_{\mathcal{G}_\mathcal{P}} 3$. The only ranking consistent with $<_{\mathcal{G}_\mathcal{P}}$ is $c_2 \succ c_3 \succ c_1$ while the only consensus ranking w.r.t. the 3-wise Kemeny rule is $c_1 \succ c_2 \succ c_3$.*

In order to adapt Theorem 2 to the $k$-wise Kemeny rule, we now introduce the concept of *$k$-wise majority digraph*. Let $\Delta^k_{cc'}(S) = \{S' \in \Delta^k(S) \text{ s.t. } \{c, c'\} \subseteq S'\}$. If $S$ is not specified, it is assumed to be $C$. Given a ranking $r$, we denote by $\Delta^k_r(S, c, c')$ the set $\{S' \in \Delta^k_{cc'}(S) \text{ s.t. } t_r(S') = c\}$. Given a profile $\mathcal{P}$, we denote by $\phi^k_\mathcal{P}(S, c, c')$ the value $\sum_{r \in \mathcal{P}} |\Delta^k_r(S, c, c')|$ and by $w^k_\mathcal{P}(S, c, c')$ the difference $\phi^k_\mathcal{P}(S, c, c') - \phi^k_\mathcal{P}(S, c', c)$. This definition implies that $w^k_\mathcal{P}(S, c', c) = -w^k_\mathcal{P}(S, c, c')$. The value $w^k_\mathcal{P}(S, c, c')$ is the net agreement loss that would be incurred by swapping $c$ and $c'$ in a feasible solution $r$ of $k$-KAP where $\mathtt{rk}(c', r) = \mathtt{rk}(c, r) + 1$ and $S = B_{c'}(r) \cup \{c, c'\}$. If $\max_{S \in \Delta^m_{cc'}} w^k_\mathcal{P}(S, c, c') \geq 0$ (resp. $\min_{S \in \Delta^m_{cc'}} w^k_\mathcal{P}(S, c, c') > 0$) then, in a consensus ranking $r$ for $\delta^k_{\mathtt{KT}}$ where $c$ and $c'$ would be consecutive, it is possible (resp. necessary) that $c \succ_r c'$.

The $k$-wise majority digraph associated to a profile $\mathcal{P}$ over a set $C$ of candidates is the weighted digraph $\mathcal{G}^k_\mathcal{P} = (\mathcal{V}, \mathcal{A})$, where $\mathcal{V} = C$ and $(c, c') \in \mathcal{A}$ iff:

$$\exists S \in \Delta^m_{cc'} \text{ s.t. } w^k_\mathcal{P}(S, c, c') > 0.$$

The weight $w^k_\mathcal{P}(c, c')$ of this arc is then given by:

$$w^k_\mathcal{P}(c, c') := \max_{S \in \Delta^m_{cc'}} w^k_\mathcal{P}(S, c, c').$$

Note that, if $k \geq 3$, we may obtain edges $(c, c')$ and $(c', c)$ both with strictly positive weights (which is impossible in the pairwise majority digraph). For instance, for the profile $\mathcal{P}$ of Example 5, $w^k_\mathcal{P}(c_3, c_4) = w^k_\mathcal{P}(\{c_2, c_3, c_4\}, c_3, c_4) = 1$ and $w^k_\mathcal{P}(c_4, c_3) = w^k_\mathcal{P}(\{c_3, c_4, c_5\}, c_4, c_3) = 4$. The digraph $\mathcal{G}^3_\mathcal{P}$ is shown on the right of Figure 1. Besides, for any $\mathcal{P}$, $\mathcal{G}^2_\mathcal{P}$ is the pairwise majority digraph as $\Delta^2_{cc'}(S) = \{\{c, c'\}\}$ $\forall S \in \Delta^m_{cc'}$. Theorem 2 adapts as follows for an arbitrary $k$:

**Theorem 3.** *Let $\mathcal{P}$ be a profile over $C$ and assume that the SCCs of $\mathcal{G}^k_\mathcal{P}$ are numbered according to a linear order $<_{\mathcal{G}^k_\mathcal{P}} \in \mathcal{O}(\mathcal{G}^k_\mathcal{P})$. Among the rankings consistent with $<_{\mathcal{G}^k_\mathcal{P}}$, there exists a consensus ranking w.r.t. the $k$-wise Kemeny rule. Besides, if $\mathcal{O}(\mathcal{G}^k_\mathcal{P}) = \{<_{\mathcal{G}^k_\mathcal{P}}\}$ and $\min_{S \in \Delta^m_{cc'}} w^k_\mathcal{P}(S, c, c') > 0$[2] for all $c \in B_i(\mathcal{G}^k_\mathcal{P})$ and $c' \in B_j(\mathcal{G}^k_\mathcal{P})$ when $i <_{\mathcal{G}^k_\mathcal{P}} j$, then all consensus rankings are consistent with $<_{\mathcal{G}^k_\mathcal{P}}$.*

[2] Or, equivalently, $\max_{S \in \Delta^m_{cc'}} w^k_\mathcal{P}(S, c', c) < 0$.

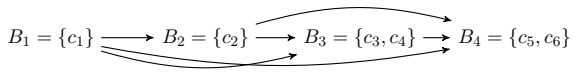



Figure 2: The meta-graph of SCCs of $\mathcal{G}^3_{\mathcal{P}}$ in Example 5.

**Example 7.** *The meta-graph of SCCs of $\mathcal{G}^3_{\mathcal{P}}$ in Example 5 is represented in Figure 2. The above result implies that there exists a consensus ranking among $c_1 \succ c_2 \succ c_3 \succ c_4 \succ c_5 \succ c_6$, $c_1 \succ c_2 \succ c_3 \succ c_4 \succ c_6 \succ c_5$, $c_1 \succ c_2 \succ c_4 \succ c_3 \succ c_5 \succ c_6$ and $c_1 \succ c_2 \succ c_4 \succ c_3 \succ c_6 \succ c_5$.*

To take advantage of Theorem 3, one could try 1) to index the SCCs of $\mathcal{G}^k_{\mathcal{P}}$ according to a linear order $<_{\mathcal{G}^k_{\mathcal{P}}} \in \mathcal{O}(\mathcal{G}^k_{\mathcal{P}})$, and then 2) to work on each SCC separately, before concatenating the obtained rankings. However, for a consensus ranking consistent with $<_{\mathcal{G}^k_{\mathcal{P}}}$, the relative positions of candidates in $B_i(\mathcal{G}^k_{\mathcal{P}})$ depend on the set of candidates in $B_{>i}(\mathcal{G}^k_{\mathcal{P}}) := B_{i+1}(\mathcal{G}^k_{\mathcal{P}}) \cup \ldots \cup B_{t(\mathcal{G}^k_{\mathcal{P}})}(\mathcal{G}^k_{\mathcal{P}})$ (but not on their order). The influence of $B_{>i}(\mathcal{G}^k_{\mathcal{P}})$ can be captured in the dynamic programming procedure by applying a modified version of Equation 2 separately for each subset $B_{t(\mathcal{G}^k_{\mathcal{P}})}(\mathcal{G}^k_{\mathcal{P}})$ downto $B_1(\mathcal{G}^k_{\mathcal{P}})$. Formally, if $r^*$ is optimal for $k$-KAP, then:

$$\delta^k_{\mathtt{KT}}(r^*, \mathcal{P}) = \sum_{i=1}^{t(\mathcal{G}^k_{\mathcal{P}})} d^k_{\mathtt{KT}}(B_i(\mathcal{G}^k_{\mathcal{P}}))$$

where, for any subset $S \subseteq B_i(\mathcal{G}^k_{\mathcal{P}})$, $d^k_{\mathtt{KT}}(S)$ is defined by $d^k_{\mathtt{KT}}(\emptyset) = 0$ and ($B_{>i}$ stands for $B_{>i}(\mathcal{G}^k_{\mathcal{P}})$):

$$d^k_{\mathtt{KT}}(S) = \min_{c \in S} [d^k_{\mathtt{KT}}(S \backslash \{c\}) + \sum_{r \in \mathcal{P}_{S \cup B_{>i}}} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S| + |B_{>i}| - \mathtt{rk}(c', r) - 1}{i}]$$

It amounts to replacing $S$ by $S \cup B_{>i}$ in the second summand of Equation 2 to take into account the existence of a consensus ranking where all the candidates of $B_{>i}$ are ranked after those of $B_i$. Let $r^*_i$ be a ranking of $B_i(\mathcal{G}^k_{\mathcal{P}})$ such that $\delta^k_{\mathtt{KT}}(r^*_{\geq i}, \mathcal{P}_{B_{\geq i}(\mathcal{G}^k_{\mathcal{P}})}) = d^k_{\mathtt{KT}}(B_i(\mathcal{G}^k_{\mathcal{P}})) + \ldots + d^k_{\mathtt{KT}}(B_{t(\mathcal{G}^k)}(\mathcal{G}^k_{\mathcal{P}}))$, where $r^*_{\geq i}$ is the ranking obtained by the concatenation of rankings $r^*_i, \ldots, r^*_{t(\mathcal{G}^k)}$ in this order. The ranking $r^*_{\geq 1}$ of $C$ is a consensus ranking w.r.t. the $k$-wise Kemeny rule. Given Theorem 3, the $k$-wise majority digraph thus seems promising to boost the computation of a consensus ranking. Unfortunately, the following negative result holds.

**Theorem 4.** *Given a profile $\mathcal{P}$ and two candidates $c$ and $c'$, determining if $\max_{S \in \Delta^m_{cc'}} w^k_{\mathcal{P}}(S, c, c') > 0$ is NP-hard for $k \geq 4$.*

Hence, computing $\mathcal{G}^k_{\mathcal{P}}$ from $\mathcal{P}$ is NP-hard for $k \geq 4$. In contrast, $\mathcal{G}^3_{\mathcal{P}}$ can be computed in polynomial time. Indeed, given a set $S \subset C$ such that $\{c, c'\} \subseteq S$, adding to $S$ an element $x \notin S$ increases $\phi^3_{\mathcal{P}}(S, c, c')$ by one for each $r \in \mathcal{P}$ such that $c \succ_r c'$ and $c \succ_r x$. Let $\mathcal{P}_{c \succ c'} := \{r \in \mathcal{P} \text{ s.t. } c \succ_r c'\}$. A set $S^*$ maximizing $w^3_{\mathcal{P}}(S, c, c')$ is $S^* := \{c, c'\} \cup \{x \in C \text{ s.t. } |\mathcal{P}_{c \succ c'} \cap \mathcal{P}_{c \succ x}| > |\mathcal{P}_{c' \succ c} \cap \mathcal{P}_{c' \succ x}|\}$.

Note that one can take advantage of the meta-graph of SCCs to trim the graph $\mathcal{G}^k_{\mathcal{P}}$ if one looks for a consensus ranking $r^*$ consistent with a specific order $<_{\mathcal{G}^k_{\mathcal{P}}} \in \mathcal{O}(\mathcal{G}^k_{\mathcal{P}})$. It may indeed happen that, for an edge $(c, c')$, the weight $w^k_{\mathcal{P}}(c, c') = w^k_{\mathcal{P}}(S, c, c') > 0$ corresponds to a set $S$ which contains candidates that will never be below $c$ in $r^*$. Conversely, the set $S$ may omit candidates that are necessarily below $c$ in $r^*$. These constraints can be induced by either unanimity dominance relations or by $<_{\mathcal{G}^k_{\mathcal{P}}}$. The following example illustrates this idea.

**Example 8.** *Let us refine the digraph $\mathcal{G}^3_{\mathcal{P}}$ previously obtained for the profile $\mathcal{P}$ of Example 5. The SCCs are $B_1 = \{c_1\}$, $B_2 = \{c_2\}$, $B_3 = \{c_3, c_4\}$ and $B_4 = \{c_5, c_6\}$ and $\mathcal{O}(\mathcal{G}^k_{\mathcal{P}}) = \{<_{\mathcal{G}^k_{\mathcal{P}}}\}$, where $1 <_{\mathcal{G}^k_{\mathcal{P}}} 2 <_{\mathcal{G}^k_{\mathcal{P}}} 3 <_{\mathcal{G}^k_{\mathcal{P}}} 4$. A set maximizing $w^3_{\mathcal{P}}(S, c_3, c_4)$ is $S = \{c_2, c_3, c_4\}$. This set contains $c_2$ while it is necessarily above $c_3$ in a consistent ranking. Conversely, candidates $c_5$ and $c_6$ are omitted while they are necessarily below $c_3$. By taking into account these constraints, we obtain that a set maximizing $w^3_{\mathcal{P}}(S, c_3, c_4)$ is $S = \{c_3, c_4, c_5, c_6\}$, for which $w^3_{\mathcal{P}}(S, c_3, c_4) = -4$. Hence, we can remove the arc $(c_3, c_4)$ from $\mathcal{G}^3_{\mathcal{P}}$. Similarly, it is possible to show that the arc $(c_6, c_5)$ can be removed from $\mathcal{G}^3_{\mathcal{P}}$. Thanks to these refinement steps, we can conclude that a consensus ranking is $r^* = c_1 \succ c_2 \succ c_4 \succ c_3 \succ c_5 \succ c_6$.*

## 5 Numerical Tests

Our numerical tests[3] have three objectives: we evaluate the computational performance of the dynamic programming approach of Section 3, we evaluate the impact of parameter $k$ on the set of consensus rankings, and we assess the efficiency of the preprocessing technique of Section 4.

**Generation of preference profiles.** The preference profiles are generated according to the Mallows model (Mallows 1957), using the Python package PrefLib-Tools (Mattei and Walsh 2013). This model takes two parameters as input: a reference ranking $\sigma$ (the mode of the distribution) and a dispersion parameter $\phi \in (0, 1)$. Given these inputs, the probability of generating a ranking $r$ is proportional to $\phi^{\delta_{\mathtt{KT}}(r,\sigma)}$. The more $\phi$ tends towards 0 (resp. 1), the more the preference rankings become correlated and resemble $\sigma$ (resp. become equally probable, i.e., we are close to the *impartial culture assumption*). This model enables us to measure in a simple way how the level of correlation in the input rankings impacts our results. In all tests, the number $n$ of voters is set to 50 and the ranking $\sigma$ is set arbitrarily as the $k$-wise Kemeny rule is neutral. For each triple $(m, k, \phi)$ considered, the results are averaged over 50 preference profiles.

**Practicability of the dynamic programming approach.** We first evaluate our dynamic programming approach on instances with different values for $m$ and $k$. Note that the computational performance measured here is not impacted by the level of correlation in the input rankings as it does

[3] Implementation in C++. All times are CPU seconds on an Intel Core I7-8700 3.20 GHz processor with 16GB of RAM.

Table 2: Average, max and min wall-clock times in seconds of the dynamic programming approach of Section 3 for varying values of $m$ and $k$ (Rows 3 to 5). Average number of consensus rankings for increasing values of $m$ and $k$ (Row 6).

| $m$ | 6 | | | 10 | | | 14 | | | 18 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $k$ | 2 | 3 | 6 | 2 | 5 | 10 | 2 | 7 | 14 | 2 | 9 | 18 |
| Average time | <0.01 | <0.01 | <0.01 | 0.07 | 0.08 | 0.08 | 2.52 | 2.54 | 2.61 | 70.93 | 72.26 | 74.95 |
| Max time | <0.01 | <0,01 | <0,01 | 0.8 | 0.08 | 0.09 | 2.64 | 2.60 | 2.64 | 71.57 | 73.57 | 75.38 |
| Min time | <0,01 | <0,01 | <0,01 | 0.7 | 0.07 | 0.08 | 2.49 | 2.49 | 2.57 | 70.27 | 71.91 | 74.33 |
| $\lvert\mathcal{R}^*\rvert_{\texttt{avg}}$ | 3.00 | 1.20 | 1.05 | 3.84 | 1.24 | 1.10 | 5.36 | 2.36 | 1.16 | 19.7 | 4.12 | 1.47 |

not change the number of states in dynamic programming nor the computation time to determine the optimal value in each state. Hence, we only consider instances generated under the impartial culture assumption, i.e., with $\phi \approx 1$. Table 2 (Rows 3-5) displays the average, max and min running times obtained for some representative $(m, k)$ values. As expected, the running times increase exponentially with $m$. Conversely, parameter $k$ seems to have a moderate impact on the running times. The dynamic programming approach enables us to solve $k$-KAP in a time of up to 3 sec. (resp. 76 sec.) for $m \leq 14$ (resp. $m \leq 18$).

**Influence of $k$ on the set of consensus rankings.** Second, we study the impact of $k$ on the set of optimal solutions to $k$-KAP. Indeed, one criticism for the Kemeny rule is that there exists instances for which the set of consensus rankings is compounded of many solutions which are quite different from one another. Thus, we investigate if increasing $k$ helps in mitigating this issue. For this purpose, we consider the same instances as before and compute the average number of consensus rankings denoted by $|\mathcal{R}^*|_{\texttt{avg}}$. The results are displayed in the sixth row of Table 2. Interestingly, this measure decreases quickly with $k$. For instance, when $m = 18$, $|\mathcal{R}^*|_{\texttt{avg}}$ is divided by 5 when $k$ increases from 2 to 9 and is below 2 when $k = m$. The intuition is that $\delta^k_{\text{KT}}$ becomes more fine-grained as $k$ increases.

**Impact of the 3-wise majority graph.** Lastly, we study the impact of the preprocessing method proposed in Section 4 for $k = 3$. This preprocessing uses the $k$-wise majority digraph to divide $k$-KAP into several subproblems which can be solved separately by dynamic programming. Hopefully, when voters' preferences are correlated (i.e., for "small" $\phi$ values), these subproblems become smaller and more numerous, making the preprocessing more efficient. The results are shown in Table 3, where the results obtained without preprocessing are also given in the last column. The obtained running times are highly dependent on $\phi$. For instance, with $m = 18$, the average running time for solving 3-KAP is above 1 minute if $\phi = 0.95$ while it is below 1 second if $\phi \leq 0.85$. This gap is necessarily related to the preprocessing step, since $\phi$ has no impact on the running time of the dynamic programming approach. To explain this significant speed-up, we display in Table 4 the average size of the largest SCC of the 3-wise majority digraph at the end of the preprocessing step. Unsurprisingly, this average size turns out to be correlated with $\phi$: when $\phi \leq 0.5$, the size of the largest SCC is almost always 1. Hence, the preprocessing step is likely to yield directly a consensus ranking. In contrast, when $\phi = 0.95$, the average size of the largest SCC is close to $m$, thus the impact of the preprocessing is low.

Table 3: Average, max and min wall-clock times (in seconds) for the 3-wise Kemeny rule with preprocessing.

| $m$ | $\phi$ | 0.5 | 0.8 | 0.85 | 0.9 | 0.95 | w/o preproc. |
|---|---|---|---|---|---|---|---|
| 6 | Avg time | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 |
| | Max time | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 |
| | Min time | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 |
| 10 | Avg time | 0.03 | 0.03 | 0.04 | 0.07 | 0.10 | 0.07 |
| | Max time | 0.03 | 0.03 | 0.10 | 0.15 | 0.17 | 0.08 |
| | Min time | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.07 |
| 14 | Avg time | 0.09 | 0.09 | 0.11 | 0.94 | 2.21 | 2.52 |
| | Max time | 0.09 | 0.13 | 0.25 | 3.14 | 3.26 | 2.59 |
| | Min time | 0.08 | 0.08 | 0.09 | 0.10 | 0.26 | 2.49 |
| 18 | Avg time | 0.20 | 0.20 | 0.55 | 14.87 | 61.72 | 71.17 |
| | Max time | 0.31 | 0.21 | 8.46 | 79.87 | 80.11 | 71.61 |
| | Min time | 0.19 | 0.19 | 0.19 | 0.22 | 6.02 | 71.02 |

Table 4: Average size of the largest SCC after preprocessing.

| $m\backslash\phi$ | 0.47 | 0.81 | 0.85 | 0.88 | 0.95 |
|---|---|---|---|---|---|
| 6 | <1.1 | 1.84 | 1.88 | 2.72 | 3.28 |
| 10 | <1.1 | 1.64 | 3.28 | 5.32 | 8.20 |
| 14 | <1.1 | 2.68 | 3.84 | 9.12 | 12.91 |
| 18 | <1.1 | 2.84 | 4.27 | 9.80 | 17.44 |

## 6 Conclusion

In this paper, we advocate using the results of *setwise* contests between candidates to design social welfare functions that are less myopic than those only based on pairwise comparisons. In this direction, we have studied a $k$-wise generalization of the Kemeny rule, and established that determining a consensus ranking is NP-hard for any $k \geq 3$. After proposing a dynamic programming procedure, we have investigated a $k$-wise variant of the majority graph, from which we developed a preprocessing step. Computing this graph is a polynomial time problem for $k = 3$ but becomes NP-hard for $k \geq 4$. The numerical tests show the practicability of the approach for up to 18 candidates. A natural research direction is to investigate the complexity of determining a consensus ranking for $\delta^k_{\text{KT}}$ when $k = m$, because our hardness result only holds for *fixed* values of $k$. Another avenue to explore is to propose alternative definitions of $k$-wise majority graphs that are easier to compute for $k > 3$.

Finally, other social welfare functions based on the results of setwise contests are worth investigating in our opinion, both from the axiomatic and the computational points of view.

## 7 Acknowledgments

This work has been partially supported by the Italian MIUR PRIN 2017 Project ALGADIMAR "Algorithms, Games, and Digital Markets".

## References

Altman, A., and Tennenholtz, M. 2008. Axiomatic foundations for ranking systems. *Journal of Artificial Intelligence Research* 31:473–495.

Arrow, K. J. 1950. A difficulty in the concept of social welfare. *Journal of political economy* 58(4):328–346.

Baldiga, K. A., and Green, J. R. 2013. Assent-maximizing social choice. *Social Choice and Welfare* 40(2):439–460.

Brandt, F.; Conitzer, V.; Endriss, U.; Lang, J.; and Procaccia, A. D. 2016. *Handbook of computational social choice*. Cambridge University Press.

Charon, I., and Hudry, O. 2010. An updated survey on the linear ordering problem for weighted or unweighted tournaments. *Annals of Operations Research* 175(1):107–158.

Cheng, W., and Hüllermeier, E. 2009. A new instance-based label ranking approach using the mallows model. In *Proceedings of the 6th International Symposium on Neural Networks, ISNN 2009, Wuhan, China, May 26-29, 2009*, 707–716.

Clémençon, S.; Korba, A.; and Sibony, E. 2018. Ranking median regression: Learning to order through local consensus. In *Proceedings of the 29th international conference on Algorithmic Learning Theory, ALT 2018, Lanzarote, Canary Islands, Spain, April 7-9, 2018*, 212–245.

Conitzer, V.; Rognlie, M.; and Xia, L. 2009. Preference functions that score rankings and maximum likelihood estimation. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, July 11-17, 2009*, 109–115.

Dwork, C.; Kumar, R.; Naor, M.; and Sivakumar, D. 2001. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web, WWW 2001, Hong Kong, China, May 1-5, 2001*, 613–622.

Fagin, R.; Kumar, R.; and Sivakumar, D. 2003. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data, San Diego, California, USA, June 9-12, 2003*, 301–312.

Fishburn, P. C. 1977. Condorcet social choice functions. *SIAM Journal on applied Mathematics* 33(3):469–489.

Gilbert, H.; Portoleau, T.; and Spanjaard, O. 2019. Beyond pairwise comparisons in social choice: A setwise kemeny aggregation problem. *CoRR* abs/1911.06226.

Jackson, B. N.; Schnable, P. S.; and Aluru, S. 2008. Consensus genetic maps as median orders from inconsistent sources. *IEEE/ACM Transactions on computational biology and bioinformatics* 5(2):161–171.

Kemeny, J. G. 1959. Mathematics without numbers. *Daedalus* 88(4):577–591.

Kumar, R., and Vassilvitskii, S. 2010. Generalized distances between rankings. In *Proceedings of the 19th international conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, 571–580.

Lu, T., and Boutilier, C. 2010. The unavailable candidate model: a decision-theoretic view of social choice. In *Proceedings of the 11th ACM conference on Electronic Commerce, EC 2010, Cambridge, Massachusetts, USA, June 7-11, 2010*, 263–274.

Mallows, C. L. 1957. Non-null ranking models. I. *Biometrika* 44(1/2):114–130.

Mattei, N., and Walsh, T. 2013. Preflib: A library of preference data HTTP://PREFLIB.ORG. In *Proceedings of the 3rd international conference on Algorithmic Decision Theory, ADT 2013, Bruxelles, Belgium, November 13-15, 2013*, 259–270.

Moulin, H. 1991. *Axioms of cooperative decision making*. Number 15 in Econometric Society Monographs. Cambridge University Press.

Wang, S.; Sun, J.; Gao, B. J.; and Ma, J. 2014. Vsrank: A novel framework for ranking-based collaborative filtering. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5(3):51.

Zwicker, W. S. 2018. Cycles and intractability in a large class of aggregation rules. *Journal of Artificial Intelligence Research* 61:407–431.

# On How Turing *and Singleton Arc Consistency* Broke the Enigma Code

**Valentin Antuori**
Renault, LAAS-CNRS, Université de Toulouse, CNRS, France

**Tom Portoleau**
LAAS-CNRS, Université de Toulouse, CNRS, France

**Louis Rivière**
LAAS-CNRS, Université de Toulouse, CNRS, ANITI, France

**Emmanuel Hebrard**
LAAS-CNRS, Université de Toulouse, CNRS, ANITI, France

## Abstract

In this paper, we highlight an intriguing connection between the cryptographic attacks on Enigma's code and local consistency reasoning in constraint programming.

The coding challenge proposed to the students during the 2020 ACP summer school, to be solved by constraint programming, was to decipher a message encoded using the well known Enigma machine, with as only clue a tiny portion of the original message. A number of students quickly crafted a model, thus nicely showcasing CP technology – as well as their own brightness. The detail that is slightly less favorable to CP technology is that solving this model on modern hardware is challenging, whereas the "Bombe", an antique computing device, could solve it eighty years ago.

We argue that from a constraint programming point of vue, the key aspects of the techniques designed by Polish and British cryptanalysts can be seen as, respectively, path consistency and singleton arc consistency on some constraint satisfaction problems.

**2012 ACM Subject Classification** Mathematics of computing → Combinatoric problems; Mathematics of computing → Combinatorial optimization

**Keywords and phrases** Constraint Programming, Cryptography

**Digital Object Identifier** 10.4230/LIPIcs.CP.2021.50

## 1 Introduction

Enigma was a cipher machine that had been commercialised since 1923. Breaking its code was a decisive breakthrough with a significant impact on the outcome of World War II.

The machine resembles a portable typewritter. Once configured in a particular setting agreed upon by the sender and the receiver, it can be used to encrypt a message. When typing with the machine, each letter is ciphered to a seemingly random letter indicated by a light bulb. The encrypted message, or *ciphertext*, can be deciphered by the receiver using his own Enigma machine. The code is indeed symmetric and typing the ciphertext with the same machine setting yields the original message.

The Enigma code was first broken by the mathematicians Marian Rejewski, Jerzy Różycki and Henryck Zygalski for the Polish Cipher Bureau before the war, although this method relied on a weakness due to an operating practice that was abandoned during the war. This knowledge on the machine, however, was shared with the allies and helped British cryptanalysts to break the code. To this end, Alan Turing and Gordon Welchman designed "The Bombe", an electro-mechanical device that made it possible to decipher Enigma's encrypted messages until the end of the war.

In this paper, we look back to this story from the viewpoint of constraint programming. We first describe a constraint model to break Enigma's code in Section 4. Given a portion of the

original message (the *crib*), a solution of this model represents the internal settings of Enigma (the *cryptographic key*) that produced, and can decrypt, the intercepted communication. Modelling this constraint satisfaction problem was the topic of a "hackathon" held during the ACP summer school in 2020, and several participants managed to break Enigma's code during this event.[1] This problem would be deceptively tricky to tackle without a rich modelling framework, and it nicely illustrates the effectiveness of constraint programming in that respect. However, solving this model with state-of-the-art solvers appears to be challenging, which highlights the prowess that was solving this problem eighty years ago. Interestingly, it appears that both methods used by the Polish Cipher Bureau or at Bletchley Park can be equated to known concepts of consistency: *Singleton Arc Consistency* on the constraint model introduced in this paper for the latter (see Section 5), and *Path Consistency* on some precomputed constraints for the former (see Section 6).

## 2 The Constraint Satisfaction Problem and Consistency

A Constraint Satisfaction Problem (CSP) is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{X} = \{x_1, \ldots, x_n\}$, is a set of *variables*; $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ a set of *domains*; and $\mathcal{C} = \{c_1, \ldots, c_t\}$ a set of *constraints*. An *assignment* maps[2] values to variables, we write $x \leftarrow v$ for the assignment of value $v$ to variable $x$. A constraint $c_j$ is given by a pair $(S(c_j), R(c_j))$ where $S(c_j)$ is a subset of $\mathcal{X}$, and $R(c_j)$ is $|S(c_j)|$-ary relation, that is, a set of satisfying assignments of $S(c_j)$.

The assignment $A = \{x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k\}$ of the set of variables $\{x_1, \ldots, x_k\}$ is *consistent* for a constraint $c$ if and only if its projection $\{x_i \leftarrow v_i \mid 1 \leq i \leq k \;\&\; x_i \in S(c)\}$ to $S(c)$ can be extended to an assignment in $R(c_j)$; Assignment $A$ is *globally* consistent if and only if it is consistent for every constraint in $\mathcal{C}$; it is *valid* if and only if, for every variable $x_i$, we have $v_i \in D(x_i)$. A *solution* of a CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ is a valid, globally consistent assignment of $\mathcal{X}$. We write $\mathcal{D}|_{x \leftarrow v}$ for the set of domains where $x$ is mapped to $\{v\}$ and equal to $\mathcal{D}$ on all other variables, and $\mathcal{D}' \subseteq \mathcal{D}$ when $\forall x \in \mathcal{X},\ D'(x) \subseteq D(x)$.

The notion of consistency is key to solving constraint satisfaction problems. We define here three consistencies that we shall relate to historical methods for attacking Enigma's code. These definitions are standard generalisations to non-binary constraints. In particular, the definition of consistency of an assignment for a constraint as its *projection* to the constraint's scope being *extendable* to the constraint's relation is useful to generalize path consistency.

▶ **Definition 1.** *A* support *for a constraint $c$ is a valid and consistent assignment of $S(c)$.*

▶ **Definition 2** (Arc Consistency ($AC$) [6])**.** *A variable $x$ is* arc consistent *(AC) with respect to a constraint $c$ if and only if, for each $v \in D(x)$, there exists a support for $c$ that contains $x \leftarrow v$. A constraint $c$ is AC if and only if every variable $x \in S(c)$ is AC with respect to $c$. A CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is AC if and only if every constraint $c \in \mathcal{C}$ is AC.*

▶ **Definition 3** (Singleton Arc Consistency ($SAC$) [3])**.** *An instantiation $x \leftarrow v$ is* singleton arc consistent *(SAC) if and only if there exists $\mathcal{D}' \subseteq \mathcal{D}|_{x \leftarrow v}$ such that the CSP $(\mathcal{X}, \mathcal{D}', \mathcal{C})$ is AC. A variable $x$ is SAC if and only if, and for each $v \in D(x)$, $x \leftarrow v$ is SAC. A CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is SAC if and only if every variable $x \in \mathcal{X}$ is SAC.*

▶ **Definition 4** (Path Support)**.** *Given a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ and three distinct variables $x_1, x_2$ and $x_3$, the assignment $\{x_3 \leftarrow v_3\}$ is a* path support *of assignment $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2\}$ for*

[1] `https://acp-iaro-school.sciencesconf.org/`
[2] We use functions instead of tuples to make variable ordering irrelevant.

*variable $x_3$ if and only if the assignment $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2, x_3 \leftarrow v_3\}$ is valid and globally consistent.*

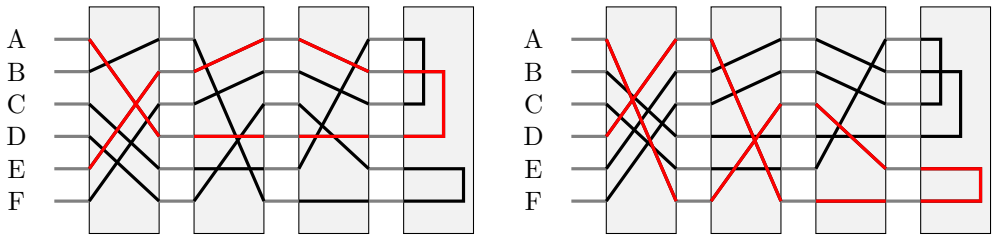
▶ **Definition 5** (Path Consistency ($PC$) [7]). *An assignment $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2\}$ of two variables is* path consistent *($PC$) if and only if it has a path support for every variable.*

*A CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is PC if and only if every valid and globally consistent assignment of every pair of variables is path consistent.*

*Enforcing* (singleton) *AC* on a CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ means finding the largest domain $\mathcal{D}'$, in the sense of inclusion as defined above, such that $\mathcal{D}' \subseteq \mathcal{D}$ and $(\mathcal{X}, \mathcal{D}', \mathcal{C})$ is (singleton) *AC*. Notice that there is a single such fix point, since the set of possible domains forms a lattice where infimum and supremum are obtained respectively by the intersection and the union.

## 3 The Enigma Code

In its simplest form, Enigma's encryption system is composed of three *rotor* wheels and a *reflector* wheel. A rotor wheel can be seen as a *simple substitution cipher* whereby every letter is mapped to a given letter, forming a permutation of the alphabet. The signal then goes through the two other wheels, then through the reflector and finally through the three rotors but backwards. Figure 1a illustrates, on a reduced alphabet, the rotor wheels (first three boxes) and reflectors (last box) wiring the input keyboard to an output system composed of lightbulbs indicating the substituted letter. In this case, pressing the key `A` eventually lights the bulb `E`. In other words, this mechanism is a simple substitution cipher whereby the alphabet is applied a permutation, e.g., (`AE`)(`BD`)(`CF`) in Figure 1a. Notice that the reflector wheel is symmetric and antireflexive. As a result, the overall permutation is symmetric and the same machine can therefore be used to decipher: pressing the key `E` lights bulb `A`. Moreover, it is antireflexive: no letter is mapped to itself, which proved to be a weakness.

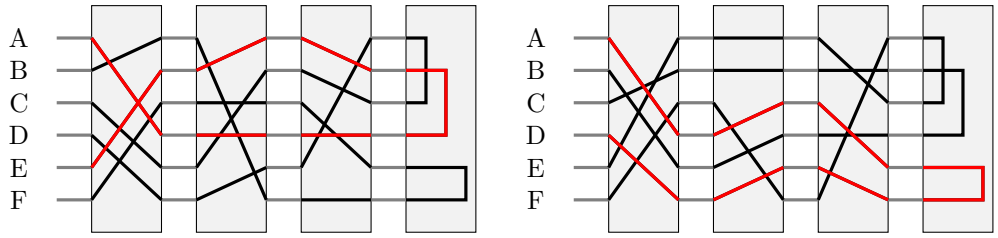



**(a)** Original rotors and reflector wiring. **(b)** The same machine after one key stroke.

**Figure 1** Illustration of Enigma's rotors and reflector.

However, the feature that made Enigma such a strong cryptographic system is that rotors are not static: they advance at every key stroke. More precisely, at every key stroke the "fast" rotor (leftmost) advances one step. Figure 1b shows the same machine after one key stroke. The fast rotor is shifted down by one position: the wire that was connecting `B` to `A` nows connects `A` to `F`, and so forth. As a result, pressing the same letter `A` now lights bulb `D`.

Moreover, when the fast rotor has completed a turn and is back to its reference position, a turnover notch is activated, and the middle rotor advances one step. Similarly, when the middle rotor has completed a turn, the "slow" (rightmost) rotor advances one step. Figure 2 shows the same machine after 6 key strokes (the fast rotor has made a full turn and is back to its original position and the middle rotor has advanced 1 step), or after 95 key strokes (the fast rotor has advanced 5 steps from its reference position, the middle rotor 3 steps and the slow rotor 2 steps). The simple substitution cipher in the latter position is (`AD`)(`BC`)(`EF`).




**(a)** After 3 key strokes **(b)** After 95 key strokes.

**Figure 2** Illustration of the same machine as in Figure 1a.

All Enigma machines shared the same set of three rotor wheels and the same reflector wheel, however, rotors could be rearranged in any order and initialised in any position. For instance, the daily settings could be given as "312, `CSP`". In that case, on that day, the 3rd rotor wheel would placed on the left, the 1st wheel on the middle and the 2nd wheel on the right. Moreover, before (de)ciphering any message, the left, middle and right rotor wheels would be positionned respectively so that the letters `C`, `S` and `P` showed up in some windows designed for this purpose. Additionally, the positions of the turnover notches of the left and middle rotor wheels, indicating the reference position at which the next rotor to the right would advance, could also be changed. The cryptographic key shared by the sender and receiver of the message was therefore the 3! *rotor orders* and the $26^3$ *rotor positions*, as well as the $26^2$ *reference positions*.[3] Therefore, there are $3! \times 26^3 \times 26^2 = 72188256$ possible keys.[4]

The first versions of the machine, commercialised in the 1920's operated on those principles, and hence had relatively few possible cryptographic keys. The version used by the German military added one sophistication: a *plugboard* (or *steckerbrett*) inserted between the input keyboard and the rotor scrambler, and also between the scrambler and the output light bulbs. The plugboard is also a simple substitution cipher, mapping $L$ letters to another – different – letter, and the remaining $26 - 2L$ to themselves. However, this further cipher could easily be changed manually and was therefore part of the cryptographic key. For $L = 10$ plugs, the number of possibilities is $150,738,274,937,250$, and the total number of possible settings is thus $158,962,555,217,826,360,000$ and even $26^2$ times more counting the reference positions.

*Breaking the code* entails finding, for a given ciphertext, the *rotor settings* (the choice of rotors, their order, the reference position of the two leftmost rotors, and their initial position) and the *plugboard configuration*. Those settings can be seen as the cryptographic key that one needs to recover in order to decipher the messages. However, if every component contributes to combinatorial number of keys, the rotor position is the most important. Indeed, the reference positions do not affect the first letters of the message. As long as the middle rotor does not advance, the text is unchanged. Moreover, the reference position of the first rotor can be deduced from when the text starts being gibberish. The plugboard configuration only affects part of the letters, for instance with six plugs, fourteen letters are not changed. Therefore given a correct guess on the order and position for the rotors, setting up the reference position arbitrary (say to `AA`) and the plugboard to the identity, would be sufficient to decrypt a small portion of the message. Therefore, verifying that this guess is correct is relatively easy, as well as deducing the reference position and the plugboard configuration. Finally, the rotor ordering (and choice thereof) was often guessed by other means than

[3] Only the left and middle rotors have a turnover notch.
[4] Later versions introduced two more rotors to choose from, raising this number to ${}^3P_5 \times 26^5 = 721882560$.

computation, or the attack was repeated for every possible order until it succeeded.

## 4 A Constraint Programming Model

In this section we introduce a constraint model that emulates the Enigma machine, and can be used to break its code. This model assumes that the rotors and their order are known. Since it is actually unknown, the problem might have to be solved $3! = 6$ (resp. ${}^3P_5 = 60$) times for three rotor wheels (resp. three out of five).

Let **ciphertext** be a string of $K$ letters; **rotor** be the rotor wiring, whereby **rotor**$[j][x]$ is the output letter, on input $x$, of rotor $j$ in its reference position; and **reflector** be the reflector wiring, whereby **reflector**$[x]$ is the output of the reflector on input $x$. In the following, we use $N$ for the size of the alphabet and $M$ for the number of rotor wheels (respectively 26 and 3 for Enigma). Moreover, we write $[a, b]$ for the discrete interval $\{a, a+1, \ldots, b\}$ and $[b]$ as a shortcut for $[0, b-1]$. The model uses four sets of variables:

$$\begin{aligned}
\textbf{plaintext} &: plaintext_i \in [N] && \forall i \in [K] \\
\textbf{key} &: key_j \in [N] && \forall j \in [M] \\
\textbf{ref} &: ref_j \in [N] && \forall j \in [M-1] \\
\textbf{plugboard} &: plug[x] \in [N] && \forall x \in [N]
\end{aligned}$$

The variables **plaintext** stand for the original message. The other variables stand for the settings of the machine used to cipher it: the variables **key** stand for the rotor position (rotor wheel $j$ was advanced by $key_j$ steps) the variables **ref** stand for the reference position of the two leftmost rotor wheels (rotor $j+1$ advances one step when rotor $j$ returns to position $ref_j$), and the variables **plugboard** stand for the plugboard connection ($plug[x]$ is the letter "steckered" to $x$ by the plugboard). In a nutshell it ensures that an Enigma machine with rotors **rotor** that have been setup with reference position **ref**, in initial position **key** and with plugboard configuration **plugboard** ciphers **plaintext** to **ciphertext**. We use the auxiliary variables $\mathbf{x}$ with $x_{i,j} \in [N]\ \forall i \in [K], \forall j \in [2M+2]$, with $x_{i,j}$ standing for the output of the scrambling device $j$ for the letter at position $i$ in the plaintext. The scrambling devices are ordered as explained in Section 3: plugboard first, then the three rotors, followed by the reflector, the three rotors backwards, and finally the plugboard again. We also use the auxiliary variables $\mathbf{p}$ with $p_{i,j} \in [K + K/N^j], \forall i \in [K], \forall j \in [M]$, to represent the total number of steps that the $j+1$-th rotor wheel has advanced when reading the $i+1$-th letter. Finally, the auxiliary variables **offset** with, for $j \in [2]$, $offset_j$ represent the reduction in number of steps to advance for rotor $j$ to reach its reference position $ref_j$ for the first time.[5]

In the reminder of the section, we will extensively use the *Element* constraint [4]:

▶ **Definition 6** (*Element*). *Let* $\mathbf{x} = \{x_1, \ldots, x_K\}$ *be a set of variables, and* $k, y$ *be two variables. The constraint* $Element(\mathbf{x}, k, y)$ *is the pair* $(S, R)$ *where the relation* $R$ *contains all assignments of* $S = \mathbf{x} \cup \{k, y\}$ *satisfying the predicate* $x_k = y$.

When we write the expression "$x_k$", with $k$ a variable and $\mathbf{x} = x_1, \ldots, x_n$ an array of variables, this should be read as an extra variable $y$ constrained with $Element(\mathbf{x}, k, y)$. Similarly, for any arithmetic operator $\oplus \in \{+, -, *, /, \text{mod}\}$, the expression "$x_1 \oplus x_2$" should be read as an extra variable $y$ with the constraint defined by the predicate $x_1 \oplus x_2 = y$.

[5] We include these variables for completeness, although they are often both set to the constant 0 (`A`).

## 4.1 Forward Rotor Model

If $\mathbf{rotor}[j][x]$ is the letter read after going through rotor $j$ from input $x$, then after advancing $k$ turns, the rotor produces the letter $(\mathbf{rotor}[j][(x+k) \bmod N] - k) \bmod N$ on the same input. The relation between the input letter $x_{i,j}$ and the output letter $x_{i,j+1}$ of the forward traversal of rotor $j$ can therefore be encoded as follows:

$$p_{i,0} = key_0 + i \qquad \forall i \in [K] \qquad (1)$$

$$offset_j = \begin{cases} -ref_j & \text{if } ref_j \leq key_j \\ N - ref_j & \text{otherwise} \end{cases} \qquad \forall j \in [M-1] \qquad (2)$$

$$p_{i,j} = key_j + \frac{p_{i,j-1} + offset_{j-1}}{N} \qquad \forall i \in [K], \forall j \in [1, M] \qquad (3)$$

$$x_{i,j+1} = (\mathbf{rotor}[j][(x_{i,j} + p_{i,j}) \bmod N] - p_{i,j}) \bmod N \qquad \forall i \in [K], \forall j \in [M] \qquad (4)$$

Constraints (1, 2 and 3) channel the auxiliary variables $\mathbf{p}$, where $p_{i,j}$ stands for the number of times the rotor $j$ has turned starting for position 0 when reading the $i$-th letter, to the initial positions **key** of the rotors and to their reference position **ref** via the variables **offset**. Constraints (4) represent the substitution cipher used with input letter $x_{i,j}$ and output letter $x_{i,j+1}$: if rotor $j$ advanced $p$ steps, the wire that initially connected the letter $\alpha$ to the letter $\beta$ now connects the letter $\alpha - p$ to the letter $\beta - p$ (modulo $N = 26$).

## 4.2 Reflector Model & Backward Rotor Model

The reflector is also a substitution cipher, but static (it does not change from a letter to the next), symmetric ($\forall x, \forall y\ \mathbf{reflector}[x] = y \iff \mathbf{reflector}[y] = x$) and antireflexive ($\forall x\ \mathbf{reflector}[x] \neq x$). Constraints 5 model the relation between the input $x_{i,M+1}$ of the reflector (the signal corresponding to the $i$-th letter after going through the all rotors forward) and its output $x_{i,M+2}$ with another *Element* constraint. Then, the signal travels through the rotors, but backward. Constraints 6 are similar as for the forward pass.

$$x_{i,M+1} = \mathbf{reflector}[x_{i,M}] \qquad \forall i \in [K] \qquad (5)$$

$$x_{i,M+j+1} = (\mathbf{rotor}[j][(x_{i,M+j+2} + p_{i,M-j}) \bmod N] - p_{i,M-j}) \bmod N \qquad \forall i \in [K], \forall j \in [M] \qquad (6)$$

## 4.3 Plugboard Model

Finally, we need to model the plugboard of the military version. Since it is composed of $L$ plugs, each one connecting two letters, it is a symmetric permutation with $N - 2L$ *fixed points*, i.e., it leaves $N - 2L$ letters unchanged. Unlike the reflector (which is fully known) or the rotors (for which the only unknowns are their positions), the plugboard is not known for the attacker: its configuration is part of the cryptographic key to be computed during the attack. It is encoded as a vector $\mathbf{plugboard} = \langle plug[1], \ldots, plug[N] \rangle$ of variables with domain $[N]$ where $plug[x]$ stands for the letter mapped to letter $x$, and the following constraints:

$$\text{AllDifferent}(\textbf{plugboard}) \tag{7}$$

$$(plug[x] = y) \Leftrightarrow (plug[y] = x) \qquad \forall x, y \in [N] \tag{8}$$

$$\sum_{x=1}^{N} (plug[x] = x) = N - 2L \tag{9}$$

$$plug[plaintext_i] = x_{i,0} \qquad \forall i \in [K] \tag{10}$$

$$plug[\textbf{ciphertext}[i]] = x_{i,2M+1} \qquad \forall i \in [K] \tag{11}$$

Constraints (8), (9) and (9) ensure that the plugboard is in a legal configuraton by stating that **plugboard** is a permutation (7); with $N - 2L$ identities (9)[6]; and is symmetric (8). Constraints (10) represent the transformation of the input by the plugboard, and Constraints (11) the final transformation, also by the plugboard, yielding the ciphertext as output.

## 4.4 Breaking the Code

So far, the model introduced in this section emulates the Enigma machine: a solution stands for a plaintext **x** whose cipher with message key **key** and plugboard **plugboard** is the given ciphertext. However, there are too many consistent assignments of **x**, **key** and **plugboard** to consider enumerating the solutions in order to find the original text.

In order to actually break the code, we use the same technique used by cryptanalyst at Bletchley Park in the 1940s. We suppose that we are somehow given a "crib", that is, a portion of deciphered message.[7] For instance, suppose that we know that the plaintext corresponding to the portion of ciphertext `IRSJYTCORS` starting at position $s$ is in fact `CONSTRAINT`. Plaintext and ciphertext can therefore be aligned as shown in Table 1.

| $s$ | $s+1$ | $s+2$ | $s+3$ | $s+4$ | $s+5$ | $s+6$ | $s+7$ | $s+8$ | $s+9$ |
|---|---|---|---|---|---|---|---|---|---|
| C | O | N | S | T | R | A | I | N | T |
| I | R | S | J | Y | T | C | O | R | S |

**Table 1** A crib: an alignment of plaintext and ciphertext

Given a string of plaintext **T** and a string of ciphertext **C** such that $|\mathbf{T}| = |\mathbf{C}|$ starting at position $s$ corresponding to a crib, we can find compatible initial rotor positions **key**, and plugboard configurations **plugboard**, by solving the model introduced in this section[8] with $K = |\mathbf{C}|$, Constraints (1–11), as well as the equalities:

$$plaintext_i = \mathbf{T}[i] \qquad \forall i \in [s, s+K] \tag{12}$$

This model may have several solutions, and only one corresponds to the actual cryptographic key. In practice, however, even small cribs can have a reasonable number of solutions, and verifying them manually can be done by deciphering the rest of the ciphertext with the same

[6] The number of connections $L$ went from 6 to 10 depending on Enigma's versions. The equality "$plug[x] = x$" is read as its natural conversion from Boolean to $\{0, 1\}$.

[7] Cribs were obtained by guessing that a word or a sentence was likely to be in the message, and using the fact that a letter is never ciphered to itself to align that portion of plaintext with the ciphertext.

[8] Notice that the definition of several constraints must take into account the positional offset $s$.

key. Experimental evaluations show that solving this model using standard CSP solvers is not trivial (see Section 7). In the two following sections, we review how the code was broken in practice and show how those ideas can be mapped to the notion of consistency. These observations directly lead to an efficient CP based method to break Enigma's code.

## 5 Breaking Enigma's Code: The British Method

The method developped by Alan Turing (and improved by Gordon Welchman) computes the rotor position and the plugboard configuration used to cipher the message. It ignores the reference positions, however, and actually fails if there was a turnover (i.e., if the middle rotor advanced when ciphering the crib). A very advanced machine for its time, called "Bombe" was built for that purpose at Bletchley Park. The Bombe was not only capable of quickly simulating several rotor positions in parallel, but it could also either rule out a message key, or (partially) compute a plugboard configuration consistent with that message key.

The method used by British cryptanalists also relied on acquiring a crib, that is, a string $\mathbf{T}$ of original text aligned to a string $\mathbf{C}$ of same length in the ciphertext, as shown in Table 1. A *menu* for the Bombe was then extracted from the crib:

▶ **Definition 7** (Menu). *Given a string of plaintext* $\mathbf{T}$ *and a string of ciphertext* $\mathbf{C}$ *such that* $|\mathbf{T}| = |\mathbf{C}|$*, the* menu *obtained from matching them is a graph with one vertex per letter in* $\mathbf{T} \cup \mathbf{C}$*, and an edge for each pair of matched letters* $\mathbf{T}[i], \mathbf{C}[i]$ *labelled by their position, i.e.,* $G = (\mathbf{T} \cup \mathbf{C}, \{(\{\mathbf{T}[i], \mathbf{C}[i]\}, i) \mid i \in [|\mathbf{T}|]\})$*. We write* $N_G(x) = \{(y, i) \mid (\{x, y\}, i) \in G\}$ *for the neighborhood of letter* $x$ *in the menu. Notice that it carries the edge labels.*

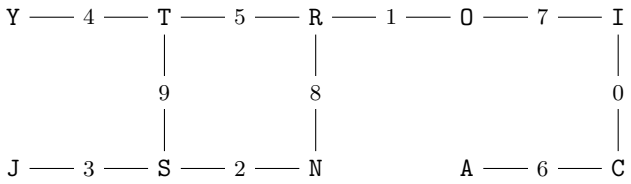



**Figure 3** Illustration of the Bombe's menu from the crib in Table 1

Figure 3 illustrates the menu extracted from the crib of Example 1.[9] An edge, say $(\{A, C\}, 6)$, of the menu indicates the letter `A` (resp. `C`) at position 6 is ciphered to `C` (resp. `A`). Now, observe that given some positions for the rotors, the Bombe can compute the scramble function (i.e., permutation of the alphabet) $S_i : [N] \mapsto [N]$ corresponding to the Enigma machine, ignoring the plugboard. It is then possible to make some inference on the configuration of the plugboard, as sketched in Figure 4. Indeed, suppose that the plugboard associates `A` to `B`. We can compute $S_i(B)$, and in particular $S_6(\texttt{B})$, let it be `R`. Now thanks to the crib, we know that the cipher for `A` at position 6 is `C`, therefore the plugboard must associate `R` to `C`. This process can be repeated starting from any edge of the menu ending in `R` or `C`, yielding more deductions. Eventually, a contradiction might be found, or a fixed point might be reached. This is Turing and Welchman's inference rule, which can be formalised as follows:

▶ **Definition 8** (Plugboard configuration). *Let* $\mathcal{P}$ *be a collection of subsets of* $[N]$ *such that* $|p| \leq 2\ \forall p \in \mathcal{P}$*. We say that* $\mathcal{P}$ *is* valid *if and only if elements of* $\mathcal{P}$ *are pairwise disjoint*

[9] For the sake of the example, we let the first index $s$ be 0

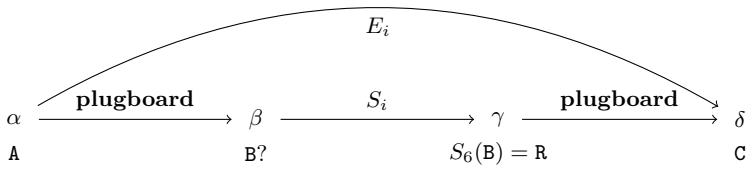



**Figure 4** Illustration of Enigma's encryption scheme: the input character (here $\alpha$) is mapped to $\beta$ by the plugboard; then scrambled to $\gamma$ by going through the rotors forward, the reflector and the rotors backward; and finally mapped to $\delta$ by the plugboard.

*and contains no more than $L$ pairs and no more than $N - 2L$ singletons. We say that $\mathcal{P}$ is* complete *if and only if $\bigcup_{p\in\mathcal{P}} p = [N]$. The collection $\mathcal{P}$ is valid and complete if and only if it corresponds to a legal plugboard configuration where, for every $\{\alpha, \beta\} \in \mathcal{P}$ the letters $\alpha$ and $\beta$ are connected, and for every $\{\alpha\} \in \mathcal{P}$, the letter $\alpha$ is not changed by the plugboard.*

▶ **Proposition 9** (Inference rule). *Let $S$ be a scrambler (Enigma without the plugboard) and $G$ a menu. If $\mathcal{P}$ is the valid and complete plugboard configuration used during encryption of $G$, then, for every $\{\alpha, \beta\} \in \mathcal{P}$:*

- *for every $(\delta, i) \in N_G(\alpha)$, $\{\delta, S_i(\beta)\} \in \mathcal{P}$, and*
- *for every $(\delta, i) \in N_G(\beta)$, $\{\delta, S_i(\alpha)\} \in \mathcal{P}$.*

**Proof.** Suppose that the first rule does not hold, i.e., $\{\alpha, \beta\} \in \mathcal{P}$, $\exists(\delta, i) \in N_G(\alpha)$ such that $\{\delta, S_i(\beta)\} \notin \mathcal{P}$. The letter $\alpha$ is mapped to $\beta$ by the plugboard and is scrambled to $S_i(\beta)$. However, since there is an edge $(\{\alpha, \delta\}, i)$ in the menu, we know that the encryption setup was such that the input letter $\alpha$ at position $i$ yields letter $\delta$. Therefore, the plugboard must match the letters $S_i(\beta)$ and $\delta$. If there is no $p \in \mathcal{P}$ such that $\beta \in p$, then $\mathcal{P}$ is not complete, and otherwise, it is not consistent, hence a contradiction.

The second rule follows from the same reason but starting from letter $\beta$. ◀

This inference rule can be used to discard a guessed plugboard connection. The Bombe is an electro-mechanical device that automatises this process. It is composed of several clones of the Enigma machine, each capable of emulating the encryption of all 26 letters in parallel for a given rotor setting, and wires for every possible plugboard connection. Given a message key **key**, and a menu, the Bombe was initialised by setting up one of the clones to emulate the scrambler $S_i$ for every edge label $i$ in the menu. Then a connection $\{\alpha, \beta\}$ could be "guessed" by sending a current flow through the corresponding wire into the Bombe. The inference rule described in Proposition 9 would then be applied as the current flowed through the input $\beta$ of the clone $S_i$, for each $(\delta, i) \in N_G(\alpha)$. The current would in turn flow to the connection between $S_i(\beta)$ and $\delta$ and again to some clones of Enigma as indicated in the menu. This can either result in a fixed point where $\mathcal{P}$ is closed under that rule, or yields an invalid plugboard configuration. In the latter case, the connection $\{\alpha, \beta\}$ can be ruled out, and another guess can be made. If there exists a letter for which no matching is possible, the message **key** is ruled out and the same process is repeated for another message key. Otherwise, **key** is a candidate key and a (partial) plugboard configuration is given by the connection in which the current flows.

▶ **Definition 10** (The Bombe method). *The Bombe made it possible, starting from a tentative connection $\{\alpha, \beta\}$, to enforce the inference rule described in Proposition 9 until either it* fails

*($\mathcal{P}$ is no longer valid), or* succeeds *(it does not fail and reaches a fixed point). In the former case, the connection $\{\alpha, \beta\}$ can be ruled out.*

The Bombe method *denotes the process where, for each rotor position, every possible plugboard connection involving a letter in the crib is ruled out if the process above fails. The Bombe method causes a* stop *if and only if it reaches a fixed point where every letter can appear in at least one connection. In that case, the current rotor position might be the correct one (the message key) and it is checked by other means. Otherwise, this key is ruled out and the process resume with another of the $26^3$ positions.*

▶ **Lemma 11.** *Let $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be the CSP of Section 4. If the rule of Proposition 9 deduces the connection $\{\delta, \gamma\}$ from menu $G$ and $\mathcal{P} = \{\{\alpha, \beta\}\}$ then enforcing AC on $(\mathcal{X}, \mathcal{D}|_{plug[\alpha] \leftarrow \beta}, \mathcal{C})$, reduces the domain of $plug[\delta]$ to $\{\gamma\}$ and of $plug[\gamma]$ to $\{\delta\}$.*

**Proof.** Suppose that the rule of Proposition 9 deduces the connection $\{\delta, \gamma\}$. We assume first that the first rule triggered and hence $(\delta, i) \in N_G(\alpha)$ and $S_i(\beta) = \gamma$.

Constraints (1 – 6) and (10 – 12) are all functional: when all but one of their variables are fixed, the last variable has a single consistent value.

Since $(\delta, i) \in N_G(\alpha)$, the letter $\alpha$ in the plaintext of the crib is matched to $\delta$ in the ciphertext at position $i$. By enforcing $AC$ on Constraint 12, we have $D(plaintext_i) = \{\alpha\}$, and since $D(plug[\alpha]) = \{\beta\}$, by enforcing $AC$ on Constraint (10) we have $D(x_{i,0}) = \{\beta\}$.

Moreover, since **key** is constant, so is **p**, by enforcing $AC$ on Constraint (1) since only one non-constant variable remains. As a result, enforcing $AC$ on Constraints (4) reduces the domains of variables $x_{i,1}, x_{i,2}$ and $x_{i,3}$ to single values. The same is true for Constraint (5) and Constraints (6). The variable $x_{i,2M+1}$ is therefore assigned, and it must be to value $S_i(\beta) = \gamma$. Enforcing $AC$ on Constraint (11) sets the domain of variable $plug[\delta]$ to $\{\gamma\}$. Finally, Constraint (8) set the domain of $plug[\gamma]$ to $\{\delta\}$.

If it was the second rule that triggered, the same demonstration applies, although the chain of propagations to consider goes in the reverse direction: from $x_{j,2M+1}$ to $x_{j,0}$. ◀

The implication in Lemma 11 is not an equivalence simply because in some cases, Constraints 7, 8 and 9 might trigger and more connections could then be deduced via propagation. For instance if $L = 1$ and $\alpha \neq \beta$, then all variables in **plugboard** would be assigned by propagation of Constraint 9. The following theorem is also an implication for the same reasons, and in this case these constraints are even more likely to be relevant since the domains are reduced while enforcing $SAC$.

▶ **Theorem 12.** *If the Bombe method does not produce a stop, then enforcing singleton arc consistency on the model described in Section 4, with the variables* **key** *assigned, also fails.*

**Proof.** Let $\mathcal{CN} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be the CSP described in Section 4. In particular, **plugboard** $\subseteq \mathcal{X}$, however, **plaintext** and **key** are constant under the theorem's hypothesis.

Suppose that the Bombe method rules out a connection $\{\alpha, \beta\}$. It means that the rule of Proposition 9 produces an invalid plugboard configuration $\mathcal{P}$. However, from Lemma 11 we know that for every $\{\alpha, \beta\} \in \mathcal{P}$, after enforcing $AC$ on the CSP $(\mathcal{X}, \mathcal{D}|_{plug[\alpha] \leftarrow \beta}, \mathcal{C})$, all but the value $\beta$ (res.p $\alpha$) are removed from $D(plug[\alpha])$ (resp. $D(plug[\beta])$). It entails that there exist two elements of $\mathcal{P}$ that are not disjoint, e.g., $\{\alpha, \beta\} \in \mathcal{P}$ and $\{\alpha, \gamma\} \in \mathcal{P}$. In this case, the corresponding domain $D(plug[\alpha])$ is wiped out.

Therefore, every connection that is ruled out by the Bombe method is also ruled out by enforcing $SAC$. The Bombe method fails if all the connections involving a particular letter have been ruled out. If this happens for letter $\alpha$, then the domain of the variable $plug[\alpha]$ is emptied, and therefore $SAC$ fails. ◀

The Bombe can therefore be seen as a CSP solver that branches on the variables **key**, and enforces singleton arc consistency on leaves of this search tree. When $SAC$ fails, the rotor position corresponding to that branch is ruled out, otherwise the machine *stops*. Then, the (partial) plugboard configuration is verified manually. If it cannot be extended to a configuration that correctly deciphers the message, then this rotor position is ruled out as well, and the Bombe resumes its search with a new assignment of **key**. It was therefore important to reduce the likelihood of such *false stops*.

## 6 Breaking Enigma's Code: The Polish Method

The Enigma code was actually first broken, before the war, by Polish mathematicians Marian Rejewski, Jerzy Różycki and Henryck Zygalski [5], who also succesfully reconstructed, from partial and indirect intelligence, the Enigma machine and built mechanical devices to emulate it. Their insights later proved essential to the British effort. Turing and Welchman's method relied on a crib and would guess the rotor position and the plugboard configuration used to encrypt the message regardless of how it was chosen (in particular we shall see that this rotor position is different to the one given in the daily settings). The method of the Polish Cipher Bureau, on the other hand, relied on intercepting several messages sent with the same daily settings, and would compute not directly the message key, but the daily settings.

Using repeatedly the same encryption key would be a serious security flaw. Operators were thus instructed to randomly choose a 3-letter *message key*. This key (e.g., `SAT`) would be encrypted *twice*[10] using the rotor position indicated in the daily settings (e.g., the text `SATSAT` would be ciphered using the key `CSP` to some 6-letter ciphertext). Then, the machine would be reset to position `SAT` and the message woud be ciphered using that message key, and both ciphers (6-letter prefix and text) sent in the same message. The receiver would then set its own Enigma machine to postion `CSP`, decipher the prefix, reset its machine to the obtained position `SAT`, and finally decrypt the message. This method still had the weakness of using the same key (here, `CSP`) to encrypt all messages keys for a given day. Rejewski devised a first method, involving a dedicated machine, the *cyclometer*, that partially automated the design of a lookup table (the *cards catalog*) from which the daily rotor positions could easily be found, provided several prefixes encrypted with the same daily settings [1].

The procedure was therefore upgraded: the sender chose a *plaintext key* besides the message key. Then, the rotors were positioned according to the daily rotor settings *shifted by the plaintext key*. In other words, the actual encryption key was equal to *rotor settings* + *plaintext key*, where "+" stands for the modular, component-wise addition. This key was used to encrypt the message key and it was sent in plaintext with the encrypted message. For instance, if the rotor setting is `CSP`, the chosen plaintext key `MIP`, then the rotors would be put in position `OAE` = `CSP` + `MIP` to cipher the chosen message key (say `SAT`). The text `SATSAT` would yield, for instance, `DGFAGX`. Then the rotors were reset to position `SAT` and the actual message was ciphered to "**ciphertext**". Finally, the message sent would be:

`MIP DGFAGX` **ciphertext**

The receiver would add the plaintext key to the daily setting to recover the actual rotor position, then, as previously, decipher the message key, reset the machine to the corresponding rotor position and decipher the message. The difference with the previous practice, however,

[10] This practice was abandoned May 1st 1940, hence making the Polish attack irrelevant.

was that the rotor position used to cipher the 6-letter prefix was never twice the same. The fact that the message key was repeated twice, however, proved nonetheless to be a fatal flaw.

When rotors and plugboard are in a given configuration, Enigma corresponds to a symmetric permutation, although the permutation changes with every letter since the rotors advance. Let $E_i^{\texttt{XYZ}}$ stand for the permutation applied by Enigma to the $i$-th letter with initial rotor position `XYZ`.[11] Consider the prefix `MIP DGFAGX` and let $\mathbf{key} = \{key_1, key_2, key_3\}$ denote the unknown rotor setting common to all messages of a given day. We know that a letter (say $x$) is mapped to the letter `D` in $E_1^{\mathbf{key+MIP}}$ and to `A` in $E_4^{\mathbf{key+MIP}}$. Moreover, since $E_1^{\mathbf{key+MIP}}$ is symmeric, it maps `A` to $x$. Therefore, the composition $E_1^{\mathbf{key+MIP}} E_4^{\mathbf{key+MIP}}$ maps `D` to `A` and vice versa. Because the plugboard is unknown we do not learn anything from that.

However, observe that $E_2^{\mathbf{key+MIP}} E_5^{\mathbf{key+MIP}}$ transforms the letter `G` to itself. Mathematician Henryk Zygalski noticed that this was relevant because the plugboard changes the letter but conserve the fixed points of permutation $E_2^{\mathbf{key+MIP}} E_5^{\mathbf{key+MIP}}$ (an element unchanged by the permutation), and because not all rotor settings have such fixed points [5].

A device, "the Bomba",[12] was designed and built by the Polish Cipher Bureau to go over all of the $26^3$ rotor settings, and record which settings could allow such a fixed point in one of the three composed permutations. Only 40% of the rotor positions had a composition fixed point. A set of 26 perforated sheets (known as the *Zygalski sheets*) were to be produced for each of the 6 possible rotor orders.[13] Each sheet corresponds to a letter, standing for the first letter of the unknown rotor position, and contains a $26 \times 26$ Boolean matrix standing for whether the second and third letters could extend the first letter to a position allowing a fixed point. We denote $Z[\alpha, \beta, \gamma]$ the fact that there is a hole at positions $\beta, \gamma$ in the Zygalski sheet for letter $\alpha$, which is true if and only if the rotor position $\alpha\beta\gamma$ has a composition fixed point (and hence may encode the two occurrences of a letter in the prefix to the same code).

Consider now a prefix `MIP VNEVSX`. There is a fixed point at position $(1, 4)$, hence the Zygalski sheets allowed some inference: one would first guess the order of the rotors and a first letter $key_1$ indicating the position of the first rotor. Then, let $\alpha = key_1 + \texttt{M}$. The $\alpha$-th sheet would be taken from the box standing for the chosen order. The matrix on that sheet (shifted by `I` and `P` in the respective dimensions[14]) thus stands for the possible values of $key_2$ and $key_3$. This narrows down the number of possibilities by 60%. Now, suppose that a message with prefix: `SMT DGFAGX` is intercepted the same day. This prefix also has a fixed point, but at position $(2, 5)$. Therefore, the same reasoning applies, using sheet $key_1 + \texttt{S} + 1$ in the same box, but shifted by `M` and `T`. The "+1" models that the fixed point is on the 2nd and 5th letters, which is equivalent to observing it on the 1st and 4th letters, however with the position of the left rotor advanced one step. The subset of holes that allow both fixed points is obtained by shining light through the two sheets, properly shifted and aligned. Usually, a dozen messages including a fixed point (and as many sheets) were necessary to narrow down the number of possibilities to either 0, in which case the guess was proven wrong; or 1, in which case the rotor position **key** and the rotor order could be easily retrieved.

This method does not take the plugboard into consideration. However, since 14 letters were not affected by the plugboard[15], it is possible to reconstruct the message manually. It also fails in case a turnover of the middle rotor happens before the end of the 6-letter prefix.

[11] The plugboard is ignored here.
[12] An aggregate of six Enigma machines, one for each permutations of the prefix.
[13] The process was long and difficult, even with the Bomba: only 2 sets had been completed when the invasion of Poland began, but the sheets were eventually finalized at Blechtley Park.
[14] The sheets had 25 repeated rows and columns to allow for shifting modulo 26
[15] Only 6 plugs were used when the Polish began to attack Enigma

Aligning several Zygalski sheets corresponds to solving the following CSP:

▶ **Definition 13** (Zygalski's CSP). *We call Zygalski's CSP the constraint satisfaction problem with set of variables* **key** $= \{key_1, key_2, key_3\}$*, and for each message with plaintext key* $\alpha\beta\gamma$ *containing a fixed point at positions* $(1+i, 4+i)$*, a constraint given by the predicate* $Z[key_1 + \alpha + i, key_2 + \beta, key_3 + \gamma]$.

▶ **Theorem 14.** *The letter* $\alpha$ *for the position of the first rotor is refuted by the Zygalski sheets method if and only if no pair of instantiations* $\{key_2 \leftarrow \beta, key_3 \leftarrow \gamma\}$ *has* $key_1 \leftarrow \alpha$ *as path support in Zygalski's CSP.*

**Proof.** By definition, a hole with coordinates $\beta, \gamma$ will let the light shine through all sheets, if and only if $\{key_1 \leftarrow \alpha, key_2 \leftarrow \beta, key_3 \leftarrow \gamma\}$ is consistent with every constraint of Zygalski's CSP, that is, if and only if $key_1 \leftarrow \alpha$ is a path support of $\{key_2 \leftarrow \beta, key_3 \leftarrow \gamma\}$. ◀

The process of superimposing several Zygalski sheets corresponding to a guess of the letter $\alpha$ for the position of the first rotor is an efficient method to compute the 2-tuples which have $key_1 \leftarrow \alpha$ as path consistent support. Moreover, since there are only three variables in the network, if $key_1 \leftarrow \alpha$ is a path support of $\{key_2 \leftarrow \beta, key_3 \leftarrow \gamma\}$ then $key_2 \leftarrow \beta$ is a path support of $\{key_1 \leftarrow \alpha, key_3 \leftarrow \gamma\}$ and $key_3 \leftarrow \gamma$ is a path support of $\{key_1 \leftarrow \alpha, key_2 \leftarrow \beta\}$. Therefore path consistency can be achieved by only checking the values of a single variable in this way, there is a solution if and only there is a tuple with a path support.

Path consistency is usually only applied to binary constraints. However, the definition we use naturally extends this consistency to non-binary constraints and in that context, the analogy stands. For instance, consider two Zygalski sheets: one allowing the keys `ABC`, `ADE`, `BBE` and `BDC` and one allowing the keys `ABE`, `ADC`, `BBC` and `BDE`. There is no solution, and indeed the Zygalski CSP is not $PC$ (e.g., the consistent and valid assignemnt $\{key_1 \leftarrow \texttt{A}, key_2 \leftarrow \texttt{B}\}$ cannot be extend to the third variable), yet the CSP is $AC$ and $SAC$.

## 7 Experimental Evaluation

We ran experiments to verify the impact of $SAC$ on this problem. The constraint model was implemented using the toolkit Choco [8].[16] To emulate the Bombe, we force the heuristic to select the variables standing for the positions of the rotors (**key**) before other variables. In the version denoted `Choco+SAC`, we run $SAC$ only once these variables are all assigned. When $SAC$ does not fail, which corresponds to a stop of the Bombe, or in the default version denoted `Choco`, we let the constraint solver either find a solution for the variables **plugboard**, or prove that no solution exists for the current rotor position. For both methods we treated every variable in **ref** as the constant 0 as was done in the methods we discussed previously. Not doing so would increase the search space, and the number of solutions, by a factor $26^2$.

In order to generate benchmark instances, we used many cribs of length 12 from wikipedia texts that we ciphered using a random position of the rotors I, II and III of the first Enigma machine that was introduced in 1930 [9], and a random reflector.

| | |
|---|---|
| Fast: | E K M F L G D Q V Z N T O W Y H X U S P A I B R C J |
| Middle: | A J D K S I R U X B L H W T M C Q G Z N P Y F V O E |
| Slow: | B D F H J L C P R T X V Z N Y E I W G A K M U S Q O |
| Reflector: | P R Y Z L X O S Q K J E N M G A I B H W V U T F C D |

[16] The source code is available here: `https://gitlab.laas.fr/vantuori/sacnigma.git`.

| Crib | | | | | `Choco` | | | `Choco+SAC` | | |
|---|---|---|---|---|---|---|---|---|---|---|
| #inst. | #cycle | size | #stops | #sol. | #solved | CPU | #fail | #solved | CPU | #fail |
| 10 | 0 | 12 | 128.4 | 17912 | 3 | 378 | 388790.7 | 10 | 1197 | 17448.6 |
| 10 | 0 | 13 | 242.8 | 12214 | 7 | 901 | 1036519.1 | 10 | 922 | 17334.3 |
| 10 | 0 | 14 | 5.6 | 593 | 7 | 989 | 968841.6 | 10 | 866 | 17570.4 |
| 10 | 0 | 15 | 2.0 | 26 | 8 | 438 | 414718.2 | 10 | 607 | 17574.0 |
| 10 | 0 | 16 | 1.0 | 14 | 8 | 1208 | 1251729.9 | 10 | 813 | 17575.0 |
| 10 | 1 | 11 | 68.6 | 124111 | 5 | 475 | 542790.6 | 10 | 908 | 17507.5 |
| 10 | 1 | 12 | 156.2 | 33912 | 6 | 958 | 1435607.2 | 10 | 751 | 17421.3 |
| 10 | 1 | 13 | 1.8 | 1182 | 5 | 604 | 693048.2 | 10 | 1061 | 17574.2 |
| 10 | 1 | 14 | 1.9 | 166 | 8 | 606 | 580758.1 | 10 | 748 | 17574.1 |
| 10 | 1 | 15 | 1.0 | 77 | 8 | 369 | 350508.9 | 10 | 560 | 17575.0 |
| 10 | 1 | 16 | 1.0 | 40 | 8 | 1057 | 1239245.6 | 10 | 832 | 17575.0 |
| 10 | 2 | 10 | 18.2 | 165837 | 4 | 604 | 585009.0 | 9 | 1048 | 17557.8 |
| 10 | 2 | 11 | 63.0 | 45071 | 7 | 637 | 958325.9 | 10 | 796 | 17515.3 |
| 10 | 2 | 12 | 1.2 | 2141 | 7 | 476 | 544253.6 | 10 | 734 | 17574.8 |
| 10 | 2 | 13 | 2.0 | 4002 | 4 | 391 | 378688.0 | 10 | 793 | 17574.0 |
| 10 | 2 | 14 | 1.0 | 85 | 9 | 391 | 376638.7 | 10 | 578 | 17575.0 |
| 10 | 2 | 15 | 1.0 | 39 | 7 | 715 | 798622.9 | 10 | 755 | 17575.0 |
| 10 | 2 | 16 | 1.0 | 32 | 9 | 352 | 349036.6 | 10 | 618 | 17575.0 |
| 10 | 3 | 9 | 7.6 | 240639 | 6 | 437 | 338330.8 | 9 | 907 | 17573.2 |
| 10 | 3 | 10 | 27.7 | 192315 | 2 | 451 | 659440.0 | 10 | 1267 | 17551.4 |
| 10 | 3 | 11 | 1.1 | 10001 | 5 | 719 | 987604.6 | 10 | 959 | 17574.9 |
| 10 | 3 | 12 | 2.1 | 19165 | 4 | 560 | 679398.8 | 10 | 916 | 17573.9 |
| 10 | 3 | 13 | 1.0 | 341 | 7 | 341 | 331790.3 | 10 | 696 | 17575.0 |
| 10 | 3 | 14 | 1.0 | 127 | 9 | 390 | 359362.0 | 10 | 534 | 17575.0 |
| 10 | 3 | 15 | 1.0 | 486 | 8 | 358 | 342439.2 | 10 | 528 | 17575.0 |
| 10 | 3 | 16 | 1.0 | 31 | 9 | 524 | 513744.6 | 10 | 612 | 17575.0 |

**Table 2** Results of `Choco` and `Choco+SAC` on a range of cribs.

We selected 260 cribs, so that we uniformly cover a range of values for the following parameters: *size of the menu* (size), i.e., the number of vertices in the graph of the extracted menu and *number of cycles* (#cycle). The idea is that sparse or acyclic menus produce more stops. In particular Turing made an analysis of how many stops would occur according to the values of these parameters [2]. Therefore, higher number of cycles and lower menu size should make the problem easier. Moreover, we ignored menus with four or more connected components. We average the results on instances with same size and number of cycles.

All experiments were run on 4 cluster nodes, with Intel Xeon CPU E5-2695 v4 2.10GHz cores running Linux Ubuntu 16.04.4. We ran both models for one hour or until completion on every instance. We report in Table 2 the characteristics of the cribs, the average number of solutions (#sol.), and the average number of solutions with distinct rotor positions, that is, the number of "stops" (#stops). Then, for both methods, we report the number of instances solved (#solved), that is, where all solutions have been enumerated within the one hour cutoff, the average CPU time (in seconds) over solved instances (CPU) and the average number of fails during search (#fail). We can observe that using $SAC$ significantly improves the model: `Choco` solves about 65% of the instances in less than one hour, whereas `Choco+SAC` solves 99% of the instances in about 15 minutes in average. The number of fails of `Choco+SAC` shows clearly that, as expected, constraint propagation does not actually cut the search tree for the rotor positions. All of the 17576 possible keys are explored. However,

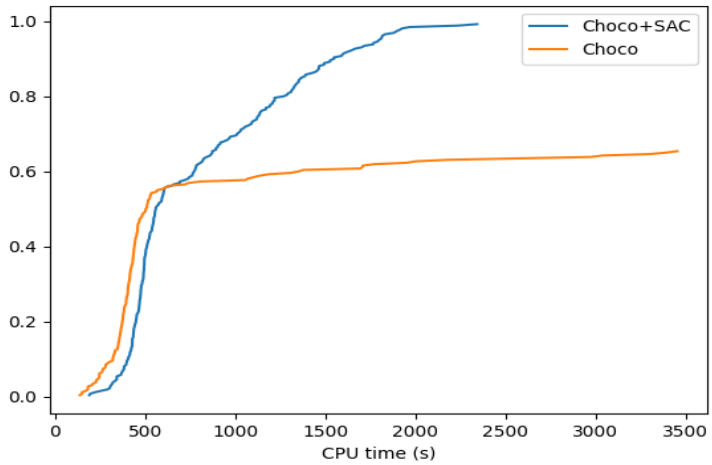



**Figure 5** Cumulative probability to break the code in less than $X$ seconds

it also shows that in the few cases where $SAC$ does not fail (the stops), virtually every singleton arc consistent permutation of the plugboard is consistent with the crib. Indeed the solver does not fail, and the number of solutions may become relatively large in that case. On the other hand, the number of fails of `Choco` shows a more conventional picture where plugoard configurations are ruled out by a blend of propagation and search.

The number of stops is lower than we would expect of the Bombe by about one or two orders of magnitude for low number of cycles. This is because we count only rotor positions for which a consistent plugboard configuration exists, whereas the Bombe stops as soon as a (slightly weaker form of) $SAC$ can be enforced. Moreover, enforcing $SAC$ on the constraint model takes advantage of propagation of the constraints modeling the plugboard (e.g., the AllDifferent constraint). Finally, the Bombe only takes into account the letters of the menu, i.e., it does not check that these extra letters too must have a legal plug connection.

Interestingly, we observe that the number of solutions tends to be larger for denser and more cyclic menus, even though the number of consistent rotor positions decreases, as we would expect from the Bombe. It shows that there are many more consistent plugboard configurations in this case. Overall, those parameters have a lower impact on the constraint model as they seem to have had on the Bombe. Overall, the method `Choco+SAC` is not extremely fast, but it is very robust. The graph in Figure 5 shows that in the most favorable cases, not enforcing singleton arc consistency can be the most efficient approach.

## 8 Conclusion

We have shown that the method designed by Alan Turing and Gordon Welchman at Bletchley Park to break the Enigma code has uncanny similarities with applying Singleton Arc Consistency on a constraint satisfaction problem modeling the machine. Experiments show that indeed, Singleton Arc Consistency significantly reduces the computation time required to decipher a message. Moreover, the method designed by Marian Rejewski et al. before that can also be related to achieving Path Consistency on another constraint satisfaction problem.

## References


1. Chris Christensen. Polish Mathematicians Finding Patterns in Enigma Messages. *Mathematics Magazine*, 80(4):247–273, 2007. URL: http://www.jstor.org/stable/27643040.
2. Cipher A. Deavours and Louis Kruh. The Turing Bombe: Was it Enough? *Cryptologia*, 14(4):331–349, 1990.
3. Romuald Debruyne and Christian Bessiere. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.
4. Pascal Van Hentenryck and Jean-Philippe Carillon. Generality versus Specificity: An Experience with AI and OR Techniques. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI)*, 1988.
5. Wladyslaw Kozaczuk. *Enigma. How the German Machine Cipher Was Broken, and how It Was Read by the Allies in World War II*. University Publications of America, 1984.
6. Alan K. Mackworth. Consistency in Networks of Relations. In Bonnie Lynn Webber and Nils J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 69–78. Morgan Kaufmann, 1981. URL: https://www.sciencedirect.com/science/article/pii/B978093461303350009X, doi:https://doi.org/10.1016/B978-0-934613-03-3.50009-X.
7. Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974. URL: https://www.sciencedirect.com/science/article/pii/0020025574900085, doi:https://doi.org/10.1016/0020-0255(74)90008-5.
8. Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. URL: http://www.choco-solver.org.
9. Wikipedia contributors. Enigma rotor details, 2021. URL: https://en.wikipedia.org/wiki/Enigma_rotor_details.

# Beyond Pairwise Comparisons in Social Choice: A Setwise Kemeny Aggregation Problem*

Hugo Gilbert[a], Tom Portoleau[b], Olivier Spanjaard[c,*]

[a] *Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016 Paris, France*
[b] *LAAS-CNRS, IRIT-CNRS, Université de Toulouse, Toulouse, France*
[c] *Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, Paris, France*

**Abstract**

In this paper, we advocate the use of setwise contests for aggregating a set of input rankings into an output ranking. We propose a generalization of the Kemeny rule where one minimizes the number of $k$-wise disagreements instead of pairwise disagreements (one counts 1 disagreement each time the top choice in a subset of alternatives of cardinality at most $k$ differs between an input ranking and the output ranking). After an algorithmic study of this $k$-wise Kemeny aggregation problem, we introduce a $k$-wise counterpart of the majority graph. This graph reveals useful to divide the aggregation problem into several subproblems, which enables to speed up the exact computation of a consensus ranking. By introducing a $k$-wise counterpart of the Spearman distance, we also provide a 2-approximation algorithm for the $k$-wise Kemeny aggregation problem. We conclude with numerical tests.

*Keywords:* Computational social choice, Generalized Kemeny rule, Weighted majority graph, Computational complexity

## 1. Introduction

Rank aggregation aims at producing a single ranking from a collection of rankings of a fixed set of alternatives. In social choice theory (see, for example, the book by Moulin [2]), where the alternatives are candidates to an election and each ranking represents the preferences of a voter, aggregation rules are called *Social Welfare Functions* (SWFs). Apart

*This is a long version of a work presented at the 34th AAAI Conference on Artificial Intelligence (AAAI 2020) [1].

*Corresponding author.

*Email addresses:* hugo.gilbert@dauphine.psl.eu (Hugo Gilbert), tom.portoleau@laas.fr (Tom Portoleau), olivier.spanjaard@lip6.fr (Olivier Spanjaard)

from social choice, rank aggregation has proved useful in many applications, including preference learning [3, 4], collaborative filtering [5], genetic map creation [6], similarity search in database systems [7] and design of web search engines [8, 9]. In the following, we use interchangeably the terms "input rankings" and "preferences", "output ranking" and "consensus ranking", as well as "alternatives" and "candidates".

The well-known Arrow's impossibility theorem states that there exists no aggregation rule satisfying a small set of desirable properties [10]. In the absense of an "ideal" rule, various aggregation rules have been proposed and studied. Following Fishburn's classification [11], we can distinguish between the SWFs for which the output ranking can be computed from the *majority graph* alone, those for which the output ranking can be computed from the *weighted majority graph* alone, and all other SWFs. The majority graph is obtained from the input rankings by defining one vertex per alternative $c$ and by adding an edge from $c$ to $c'$ if $c$ is preferred to $c'$ in a strict majority of input rankings. In the weighted majority graph, each edge is weighted by the majority margin. The many SWFs that rely on these graphs alone take therefore only pairwise comparisons into account to determine an output ranking. Note that Fishburn's classification actually applies to social *choice* functions, which prescribe a subset of winning alternatives from a collection of rankings, but the extension to SWFs is straightforward. For a compendium of SWFs that rely solely on (weighted) majority graphs, we refer the reader to the book chapter by Zwicker [12].

The importance of this class of SWFs can be explained by their connection with the *Condorcet consistency* property, stating: if there is a Condorcet winner (i.e., an alternative with outgoing edges to every other ones in the majority graph), then it should be ranked first in the output ranking. Nevertheless, as shown by Baldiga and Green [13], the lack of Condorcet consistency is not necessarily a bad thing, because this property may come into contradiction with the objective of maximizing voters' agreement with the output ranking. The following example illustrates this point.

**Example 1** (Baldiga and Green [13])**.** *Consider an election with 100 voters and 3 candidates $c_1, c_2, c_3$, where 49 voters have preferences $c_1 \succ c_2 \succ c_3$, 48 have preferences $c_3 \succ c_2 \succ c_1$ and 3 have preferences $c_2 \succ c_3 \succ c_1$. Candidate $c_2$ is the Condorcet winner, but is the top choice of only 3 voters. In contrast, candidate $c_1$ is in slight minority against $c_2$ and $c_3$, but $c_1$ is the top choice of 49 voters. This massive gain in agreement may justify to put $c_1$ instead of $c_2$ in first position of the output ranking.*

Following Baldiga and Green [13], we propose to handle this tension between the pairwise comparisons (leading to ranking $c_2$ first) and the plurality choice (leading to ranking $c_1$ first) by using SWFs that take into account not only pairwise comparisons but *setwise* contests. More precisely, given input rankings on a set $C$ of candidates and $k \in \{2, \ldots, |C|\}$, the idea is to consider the plurality score of each candidate $c$ for each subset $S \subseteq C$ such that

Table 1: Results of setwise contests in Example 1. The cell at the intersection of the row corresponding to a set $S$ and the column corresponding to a candidate $c$ displays the number of voters who rank $c$ first in $S$.

| set | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|
| $\{c_1, c_2\}$ | 49 | 51 | – |
| $\{c_1, c_3\}$ | 49 | – | 51 |
| $\{c_2, c_3\}$ | – | 52 | 48 |
| $\{c_1, c_2, c_3\}$ | 49 | 3 | 48 |

$2 \leq |S| \leq k$, where the plurality score of $c$ for $S$ is the number of voters for which $c$ is the top choice in $S$. The results of setwise contests for the preferences of Example 1 are given in Table 1 for $k=3$. Note that the three top rows obviously encode the same information as the weighted majority graph while the bottom row makes it possible to detect the tension between the pairwise comparisons and the plurality choice.

One can then define a new class of SWFs, those that rely on the results of setwise contests alone to determine an output ranking. The many works that have been carried out regarding voting rules based on the (weighted) majority graph can be revisited in this broader setting. This line of research has already been investigated by Lu and Boutilier [14] and Baldiga and Green [13]. However, note that both of these works consider a setting where candidates may become unavailable after voters express their preferences. We do not make this assumption. We indeed believe that this new class of SWFs makes sense in the standard setting where the set of candidates is known and deterministic, as it amounts to generate an output ranking by examining the choices that are made by the voters on subsets of candidates of various sizes (while usually only pairwise choices are considered).

A natural SWF in this class consists in determining an output ranking that minimizes the number of disagreements with the results of setwise contests for sets of cardinality at most $k$. This is a $k$-wise generalization of the Kemeny rule, obtained as a special case for $k=2$. We recall that the Kemeny rule consists in producing a ranking that minimizes the number of *pairwise* disagreements [15].

**Example 2.** *Let us come back to Example 1 and assume that we use the 3-wise Kemeny rule. Consider the output ranking* $r = c_1 \succ c_2 \succ c_3$. *For set* $S = \{c_1, c_2\}$, *the number of disagreements with the results of setwise contests is* 51 *because* $c_2$ *is the top choice in* $S$ *for* 51 *voters (see Table 1) while it is* $c_1$ *for* $r$. *Similarly, the number of disagreements induced by* $\{c_1, c_3\}$, $\{c_2, c_3\}$ *and* $\{c_1, c_2, c_3\}$ *are respectively* 51, 48 *and* $3+48$. *The total number of disagreements is thus* $51+51+48+3+48=201$. *This is actually the minimum number of disagreements that can be achieved for these input rankings, which makes* $r$ *the* $k$*-wise Kemeny ranking.*

The purpose of this paper is to study the $k$-wise Kemeny aggregation problem. Section 2

formally defines the problem and reports on related work. Section 3 is devoted to some axiomatic considerations of the corresponding voting rule, and to an algorithmic study of the problem. More precisely, we show that the decision variant of the $k$-wise Kemeny aggregation problem is NP-complete for any constant $k \geq 2$ and we provide an efficient fixed-parameter algorithm for parameter $m$ which relies on dynamic programming. We then investigate a $k$-wise variant of the majority graph in Section 4. We prove that determining this graph is easy for $k = 3$ but becomes NP-hard for $k > 3$, and we show how to use it in a preprocessing step to speed up the computation of the output ranking. In Section 5, we propose a 2-approximation algorithm for the $k$-wise Kemeny aggregation problem, by introducing a $k$-wise variant of the Spearman distance. Numerical tests are presented in Section 6 to assess both the efficiency of our exact methods and the accuracy of our approximation algorithm.

## 2. Preliminaries

Adopting the terminology of social choice theory, we consider an election with a set $V$ of $n$ voters and a set $C$ of $m$ candidates. Each voter $v$ has a complete and transitive preference order $r_v$ over candidates (also called ranking). The collection of these rankings defines a preference profile $\mathcal{P}$.

### *2.1. Notations and Definitions*

Let us introduce some notations related to rankings. We denote by $\mathcal{R}(C)$ the set of $m!$ rankings over $C$. Given a ranking $r$ and two candidates $c$ and $c'$, we write $c \succ_r c'$ if $c$ is in a higher position than $c'$ in $r$. Given a ranking $r$ and a candidate $c$, $\mathtt{rk}(c, r)$ denotes the rank of $c$ in $r$. For instance, $\mathtt{rk}(c, r_v) = 1$ if $c$ is the preferred candidate of voter $v$ (the candidate ranked highest in $r_v$). Given a ranking $r$ and a set $S \subseteq C$, we define $r_S$ as the restriction of $r$ to $S$ and $\mathtt{top}_r(S)$ as the top choice (i.e., preferred candidate) in $S$ according to $r$. Similarly, given a preference profile $\mathcal{P}$ and a set $S \subseteq C$, we define $\mathcal{P}_S$ as the restriction of $\mathcal{P}$ to $S$. Lastly, we denote by $\mathtt{tail}_k(r)$ (resp. $\mathtt{head}_k(r)$) the subranking compounded of the $k$ least (resp. most) preferred candidates in $r$.

We are interested in SWFs which, given a preference profile $\mathcal{P}$, should return a consensus ranking which yields a suitable compromise between the preferences in $\mathcal{P}$. One of the most well-known SWFs is the *Kemeny rule*, which selects a ranking $r$ with minimal Kendall tau distance to $\mathcal{P}$. We recall:

**Definition 1.** *The Kendall tau distance between two rankings $r$ and $r'$ is defined by*

$$\delta_{\mathrm{KT}}(r, r') = \sum_{(c,c') \in C^2} \mathit{disagree}_{c,c'}(r, r')$$

*where* $\texttt{disagree}_{c,c'}(r,r') = 1$ *if* $c \succ_r c'$ *and* $c' \succ_{r'} c$*, and 0 otherwise.*

Stated differently, $\delta_{\texttt{KT}}$ measures the distance between two rankings by the number of pairwise disagreements between them. The distance $\delta_{\texttt{KT}}(r, \mathcal{P})$ between a ranking $r$ and a preference profile $\mathcal{P}$ is then obtained by summation:

$$\delta_{\texttt{KT}}(r, \mathcal{P}) = \sum_{r' \in \mathcal{P}} \delta_{\texttt{KT}}(r, r').$$

However, the Kendall tau distance only takes into account pairwise comparisons, which may entail counterintuitive results as illustrated by Example 1. To address this issue, the Kendall tau distance can be generalized to take into consideration disagreements on sets of cardinality greater than two. Given a set $S \subseteq C$ and $t \leq m$, we denote by $\Delta^t(S)$ the set of subsets of $S$ of cardinality lower than or equal to $t$, i.e., $\Delta^t(S) = \{S' \subseteq S \text{ s.t. } |S'| \leq t\}$. When $S$ is not specified, it is assumed to be $C$, i.e., $\Delta^t = \Delta^t(C)$.

**Definition 2.** *Let* $k \geq 2$ *be an integer. The k-wise Kendall tau distance* $\delta^k_{\texttt{KT}}$ *between* $r$ *and* $r'$ *is defined by:*

$$\delta^k_{\texttt{KT}}(r, r') = \sum_{S \in \Delta^k} \texttt{disagree}_S(r, r')$$

*where* $\texttt{disagree}_S(r,r') = 1$ *if* $\texttt{top}_r(S) \neq \texttt{top}_{r'}(S)$*, and 0 otherwise.*

In other words, $\delta^k_{\texttt{KT}}$ measures the distance between two rankings by the number of top-choice disagreements on sets of cardinality lower than or equal to $k$. It is not hard to see that $\delta^k_{\texttt{KT}}(r, r')$ can also be computed by using the following formula:

$$\begin{aligned}\delta^k_{\texttt{KT}}(r, r') &= \sum_{(c,c') \in C^2} \texttt{disagree}_{c,c'}(r, r') \; |\Delta^{k-2}(\texttt{Below}_c(r) \cap \texttt{Below}_{c'}(r')| \\ &= \sum_{(c,c') \in C^2} \texttt{disagree}_{c,c'}(r, r') \sum_{i=0}^{k-2} \binom{|\texttt{Below}_c(r) \cap \texttt{Below}_{c'}(r')|}{i} \qquad (1)\end{aligned}$$

where $\texttt{Below}_c(r) = \{x \in C \text{ s.t. } c \succ_r x\}$ is the set of candidates that are ranked below $c$ in $r$. Formula 1 amounts to counting, for any pair $\{c, c'\}$ of candidates such that $c \succ_r c'$ and $c' \succ_{r'} c$, the number of sets in $\Delta^k$ on which there is a disagreement because the top choice is $c$ for $r$ while it is $c'$ for $r'$. Such sets are of the form $S \cup \{c, c'\}$, where $S \in \Delta^{k-2}(\texttt{Below}_c(r) \cap \texttt{Below}_{c'}(r'))$, otherwise $c$ and $c'$ would not be the top choices. Hence the formula.

Several observations can be made regarding $\delta^k_{\texttt{KT}}$. Firstly, the following result states that $\delta^k_{\texttt{KT}}$ has all the properties of a distance:

**Proposition 1.** *The function $\delta^k_{\mathtt{KT}}$ has all the properties of a distance: non-negativity, identity of indiscernibles, symmetry and triangle inequality.*

*Proof.* Non-negativity and symmetry are obvious from the definition of $\delta^k_{\mathtt{KT}}$. It also verifies identity of indiscernibles: if $\delta^k_{\mathtt{KT}}(r,r')=0$, then rankings $r$ and $r'$ must in particular agree on each pairwise comparison, hence $\delta_{\mathtt{KT}}(r,r')=0$ and $r=r'$ because $\delta_{\mathtt{KT}}$ verifies identity of indiscernibles. Lastly, triangular inequality comes from the fact that given three rankings $r_1$, $r_2$ and $r_3$ and a set $S$, $\mathtt{disagree}_S(r_1,r_3) \leq \mathtt{disagree}_S(r_1,r_2) + \mathtt{disagree}_S(r_2,r_3)$. □

Secondly, as mentioned in the introduction, we have $\delta^2_{\mathtt{KT}}=\delta_{\mathtt{KT}}$. Thirdly and maybe most importantly, $\delta^k_{\mathtt{KT}}(r,r')$ can be computed in polynomial time in the number $m$ of candidates:

**Proposition 2.** *Given two rankings $r$ and $r'$, $\delta^k_{\mathtt{KT}}(r,r')$ can be computed in $O(m^3)$ by using Formula 1.*

*Proof.* We prove the $O(m^3)$ complexity of the method. First note that the computation of all binomial coefficients $\binom{p}{i}$ for $i \in \{0,\dots,k-2\}$ and $p \in \{i,\dots,m-2\}$ can be performed in $O(mk)$ thanks to Pascal's formula $\binom{p}{i}+\binom{p}{i+1}=\binom{p+1}{i+1}$. Then the computation of the sums $\sum_{i=0}^{k-2}\binom{p}{i}$ for $p \in \{0,\dots,m-2\}$ can also be computed in $O(mk)$. For each pair $\{c,c'\}$ of candidates such that $\mathtt{top}_r(\{c,c'\}) = c$ and $\mathtt{top}_{r'}(\{c,c'\}) = c'$, the computation of $|\,\mathtt{Below}_c(r) \cap \mathtt{Below}_{c'}(r')|$ can be performed in $O(m)$. As there are at most $O(m^2)$ such pairs, the overall complexity of the method is $O(m^3+mk)=O(m^3)$. □

The distance $\delta^k_{\mathtt{KT}}$ induces a new SWF, the $k$-wise Kemeny rule, which, given a profile $\mathcal{P}$, returns a ranking $r$ with minimal distance $\delta^k_{\mathtt{KT}}$ to $\mathcal{P}$, where:

$$\delta^k_{\mathtt{KT}}(r,\mathcal{P}) = \sum_{r'\in\mathcal{P}} \delta^k_{\mathtt{KT}}(r,r').$$

Note that this coincides with the rule we used in the introduction, by commutativity of addition:

$$\sum_{r'\in\mathcal{P}}\sum_{S\in\Delta^k} \mathtt{disagree}_S(r,r') = \sum_{S\in\Delta^k}\sum_{r'\in\mathcal{P}} \mathtt{disagree}_S(r,r').$$

Determining a consensus ranking for this rule induces an optimization problem that we term the $k$-wise Kemeny Aggregation Problem ($k$-KAP for short).

| $k$-**WISE KEMENY AGGREGATION PROBLEM ($k$-KAP)** |
|---|
| *INSTANCE:* A profile $\mathcal{P}$ with $n$ voters and $m$ candidates. |
| *SOLUTION:* A ranking $r$ of the $m$ candidates minimizing $\delta^k_{\mathtt{KT}}(r,\mathcal{P})$. |

When discussing the complexity of the problem, we may also refer to the decision version of the problem, $k$-KAP-DEC:

| $k$-**KAP-DEC** |
| --- |
| *INSTANCE:* A profile $\mathcal{P}$ with $n$ voters and $m$ candidates and a threshold $\tau \in \mathbb{N}$. |
| *QUESTION:* Does there exist a ranking $r$ of the $m$ candidates such that $\delta^k_{\mathtt{KT}}(r, \mathcal{P}) \leq \tau$? |

### *2.2. Related Work*

Several other variants of the Kemeny rule have been proposed in the literature, either to obtain generalizations able to deal with partial or weak orders [9, 16], to penalize more some pairwise disagreements than others [17], or to account for candidates that may become unavailable after voters express their preferences [13, 14].

Despite its popularity, the Kemeny rule has received several criticisms. One of them is that the Kendall tau distance counts equally the disagreements on every pair of candidates. This property is undesirable in many settings. For instance, with a web search engine, a disagreement on a pair of web pages with high positions in the considered rankings should have a higher cost than a disagreement on pairs of web pages with lower ones. This drawback motivated the introduction of weighted Kendall tau distances by Kumar and Vassilvitskii [17].

#### *2.2.1. Comparison between the k-wise and the weighted Kendall tau distances*

As mentioned above, Kumar and Vassilvitskii proposed that disagreements on highly ranked candidates be more costly than disagreements on lowly ranked ones. To achieve this, they defined a position-weighted version of Kendall tau, denoted by $K_w$, where an inversion of the two candidates at positions $i$ and $i-1$ has a cost $w_i$. For convenience, a cost $w_1 = 1$ is also defined. Given the costs $w_i$, one can then measure the average swap-cost of moving a candidate from position $i$ to $j$ by computing the ratio $\frac{p_i - p_j}{i-j}$ where $p_i = \sum_{j=1}^{i} w_j$. This observation motivated the following definition for a position-weighted version of Kendall tau [17]:

**Definition 3.** *Let $w = (w_1, \ldots, w_m)$ be a vector of $m$ weights $w_i > 0$. The position-weighted Kendall tau distance $K_w$ between $r$ and $r'$ is defined by:*

$$K_w(r, r') = \sum_{(c,c') \in C^2} \mathtt{disagree}_{c,c'}(r, r')\, \overline{p}(r, r', c) \overline{p}(r, r', c')$$

*where* $\overline{p}(r, r', c) = \dfrac{p_{\mathtt{rk}(c,r)} - p_{\mathtt{rk}(c,r')}}{\mathtt{rk}(c, r) - \mathtt{rk}(c, r')}$ *and* $\overline{p}(r, r', c) = 1$ *if* $\mathtt{rk}(c, r) = \mathtt{rk}(c, r')$.

Put another way, if two rankings $r$ and $r'$ disagree on a pairwise comparison between two candidates $c$ and $c'$, then the cost of this disagreement is weighted by the product of the average swap-cost of moving $c$ from its position in $r$ to its position in $r'$ with the average swap-cost of moving $c'$ from its position in $r$ to its position in $r'$. Note that if $w_i = 1$ for all $i$ in $\{1, \ldots, m\}$, one obtains the usual Kendall tau distance.

Both the position-weighted Kendall tau distance and the $k$-wise Kendall tau distance can be used in order to penalize more strongly disagreements on candidates with high ranks (i.e., candidates that appear near the top of the ranking). For the $k$-wise Kendall tau distance, this property results from the fact that for a pair $\{c, c'\}$ such that $\texttt{top}_r(\{c, c'\}) \neq \texttt{top}_{r'}(\{c, c'\})$ the number of resulting subsets $S$ for which $\texttt{top}_r(S) \neq \texttt{top}_{r'}(S)$ is all the larger as $c$ and $c'$ are ranked high in $r$ and $r'$. Note however that the position-weighted Kendall tau distance requires to specify the $m-1$ parameters $w_2, \ldots, w_m$, the tuning of which does not seem to be obvious. In comparison, the $k$-wise Kendall tau distance only requires to choose the value of $k$, from which the cost of each disagreement on a pair $\{c, c'\}$ of candidates is naturally entailed: it corresponds to the number of subsets of $C$ of size less than or equal to $k$ for which the top choice in $r$ is $c$ while the top choice in $r'$ is $c'$ (see Section 2, Equation 1 for the formal expression of swap-costs according to $k$).

Let us illustrate with the following example that the $k$-wise Kendall tau distance is also well suited to penalize more the disagreements involving alternatives at the top of the input rankings.

**Example 3.** *Consider rankings $r_1, r_2, r_3$ defined by $c_1 \succ_{r_1} c_2 \succ_{r_1} c_3$, $c_1 \succ_{r_2} c_3 \succ_{r_2} c_2$, and $c_2 \succ_{r_3} c_1 \succ_{r_3} c_3$. We have $\delta_{\texttt{KT}}(r_1, r_2) = \delta_{\texttt{KT}}(r_1, r_3) = 1$ while $\delta^3_{\texttt{KT}}(r_1, r_2) = 1 < 2 = \delta^3_{\texttt{KT}}(r_1, r_3)$ because $r_1$ and $r_3$ disagree on both subsets $\{c_1, c_2\}$ and $\{c_1, c_2, c_3\}$. Put another way, $\delta^3_{\texttt{KT}}(r_1, r_3) > \delta^3_{\texttt{KT}}(r_1, r_2)$ because $r_1$ and $r_3$ disagree on their top-ranked alternatives whereas $r_1$ and $r_2$ disagree on the alternatives ranked in the last places.*

### *2.3. Comparison with aggregation models where candidates may become unavailable*

The two works closest to ours are related to another extension of the Kemeny rule. This extension considers a setting in which, besides the fact that voters have preferences over a set $C$, the election will in fact occur on a subset $S \subseteq C$ drawn according to a probability distribution [13, 14]. The optimization problem considered is then to find a consensus ranking $r$ which minimizes, in expectation, the number of voters' disagreements with the chosen candidate in $S$ (a voter $v$ disagrees if $\texttt{top}_{r_v}(S) \neq \texttt{top}_r(S)$). The differences between the work of Baldiga and Green [13] and the one of Lu and Boutilier [14] are then twofold. Firstly, while Baldiga and Green mostly focused on the axiomatic properties of this aggregation procedure, the work of Lu and Boutilier has more of an algorithmic flavor. Secondly, while Baldiga and Green mostly study a setting in which the probability $\mathbb{P}(S)$ of $S$ is only dependent on its cardinality (i.e., $\mathbb{P}(S)$ is only a function of $|S|$), Lu and Boutilier

study a setting that can be viewed as a special case of the former, where each candidate is absent of $S$ independently of the others with a probability $p$ (i.e., $\mathbb{P}(S) = p^{|C \setminus S|}(1-p)^{|S|}$). The Kemeny aggregation problem can be formulated in both settings, either by defining $\mathbb{P}(S) = 0$ for $|S| \geq 3$, or by defining a probability $p$ that is "sufficiently high" w.r.t. the size of the instance [14]. Lu and Boutilier conjectured that the determination of a consensus ranking is NP-hard in their setting, designed an exact method based on mathematical programming, two approximation greedy algorithms and a polynomial-time approximation scheme.

Our model can be seen as a special case of the model of Baldiga and Green where the set $S$ is drawn uniformly at random within the set of subsets of $C$ of cardinality smaller than or equal to a given constant $k \geq 2$. While it cannot be casted in the specific setting studied by Lu and Boutilier, our model is closely related and may be used to obtain new insights on their work.

## 3. Aggregation with the *k*-wise Kemeny Rule

In this section, we investigate the axiomatic properties of the $k$-wise Kemeny rule, and then we turn to the algorithmic study of $k$-KAP.

### *3.1. Axiomatic Properties of the k-wise Kemeny Rule*

Several properties of the $k$-wise Kemeny rule have already been studied by Baldiga and Green [13], because their setting includes the $k$-wise Kemeny rule as a special case. Among other things, they showed that the rule is not *Condorcet consistent*. That is to say, a Condorcet winner may not be ranked first in any consensus ranking even when one exists, as illustrated by Example 2. The example indeed shows that the $k$-wise Kemeny rule is not Condorcet consistent for $k = 3$, as the Condorcet winner is not ranked first in the unique consensus ranking for the 3-wise Kemeny rule. Note that any $k > 3$ would yield the same consensus ranking, as there are only $m = 3$ candidates in the profile, hence the result holds for any $k \geq 3$. The example can be generalized to show that the result holds even if $m \geq k$:

**Proposition 3.** *The $k$-wise Kemeny rule is not Condorcet consistent for any $k \geq 3$, even if $m \geq k$.*

*Proof.* We provide an example similar to Example 2. Consider an election with $k$ candidates $c_1, c_2, \ldots, c_k$ and 100 voters, with:
– 49 voters having preferences $r_1$ defined by $c_1 \succ c_2 \succ c_3 \succ c_4 \succ \ldots \succ c_k$;
– 49 voters having preferences $r_2$ defined by $c_3 \succ c_2 \succ c_1 \succ c_4 \succ \ldots \succ c_k$;
– 2 voters having preferences $r_3$ defined by $c_2 \succ c_1 \succ c_3 \succ c_4 \succ \ldots \succ c_k$.
Note that $c_2$ is a Condorcet winner and that the ranking $r_1$ has a $k$-wise Kemeny score

worth $0 + 49 \times (2^{k-2} + 2^{k-3} + 2^{k-3}) + 2 \times 2^{k-2} = 100 \times 2^{k-2}$ while ranking $r_3$ has a score worth $49 \times 2^{k-2} + 49 \times (2^{k-2} + 2^{k-3}) + 0 = 122.5 \times 2^{k-2}$. It is easy to convince oneself that $r_3$ is the best possible ranking if $c_2$ is ranked first. Indeed, candidates $c_1$, $c_2$ and $c_3$ are ranked higher than other candidates in all three rankings $r_1$, $r_2$ and $r_3$ and therefore should be placed at the top in a consensus ranking (see Lemma 1). Additionally, $c_1$ and $c_3$ have symmetric positions in rankings $r_1$ and $r_2$, and $c_1$ is preferred to $c_3$ in $r_3$, thus $c_1$ should be placed before $c_3$ in a consensus ranking. The result follows. □

The authors also show that the $k$-wise Kemeny rule is neutral, i.e., all candidates are treated equally, and that for $k \geq 3$ it is different from any positional method or any method that uses only the pairwise majority margins (among which is the standard Kemeny rule). We provide here some additional properties satisfied by the $k$-wise Kemeny rule:

- *Monotonicity*: up-ranking cannot harm a winner; down-ranking cannot enable a loser to win. Let us state this axiom more formally. Let $\mathcal{R}^*_{\mathcal{P}}$ denote the set of consensus rankings for preference profile $\mathcal{P}$. Then for any candidate $c \in C$ and profiles $\mathcal{P}$ and $\mathcal{P}'$ such that $\mathcal{P}'$ can be obtained from $\mathcal{P}$ by decreasing the position of $c$ in some ranking in $\mathcal{P}$ (all other things being equal): $c \in \{\mathtt{top}_r(C) : r \in \mathcal{R}^*_{\mathcal{P}'}\}$ implies $c \in \{\mathtt{top}_r(C) : r \in \mathcal{R}^*_{\mathcal{P}}\}$ and $c \notin \{\mathtt{top}_r(C) : r \in \mathcal{R}^*_{\mathcal{P}}\}$ implies $c \notin \{\mathtt{top}_r(C) : r \in \mathcal{R}^*_{\mathcal{P}'}\}$.

- *Unanimity*: if all voters rank $c$ before $c'$, then $c$ is ranked before $c'$ in any consensus ranking.

- *Reinforcement*: let $\mathcal{R}^*_{\mathcal{P}}$ and $\mathcal{R}^*_{\mathcal{P}'}$ denote the sets of consensus rankings for preference profiles $\mathcal{P}$ and $\mathcal{P}'$ respectively. If $\mathcal{R}^*_{\mathcal{P}} \cap \mathcal{R}^*_{\mathcal{P}'} \neq \emptyset$ and $\mathcal{P}''$ is the profile obtained by concatenating $\mathcal{P}$ and $\mathcal{P}'$, then $\mathcal{R}^*_{\mathcal{P}''} = \mathcal{R}^*_{\mathcal{P}} \cap \mathcal{R}^*_{\mathcal{P}'}$.

Examples of voting rules for which the monotonicity property does not hold are *plurality with run-off* (a two-round election system where, if some candidate is top ranked by a majority of the voters, it wins in round 1; otherwise, round 2 consists of the majority rule applied to the two candidates with highest plurality score in round 1) and *single transferable vote* (at each stage, the candidate with lowest plurality score is dropped from all votes and each vote for which this candidate was top ranked is transferred to the next remaining candidate in the ranking; at the first stage for which some candidate $c$ sits atop a majority of the votes, $c$ is declared the winner). For more details, the reader may refer to the book chapter by Zwicker [12]. The unanimity property is obviously desirable for any reasonable voting rule. Finally, the reinforcement property has been introduced by Young [18], originally calling it *consistency*, for the axiomatization of Borda's rule viewed as a *social choice function*, i.e., returning a (set of) winning candidate(s). Reinforcement states that a candidate elected by two disjoint electorates should remain a winning candidate if one merges the voters,

and that a candidate elected by only one electorate is in some sense not as "good" as a candidate elected by both electorates. The adaptation of the property to *social welfare functions*, i.e., functions returning a (set of) consensus ranking(s), has been proposed by Young and Levenglick [19] for an axiomatic characterization of Kemeny's rule: they show that it is the unique Condorcet consistent social welfare function that satisfy neutrality and reinforcement, where the neutrality property states that all candidates are treated equally (in the sense that permuting the names of the candidates in the input rankings result in the same permutation in the consensus rankings).

While the fact that the $k$-wise Kemeny rule satisfies neutrality and reinforcement is quite obvious from its definition, the two following results state that the monotonicity and unanimity conditions also hold.

**Proposition 4.** *The $k$-wise Kemeny rule satisfies monotonicity.*

*Proof.* Let $r$ be a ranking and $v$ be a voter such that $c$ and $c'$ are consecutive in $r_v$ and $c' \succ_{r_v} c$. Let us denote by $r_v^{c \leftrightarrow c'}$ the ranking obtained from $r_v$ by switching the positions of $c$ and $c'$. Then, the sets $S$ for which $\mathtt{top}_{r_v}(S) \neq \mathtt{top}_{r_v^{c \leftrightarrow c'}}(S)$ are of the form $\{c, c'\} \cup S'$ where $S' \subseteq \mathtt{Below}_c(r_v)$. Furthermore, if such a set $S$ contains a candidate $c''$ such that $c'' \succ_r \mathtt{top}_r(\{c, c'\})$, then we will both have $\mathtt{top}_r(S) \neq \mathtt{top}_{r_v}(S)$ and $\mathtt{top}_r(S) \neq t_{r_v^{c \leftrightarrow c'}}(S)$. Hence, the sets which account for the difference between $\delta_{\mathtt{KT}}^k(r, r_v)$ and $\delta_{\mathtt{KT}}^k(r, r_v^{c \leftrightarrow c'})$ are of the form $\{c, c'\} \cup S'$ where $S' \subseteq \mathtt{Below}_c(r_v) \cap \mathtt{Below}_{\mathtt{top}_r(\{c,c'\})}(r)$.

More precisely, using Equation 1, we obtain that $\delta_{\mathtt{KT}}^k(r, r_v^{c \leftrightarrow c'})$ is equal to:

- $\delta_{\mathtt{KT}}^k(r, r_v) - \sum_{i=0}^{k-2} \binom{|\mathtt{Below}_{c'}(r_v) \cap \mathtt{Below}_c(r)|}{i}$ if $c \succ_r c'$;
- $\delta_{\mathtt{KT}}^k(r, r_v) + \sum_{i=0}^{k-2} \binom{|\mathtt{Below}_c(r_v) \cap \mathtt{Below}_{c'}(r)|}{i}$ if $c' \succ_r c$.

Hence, $\delta_{\mathtt{KT}}^k(r, r_v)$ will decrease if $r$ ranks $c$ before $c'$ and the decrease is maximal when $c$ is ranked first in $r$ (because it maximizes $|\mathtt{Below}_{c'}(r_v) \cap \mathtt{Below}_c(r)|$). Repeating this argument shows that no winner is harmed by up-ranking. Similarly, $\delta_{\mathtt{KT}}^k(r, r_v)$ will increase if $r$ ranks $c'$ before $c$ and the increase is maximal when $c'$ is ranked first in $r$ (because it maximizes $|\mathtt{Below}_c(r_v) \cap \mathtt{Below}_{c'}(r)|$). Repeating this argument shows that no loser can win by down-ranking. □

**Proposition 5.** *The $k$-wise Kemeny rule satisfies unanimity.*

*Proof.* Let $c, c' \in C$ be two candidates and $\mathcal{P}$ be a preference profile such that for all ranking $r''$ in $\mathcal{P}$, $c \succ_{r''} c'$. Let $r$ be a ranking such that $c' \succ_r c$, and $r'$ be the ranking obtained from $r$ by exchanging the positions of $c'$ and $c$. Moreover, let $K$ denote the set of candidates between $c$ and $c'$ in $r$. Let us assume for the sake of contradiction that $r$ minimizes $\delta_{\mathtt{KT}}^k(\cdot, \mathcal{P})$.

We will prove that for any $r'' \in \mathcal{P}$, $\delta^k_{\mathtt{KT}}(r', r'') < \delta^k_{\mathtt{KT}}(r, r'')$. Let $r'' \in \mathcal{P}$ and $S \subseteq C$ such that $\mathtt{top}_{r'}(S) \neq \mathtt{top}_{r''}(S)$ and $\mathtt{top}_r(S) = \mathtt{top}_{r''}(S)$. Then $S$ must contain either $c$ or $c'$, and does not contain any element ranked higher than $c'$ in $r$ because otherwise we would have $\mathtt{top}_r(S) = \mathtt{top}_{r'}(S)$. This implies that either $\mathtt{top}_{r'}(S) = c$ (if $c \in S$) or $\mathtt{top}_r(S) = c'$ (if $c' \in S$). These situations are exclusive: there cannot be both $c$ and $c'$ in $S$ as we cannot have $\mathtt{top}_{r''}(S) = c'$ if $c \in S$. To sum up, there are two possibilities:

1. $\mathtt{top}_{r'}(S) = c$ and $\mathtt{top}_{r''}(S) = \mathtt{top}_r(S) = c''$ is in $K$. This implies that $c''$ is the second choice of $r'$ in $S$ and that $c'' \succ_{r''} c'$ as $c'' \succ_{r''} c$. In this case, necessarily, $c' \notin S$ and we consider $S' = (S \setminus \{c\}) \cup \{c'\}$.
2. $\mathtt{top}_{r'}(S) \in K$ and $\mathtt{top}_{r''}(S) = \mathtt{top}_r(S) = c'$. In this case, necessarily, $c \notin S$ and we consider $S' = S \cup \{c\}$.

In both cases, we obtain a set $S'$ such that $\mathtt{top}_{r'}(S') = \mathtt{top}_{r''}(S')$ and $\mathtt{top}_r(S') \neq \mathtt{top}_{r''}(S')$. Note that any set $S$ will induce a different $S'$ and that $\{c, c'\}$ is not one of these sets $S'$. As we also have $\mathtt{top}_{r'}(\{c, c'\}) = \mathtt{top}_{r''}(\{c, c'\})$ and $\mathtt{top}_r(\{c, c'\}) \neq \mathtt{top}_{r''}(\{c, c'\})$, this proves that for any $r'' \in \mathcal{P}$, $\delta^k_{\mathtt{KT}}(r', r'') < \delta^k_{\mathtt{KT}}(r, r'')$ and hence the claim. □

Besides, the $k$-wise Kemeny rule does not satisfy *Independence of irrelevant alternatives*, i.e., the relative positions of two candidates in a consensus ranking can depend on the presence of other candidates. Let us illustrate this point with the following example.

**Example 4.** *Considering the preference profile from Example 1, the only consensus ranking for $\delta^3_{\mathtt{KT}}$ is $c_1 \succ c_2 \succ c_3$. Yet, without $c_3$ the only consensus ranking would be $c_2 \succ c_1$.*

Lastly, note that there exists a noise model such that the $k$-wise Kemeny rule can be interpreted as a maximum likelihood estimator [20]. In this view of voting, one assumes that there exists a "correct" ranking $r$, and each vote corresponds to a noisy perception of this correct ranking. Consider the conditional probability measure $\mathbb{P}$ on $\mathcal{R}(C)$ defined by $\mathbb{P}(r'|r) \propto e^{-\delta^k_{\mathtt{KT}}(r,r')}$. It is easy to convince oneself that the $k$-wise Kemeny rule returns a ranking $r^*$ that maximizes $\mathbb{P}(\mathcal{P}\,|r^*) = \prod_{r' \in \mathcal{P}} \mathbb{P}(r'|r^*)$ and is thus a maximum likelihood estimate of $r$.

### *3.2. Computational Complexity of k-KAP*

We now turn to the algorithmic study of $k$-KAP. After providing a hardness result, we will design an efficient Fixed Parameter Tractable (FPT) algorithm for parameter $m$.

While $k$-KAP is obviously NP-hard for $k = 2$ as it then corresponds to determining a consensus ranking w.r.t. the Kemeny rule, we strengthen this result by showing that $k$-KAP-DEC is also NP-complete for *any* constant value $k \geq 3$. To prove this result, we first need two lemmas.

**Lemma 1.** *If the candidates in a set $S \subseteq C$ are ranked in the $|S|$ last positions by all voters and in the same order, then for any $k \geq 2$, any consensus ranking w.r.t. the $k$-wise Kemeny rule has the same property.*

*Proof.* This is a simple consequence of the unanimity property that is satisfied by the $k$-wise Kemeny rule. □

**Lemma 2.** *For any $p \in \mathbb{N}^*$ and $\varepsilon \in (0, \frac{1}{2p})$ we have the following inequality:*

$$(1+\varepsilon)^p < 1 + 2p\varepsilon$$

*Proof.* We prove the claim by induction. It is obvious for $p=1$. Consider the claim true for $p=k$, then for $\varepsilon \in (0, \frac{1}{2(k+1)})$

$$\begin{aligned}(1+\varepsilon)^{k+1} &= (1+\varepsilon)(1+\varepsilon)^k < (1+\varepsilon)(1+2k\varepsilon) \\ &= 1+2k\varepsilon+\varepsilon+2k\varepsilon^2 < 1+2(k+1)\varepsilon\end{aligned}$$

where the first inequality uses the induction hypothesis and the second inequality uses the fact that $2k\varepsilon^2 < \varepsilon$ because $2k\varepsilon < 1$ for $\varepsilon \in (0, \frac{1}{2(k+1)})$. □

We can now prove the hardness result, by using a reduction from 2-KAP-DEC.

**Theorem 1.** *For any constant $k \geq 3$, $k$-KAP-DEC is NP-complete, even if the number of voters equals 4 or if the average range of candidates is less than or equal to 2 (where the range of a candidate $c$ is defined by $\max_{r \in \mathcal{P}} \mathtt{rk}(c,r) - \min_{r \in \mathcal{P}} \mathtt{rk}(c,r) + 1$ and the average is taken over all candidates).*

*Proof.* Membership in NP follows from Proposition 2. We obtain our hardness result via a reduction from the standard Kemeny aggregation problem (2-KAP-DEC), which is known to be NP-complete [21]. Consider a preference profile $\mathcal{P}$ with $n \geq 1$ voters and $m \geq k \geq 3$ candidates and an integer $\tau$. We wish to determine if there exists a ranking $r$ of the $m$ candidates such that $\delta_{\mathtt{KT}}(r, \mathcal{P}) \leq \tau$. Stated otherwise, we wish to determine if the Kemeny score of a consensus ranking is lower than or equal to $\tau$. Note that we can assume $m \geq k$ as $k$ is a constant and 2-KAP is fixed parameter tractable w.r.t. $m$ [22]. We add to the problem $\lambda = 4nm^4$ candidates $c_1^*, \ldots, c_\lambda^*$ that are ranked last by all voters and in the same order, i.e., $c_1^* \succ_r \ldots \succ_r c_\lambda^*$ for all $r$ in $\mathcal{P}$. We denote the resulting set of candidates by $C'$ (i.e., $C' = C \cup \{c_1^*, \ldots, c_\lambda^*\}$) and the resulting preference profile by $\mathcal{P}'$. By using Lemma 1, we will restrict our attention to rankings that rank these additional voters last and in the same order as the voters, because they are the only possible consensus rankings. Lastly, in the resulting $k$-KAP-DEC instance, we set $\tau' = (1 + \sum_{i=1}^{k-2} \binom{4nm^4}{i})(\tau+1) - 1$.

Given two such rankings $r$ and $r'$, we have by Equation 1 that $\delta^k_{\mathtt{KT}}(r,r')$ is equal to:

$$\sum_{(c,c')\in C^2} \mathtt{disagree}_{c,c'}(r,r') \sum_{i=0}^{k-2} \binom{|\,\mathtt{Below}_c(r_C)\cap \mathtt{Below}_{c'}(r'_C)|+\lambda}{i}$$
$$= \sum_{(c,c')\in C^2} \mathtt{disagree}_{c,c'}(r,r') \left(1+\sum_{i=1}^{k-2} \binom{|\,\mathtt{Below}_c(r_C)\cap \mathtt{Below}_{c'}(r'_C)|+\lambda}{i}\right)$$

because there is no disagreement for $\{c,c'\}\not\subseteq C$ and $\binom{|\,\mathtt{Below}_c(r_C)\cap \mathtt{Below}_{c'}(r'_C)|+\lambda}{0}=1$.

From $0\leq |\,\mathtt{Below}_c(r_C)\cap \mathtt{Below}_{c'}(r'_C)|<m$, we deduce:

$$\sum_{i=1}^{k-2}\binom{\lambda}{i} \leq \sum_{i=1}^{k-2}\binom{|\,\mathtt{Below}_c(r_C)\cap \mathtt{Below}_{c'}(r'_C)|+\lambda}{i} < \sum_{i=1}^{k-2}\binom{m+\lambda}{i}.$$

Consequently:

$$\delta_{\mathtt{KT}}(r_C,r'_C)\left(1+\sum_{i=1}^{k-2}\binom{\lambda}{i}\right) \leq \delta^k_{\mathtt{KT}}(r,r') < \delta_{\mathtt{KT}}(r_C,r'_C)\left(1+\sum_{i=1}^{k-2}\binom{m+\lambda}{i}\right)$$

because $\sum_{(c,c')\in C^2} \mathtt{disagree}_{c,c'}(r,r') = \delta_{\mathtt{KT}}(r_C,r'_C)$; from which we obtain:

$$\delta_{\mathtt{KT}}(r_C,\mathcal{P})\left(1+\sum_{i=1}^{k-2}\binom{\lambda}{i}\right) \leq \delta^k_{\mathtt{KT}}(r,\mathcal{P}') < \delta_{\mathtt{KT}}(r_C,\mathcal{P})\left(1+\sum_{i=1}^{k-2}\binom{m+\lambda}{i}\right) \tag{2}$$

because $\sum_{r'\in\mathcal{P}'}\delta_{\mathtt{KT}}(r_C,r'_C)=\delta_{\mathtt{KT}}(r_C,\mathcal{P})$ and $\sum_{r'\in\mathcal{P}'}\delta^k_{\mathtt{KT}}(r,r')=\delta^k_{\mathtt{KT}}(r,\mathcal{P}')$, provided that $\mathtt{tail}_\lambda(r)=c^*_1\succ_r\ldots\succ_r c^*_\lambda$ and $\mathtt{tail}_\lambda(r')=c^*_1\succ_{r'}\ldots\succ_{r'}c^*_\lambda\ \forall r'\in\mathcal{P}'$.

Now, note that:

$$\binom{m+\lambda}{i} = \frac{\lambda!}{i!(\lambda-i)!}\frac{\prod_{j=1}^m(\lambda+j)}{\prod_{j=1}^m(\lambda-i+j)} = \binom{\lambda}{i}\prod_{j=1}^m\frac{\lambda+j}{\lambda+j-i}. \tag{3}$$

If one sets $\lambda=4nm^4$, the following inequalities hold:

$$\frac{4nm^4+j}{4nm^4+j-i} = 1+\frac{i}{4nm^4+j-i} \leq 1+\frac{i}{4nm^4-i}$$
$$\leq 1+\frac{2i}{4nm^4} \leq 1+\frac{1}{2nm^3}$$

where the second inequality follows from $4nm^4/(4nm^4 - i) \leq 2$ for $i \in [\![1, m]\!]$.

From Equation 3, we deduce then:

$$\binom{m + 4nm^4}{i} \leq \binom{4nm^4}{i}(1 + \frac{1}{2nm^3})^m \leq \binom{4nm^4}{i}(1 + \frac{1}{nm^2})$$

where the second inequality follows from Lemma 2 with $\varepsilon = \frac{1}{2nm^3}$ because $\frac{1}{2nm^3} < \frac{1}{2m}$ for $m \geq 3$.

Coming back to Equation 2, this implies:

$$\delta_{\mathtt{KT}}(r_C, \mathcal{P}) \leq \frac{\delta^k_{\mathtt{KT}}(r, \mathcal{P}')}{1 + \sum_{i=1}^{k-2} \binom{4nm^4}{i}} < \delta_{\mathtt{KT}}(r_C, \mathcal{P}) + \frac{1}{nm^2}\, \delta_{\mathtt{KT}}(r_C, \mathcal{P})$$

and therefore:

$$\delta_{\mathtt{KT}}(r_C, \mathcal{P}) \leq \frac{\delta^k_{\mathtt{KT}}(r, \mathcal{P}')}{1 + \sum_{i=1}^{k-2} \binom{4nm^4}{i}} < \delta_{\mathtt{KT}}(r_C, \mathcal{P}) + 1$$

because $\delta_{\mathtt{KT}}(r_C, \mathcal{P}) \leq n\binom{m}{2} \leq nm^2$. In particular, $\delta_{\mathtt{KT}}(r_C, \mathcal{P}) \leq \tau$ for an integer $\tau$ iff $\delta^k_{\mathtt{KT}}(r, \mathcal{P}') \leq (1 + \sum_{i=1}^{k-2} \binom{4nm^4}{i})(\tau + 1) - 1 = \tau'$. This shows that $(\mathcal{P}, \tau)$ is a yes instance of 2-KAP-DEC iff $(\mathcal{P}', \tau')$ is a yes instance of $k$-KAP-DEC.

It is known that 2-KAP-DEC is NP-complete even if the number $n$ of voters equals 4 [9] and even if the average range of candidates equals 2 [22]. As the reduction above preserves the number of voters and decreases the average range of candidates, the same results hold for $k$-KAP-DEC. □

Although the above proof uses a conversion from the $k$-wise Kemeny rule to the standard Kemeny rule, note that it means in no way that both rules are equivalent. There indeed exist instances with arbitrary large sets of candidates such that the $k$-wise Kemeny rule differs from the Kemeny rule. Consider for instance the election described in Example 1. The only $k$-wise Kemeny consensus ranking is $c_1 \succ c_2 \succ c_3$, while the only pairwise Kemeny consensus ranking is $c_2 \succ c_3 \succ c_1$. Now modify the preference profile of the previous election by adding candidates $c_4$ to $c_k$, with $k$ arbitrarily large such that for any ranking $r$ in the new preference profile, $\mathtt{head}_{k-3}(r) = c_k \succ c_{k-1} \succ \ldots \succ c_4$. With this new preference profile, the only $k$-wise Kemeny consensus ranking is $c_k \succ c_{k-1} \succ \ldots \succ c_4 \succ c_1 \succ c_2 \succ c_3$, while the only pairwise Kemeny consensus ranking is $c_k \succ c_{k-1} \succ \ldots \succ c_4 \succ c_2 \succ c_3 \succ c_1$.

Despite Theorem 1, $k$-KAP is obviously FPT w.r.t. the number $m$ of candidates, by simply trying the $m!$ rankings in $\mathcal{R}(C)$. We now design a dynamic programming procedure which significantly improves this time complexity.

**Proposition 6.** *If $r^*$ is an optimal ranking for k-KAP, then $\delta^k_{\mathtt{KT}}(r^*, \mathcal{P}) = d^k_{\mathtt{KT}}(C)$, where, for any subset $S \subseteq C$, $d^k_{\mathtt{KT}}(S)$ is defined by the recursive relation:*

$$\begin{aligned} d^k_{\mathtt{KT}}(S) = \min_{c \in S} [& d^k_{\mathtt{KT}}(S \setminus \{c\}) \\ & + \sum_{r \in \mathcal{P}_S} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S| - \mathtt{rk}(c', r) - 1}{i}] \\ d^k_{\mathtt{KT}}(\emptyset) = 0. & \end{aligned} \tag{4}$$

*Proof.* Given $S \subseteq C$ and $c \in S$, let us define $\mathcal{R}_c(S)$ as $\{r \in \mathcal{R}(S) \text{ s.t. } \mathtt{top}_r(S) = c\}$. The set $\Delta^k(S)$ can be partitioned into $\Delta^k_c(S) = \{S' \in \Delta^k(S) \text{ s.t. } c \in S'\}$ and $\Delta^k_{\bar{c}}(S) = \{S' \in \Delta^k(S) \text{ s.t. } c \notin S'\} = \Delta^k(S \setminus \{c\})$. Given a preference profile $\mathcal{P}$ over $C$ and a ranking $\hat{r} \in \mathcal{R}_c(S)$, the summation defining $\delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_S)$ breaks down as follows:

$$\begin{aligned} \delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_S) &= \sum_{r \in \mathcal{P}_S} \sum_{S' \in \Delta^k(S)} \mathtt{disagree}_{S'}(\hat{r}, r) \\ &= \delta^k_{\mathtt{KT}}(\hat{r}_{S \setminus \{c\}}, \mathcal{P}_{S \setminus \{c\}}) + \sum_{r \in \mathcal{P}_S} \sum_{S' \in \Delta^k_c(S)} \mathtt{disagree}_{S'}(\hat{r}, r). \end{aligned} \tag{5}$$

Using the same reasoning as in Equation 1 on page 5, the second summand in Equation 5 can be rewritten as follows:

$$\sum_{r \in \mathcal{P}_S} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|\,\mathtt{Below}_c(\hat{r}) \cap \mathtt{Below}_{c'}(r)\,|}{i}$$

because $\mathtt{top}_{\hat{r}}(S') = c$ for all $S' \in \Delta^k_c(S)$. Note that $\mathtt{Below}_c(\hat{r}) = S \setminus \{c\}$ and $\mathtt{Below}_{c'}(r) = \{c'' \in S \text{ s.t. } c' \succ_r c''\} \subseteq S$, thus $|\,\mathtt{Below}_c(\hat{r}) \cap \mathtt{Below}_{c'}(r)\,| = |S| - \mathtt{rk}(c', r) - 1$. Hence, $\delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_S)$ is equal to:

$$\delta^k_{\mathtt{KT}}(\hat{r}_{S \setminus \{c\}}, \mathcal{P}_{S \setminus \{c\}}) + \sum_{r \in \mathcal{P}_S} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S| - \mathtt{rk}(c', r) - 1}{i}. \tag{6}$$

Consider now a ranking $r^* \in \mathcal{R}(S)$ such that $\delta^k_{\mathtt{KT}}(r^*, \mathcal{P}_S) = \min_{r \in \mathcal{R}(S)} \delta^k_{\mathtt{KT}}(r, \mathcal{P}_S)$. We have:

$$\begin{aligned} \delta^k_{\mathtt{KT}}(r^*, \mathcal{P}_S) &= \min_{c \in S} \min_{\hat{r} \in \mathcal{R}_c(S)} \delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_S) \\ &= \min_{c \in S} \Big( \big( \min_{\hat{r} \in \mathcal{R}(S \setminus \{c\})} \delta^k_{\mathtt{KT}}(\hat{r}, \mathcal{P}_{S \setminus \{c\}}) \big) \\ &\quad + \sum_{r \in \mathcal{P}_S} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S| - \mathtt{rk}(c', r) - 1}{i} \Big) \end{aligned}$$

because the second summand in Equation 6 does not depend on $\hat{r}$ (it only depends on $c$, which is the argument of the first min operator). If one denotes $\min_{r\in\mathcal{R}(S)} \delta^k_{\mathtt{KT}}(r,\mathcal{P}_S)$ by $d^k_{\mathtt{KT}}(S)$, one obtains Equation 4. This concludes the proof. □

A candidate $c\in S$ that realizes the minimum in Equation 4 can be ranked in first position in an optimal ranking for $\mathcal{P}_S$. Once $d^k_{\mathtt{KT}}(S)$ is computed for each $S\subseteq C$, a ranking $r^*$ achieving the optimal value $d^k_{\mathtt{KT}}(C)$ can thus be determined recursively starting from $S=C$. The complexity of the induced dynamic programming method is $O(2^m m^2 n)$ as there are $2^m$ subsets $S\subseteq C$ to consider and each value $d^k_{\mathtt{KT}}(S)$ is computed in $O(m^2 n)$ by Equation 4. The min operation is indeed performed on $m$ values and the sum $\sum_{c'\succ_r c}\sum_{i=0}^{k-2}\binom{|S|-\mathtt{rk}(c',r)-1}{i}$ is computed incrementally in $O(m)$, which entails an $O(mn)$ complexity for the second summand in Equation 4 (the $n$ factor is due to the sum over all $r\in\mathcal{P}_S$). The computation of binomial coefficients $\binom{p}{i}$ for $i\in\{0,\ldots,k-2\}$ and $p\in\{i,\ldots,m-2\}$ is performed in $O(mk)$ in a preliminary step thanks to Pascal's formula.

## 4. The $k$-Wise Majority Digraph

We now propose and investigate a $k$-wise counterpart of the pairwise majority digraph, that will be used in a preprocessing procedure for $k$-KAP.

As stated in the introduction, the pairwise Kemeny rule is strongly related to the *pairwise majority digraph*. We denote by $\mathcal{G}_\mathcal{P}$ the pairwise majority digraph associated to profile $\mathcal{P}$. We recall that in this digraph, there is one vertex per candidate, and there is an arc from candidate $c$ to candidate $c'$ if a strict majority of voters prefers $c$ to $c'$. In the weighted pairwise majority digraph, each arc $(c,c')$ is weighted by $w_\mathcal{P}(c,c') \coloneqq |\{r\in\mathcal{P} \text{ s.t. } c\succ_r c'\}| - |\{r\in\mathcal{P} \text{ s.t. } c'\succ_r c\}|$.

**Example 5.** *Consider a profile $\mathcal{P}$ with 10 voters and 6 candidates such that:*
*– 4 voters have preferences $c_1\succ c_2\succ c_4\succ c_3\succ c_5\succ c_6$;*
*– 4 voters have preferences $c_1\succ c_3\succ c_2\succ c_4\succ c_5\succ c_6$;*
*– 1 voter has preferences $c_6\succ c_1\succ c_2\succ c_4\succ c_3\succ c_5$;*
*– 1 voter has preferences $c_6\succ c_1\succ c_4\succ c_3\succ c_2\succ c_5$.*
*The weighted pairwise majority digraph $\mathcal{G}_\mathcal{P}$ is displayed on the left of Figure 1.*

From $\mathcal{G}_\mathcal{P}$, we can define a set of *consistent* rankings:

**Definition 4.** *Let $\mathcal{G}$ be a digraph whose vertices correspond to the candidates in $C$. Let $B_1(\mathcal{G}),\ldots,B_{\sigma(\mathcal{G})}(\mathcal{G})$ denote the subsets of $C$ corresponding to the Strongly Connected Components (SCCs) of $\mathcal{G}$, and $\mathcal{O}(\mathcal{G})$ denote the set of linear orders $<_\mathcal{G}$ on $\{1,\ldots,\sigma(\mathcal{G})\}$ such that if there exists an arc $(c,c')$ from $c\in B_i(\mathcal{G})$ to $c'\in B_j(\mathcal{G})$ then $i<_\mathcal{G} j$. Given $<_\mathcal{G}\in\mathcal{O}(\mathcal{G})$, we say that a ranking $r$ is* consistent *with $<_\mathcal{G}$ if the candidates in $B_i$ are ranked before the ones of $B_j$ when $i<_\mathcal{G} j$.*

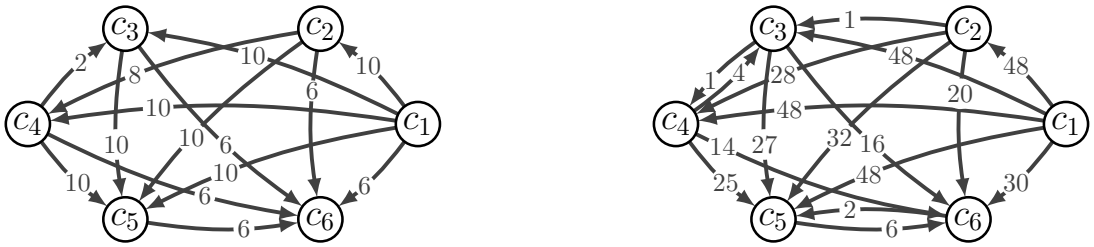



Figure 1: Weighted $k$-wise majority digraph in Example 5 for $k=2$ (left) and $k=3$ (right).

The following result states that, for any $<_{\mathcal{G}_\mathcal{P}} \in \mathcal{O}(\mathcal{G}_\mathcal{P})$, there exists a consensus ranking for $\delta_{\mathtt{KT}}$ among the rankings consistent with $<_{\mathcal{G}_\mathcal{P}}$.

**Theorem 2** (Theorem 16 in [23], by Charon and Hudry)**.** *Let $\mathcal{P}$ be a profile over $C$ and assume that the SCCs of $\mathcal{G}_\mathcal{P}$ are numbered according to a linear order $<_{\mathcal{G}_\mathcal{P}} \in \mathcal{O}(\mathcal{G}_\mathcal{P})$. Consider the ranking $r^*$, consistent with $<_{\mathcal{G}_\mathcal{P}}$, obtained by the concatenation of rankings $r^*_1, \ldots, r^*_{t(\mathcal{G}_\mathcal{P})}$ where $\delta_{\mathtt{KT}}(r^*_i, \mathcal{P}_{B_i(\mathcal{G}_\mathcal{P})}) = \min_{r \in \mathcal{R}(B_i(\mathcal{G}_\mathcal{P}))} \delta_{\mathtt{KT}}(r, \mathcal{P}_{B_i(\mathcal{G}_\mathcal{P})})$. We have:*

$$\delta_{\mathtt{KT}}(r^*, \mathcal{P}) = \min_{r \in \mathcal{R}(C)} \delta_{\mathtt{KT}}(r, \mathcal{P}).$$

*That is, $r^*$ is a consensus ranking according to the Kemeny rule. Furthermore, if $\mathcal{O}(\mathcal{G}_\mathcal{P}) = \{<_{\mathcal{G}_\mathcal{P}}\}$ and $w_\mathcal{P}(c,c') > 0$ for all $c \in B_i(\mathcal{G}_\mathcal{P})$ and $c' \in B_j(\mathcal{G}_\mathcal{P})$ when $i <_{\mathcal{G}_\mathcal{P}} j$, then all consensus rankings are consistent with $<_{\mathcal{G}_\mathcal{P}}$.*

This result does not hold anymore if one uses $\delta^k_{\mathtt{KT}}$ (with $k \geq 3$) instead of $\delta_{\mathtt{KT}}$, as shown by the following example.

**Example 6.** *Let us denote by $\mathcal{P}$ the profile of Example 1. The pairwise majority digraph $\mathcal{G}_\mathcal{P}$ has three SCCs $B_1(\mathcal{G}_\mathcal{P}) = \{c_2\}$, $B_2(\mathcal{G}_\mathcal{P}) = \{c_3\}$ and $B_3(\mathcal{G}_\mathcal{P}) = \{c_1\}$. In this example, $\mathcal{O}(\mathcal{G}_\mathcal{P}) = \{<_{\mathcal{G}_\mathcal{P}}\}$ where $1 <_{\mathcal{G}_\mathcal{P}} 2 <_{\mathcal{G}_\mathcal{P}} 3$. The only ranking consistent with $<_{\mathcal{G}_\mathcal{P}}$ is $c_2 \succ c_3 \succ c_1$ while the only consensus ranking w.r.t. the 3-wise Kemeny rule is $c_1 \succ c_2 \succ c_3$.*

In order to adapt Theorem 2 to the $k$-wise Kemeny rule, we now introduce the concept of *$k$-wise majority digraph*. Let $\Delta^k_{cc'}(S) = \{S' \in \Delta^k(S) \text{ s.t. } \{c,c'\} \subseteq S'\}$. If $S$ is not specified, it is assumed to be $C$. Given a ranking $r$, we denote by $\Delta^k_r(S,c,c')$ the set

$\{S' \in \Delta^k_{cc'}(S)$ s.t. $\texttt{top}_r(S') = c\}$. Given a profile $\mathcal{P}$, we denote by $\phi^k_{\mathcal{P}}(S,c,c')$ the value $\sum_{r\in\mathcal{P}} |\Delta^k_r(S,c,c')|$ and by $w^k_{\mathcal{P}}(S,c,c')$ the difference $\phi^k_{\mathcal{P}}(S,c,c') - \phi^k_{\mathcal{P}}(S,c',c)$. This definition implies that $w^k_{\mathcal{P}}(S,c',c) = -w^k_{\mathcal{P}}(S,c,c')$. The value $w^k_{\mathcal{P}}(S,c,c')$ is the net agreement loss that would be incurred by swapping $c$ and $c'$ in a feasible solution $r$ of $k$-KAP where $\texttt{rk}(c',r)=\texttt{rk}(c,r)+1$ and $S=\texttt{Below}_{c'}(r)\cup\{c,c'\}$. If $\max_{S\in\Delta^m_{cc'}} w^k_{\mathcal{P}}(S,c,c')\geq 0$ (resp. $\min_{S\in\Delta^m_{cc'}} w^k_{\mathcal{P}}(S,c,c')>0$) then, in a consensus ranking $r$ for $\delta^k_{\texttt{KT}}$ where $c$ and $c'$ would be consecutive, it is possible (resp. necessary) that $c \succ_r c'$.

**Definition 5.** *The $k$-wise majority digraph associated to a profile $\mathcal{P}$ over a set $C$ of candidates is the digraph $\mathcal{G}^k_{\mathcal{P}} = (\mathcal{V},\mathcal{A})$, where $\mathcal{V}=C$ and $(c,c')\in\mathcal{A}$ iff:*

$$\exists S \in \Delta^m_{cc'} \text{ s.t. } w^k_{\mathcal{P}}(S,c,c') > 0.$$

*In the weighted $k$-wise majority digraph, each edge $(c,c')$ is weighted by:*

$$w^k_{\mathcal{P}}(c,c') \coloneqq \max_{S\in\Delta^m_{cc'}} w^k_{\mathcal{P}}(S,c,c').$$

Note that, if $k \geq 3$, we may obtain edges $(c,c')$ and $(c',c)$ both with strictly positive weights (which is impossible in the pairwise case). For instance, for the profile $\mathcal{P}$ of Example 5, $w^3_{\mathcal{P}}(c_3,c_4)=w^3_{\mathcal{P}}(\{c_2,c_3,c_4\},c_3,c_4)=1$ and $w^3_{\mathcal{P}}(c_4,c_3)=w^3_{\mathcal{P}}(\{c_3,c_4,c_5\},c_4,c_3)=4$. For illustration, let us explain how $w^3_{\mathcal{P}}(\{c_3,c_4,c_5\},c_4,c_3)$ yields 4. Table 2 summarizes the number of times $c_4$ and $c_3$ appear in top position of $\{c_3,c_4\}$ or $\{c_3,c_4,c_5\}$ for each ranking $r$ in $\mathcal{P}$. By summing over $r\in\mathcal{P}$: $\phi^3_{\mathcal{P}}(\{c_3,c_4,c_5\},c_4,c_3) - \phi^3_{\mathcal{P}}(\{c_3,c_4,c_5\},c_3,c_4) = (4\times 2+2+2)-(4\times 2)=4$.

Table 2: Number of times $c_4$ and $c_3$ appear in top position of $\{c_3,c_4\}$ or $\{c_3,c_4,c_5\}$ for each ranking $r$ in the profile $\mathcal{P}$ of Example 5.

| $r$ | $\lvert\Delta^3_r(\{c_3,c_4,c_5\},c_4,c_3)\rvert$ | $\lvert\Delta^3_r(\{c_3,c_4,c_5\},c_3,c_4)\rvert$ |
|---|---|---|
| $c_1 \succ c_2 \succ c_4 \succ c_3 \succ c_5 \succ c_6$ $(\times 4)$ | 2 | 0 |
| $c_1 \succ c_3 \succ c_2 \succ c_4 \succ c_5 \succ c_6$ $(\times 4)$ | 0 | 2 |
| $c_6 \succ c_1 \succ c_2 \succ c_4 \succ c_3 \succ c_5$ $(\times 1)$ | 2 | 0 |
| $c_6 \succ c_1 \succ c_4 \succ c_3 \succ c_2 \succ c_5$ $(\times 1)$ | 2 | 0 |

The obtained weighted digraph $\mathcal{G}^3_{\mathcal{P}}$ is shown on the right of Figure 1 (an efficient manner to compute a set $S$ maximizing $w^k_{\mathcal{P}}(S,c,c')$ will be explained later on page 24). Besides, for any $\mathcal{P}$, $\mathcal{G}^2_{\mathcal{P}}$ is the pairwise majority digraph as $\Delta^2_{cc'}(S) = \{\{c,c'\}\}\ \forall S\in\Delta^m_{cc'}$. Theorem 2 adapts as follows for an arbitrary $k$:

**Theorem 3.** *Let $\mathcal{P}$ be a profile over $C$ and assume that the SCCs of $\mathcal{G}^k_{\mathcal{P}}$ are numbered according to a linear order $<_{\mathcal{G}^k_{\mathcal{P}}} \in \mathcal{O}(\mathcal{G}^k_{\mathcal{P}})$. Among the rankings consistent with $<_{\mathcal{G}^k_{\mathcal{P}}}$, there*

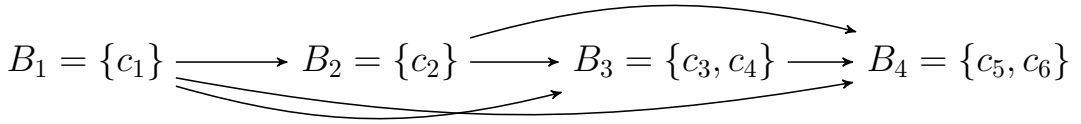



Figure 2: The meta-graph of SCCs of $\mathcal{G}^3_{\mathcal{P}}$ in Example 5.

*exists a consensus ranking w.r.t. the k-wise Kemeny rule. Besides, if* $\mathcal{O}(\mathcal{G}^k_{\mathcal{P}})=\{<_{\mathcal{G}^k_{\mathcal{P}}}\}$ *and* $\min_{S\in\Delta^m_{cc'}} w^k_{\mathcal{P}}(S,c,c')>0$[1] *for all* $c\in B_i(\mathcal{G}^k_{\mathcal{P}})$ *and* $c'\in B_j(\mathcal{G}^k_{\mathcal{P}})$ *when* $i<_{\mathcal{G}^k_{\mathcal{P}}} j$*, then all consensus rankings are consistent with* $<_{\mathcal{G}^k_{\mathcal{P}}}$*.*

*Proof.* Assume that the SCCs of $\mathcal{G}^k_{\mathcal{P}}$ are numbered according to a linear order $<_{\mathcal{G}^k_{\mathcal{P}}}\in\mathcal{O}(\mathcal{G}^k_{\mathcal{P}})$ and consider a ranking $r$ which is not consistent with $<_{\mathcal{G}^k_{\mathcal{P}}}$. Hence, there exists a pair $(c,c')$ such that $c$ directly follows $c'$ in $r$ while $c\in B_i(\mathcal{G}^k_{\mathcal{P}})$ and $c'\in B_j(\mathcal{G}^k_{\mathcal{P}})$ with $i<j$. Since $i<j$, there is no arc from $c'$ to $c$ in $\mathcal{G}^k_{\mathcal{P}}$ (i.e., $\forall S\in\Delta^m_{cc'}, w^k_{\mathcal{P}}(S,c,c')\geq 0$). Let $S$ be the set composed of $c'$ and all candidates placed after $c'$ in $r$, including $c$. Then the ranking $r^{c\leftrightarrow c'}$ obtained from $r$ by exchanging the positions of $c$ and $c'$ verifies $\delta^k_{\mathtt{KT}}(r^{c\leftrightarrow c'},\mathcal{P})=\delta^k_{\mathtt{KT}}(r,\mathcal{P})-w^k_{\mathcal{P}}(S,c,c')\leq\delta^k_{\mathtt{KT}}(r,\mathcal{P})$. The repetition of this argument concludes the proof of the first claim. The second claim is proved similarly because, in this case, $\delta^k_{\mathtt{KT}}(r^{c\leftrightarrow c'},\mathcal{P})=\delta^k_{\mathtt{KT}}(r,\mathcal{P})-w^k_{\mathcal{P}}(S,c,c')<\delta^k_{\mathtt{KT}}(r)$. □

**Example 7.** *The meta-graph of SCCs of* $\mathcal{G}^3_{\mathcal{P}}$ *in Example 5 is represented in Figure 2. The above result implies that there exists a consensus ranking among* $c_1\succ c_2\succ c_3\succ c_4\succ c_5\succ c_6$*,* $c_1\succ c_2\succ c_3\succ c_4\succ c_6\succ c_5$*,* $c_1\succ c_2\succ c_4\succ c_3\succ c_5\succ c_6$ *and* $c_1\succ c_2\succ c_4\succ c_3\succ c_6\succ c_5$*.*

To take advantage of Theorem 3, one could try 1) to index the SCCs of $\mathcal{G}^k_{\mathcal{P}}$ according to a linear order $<_{\mathcal{G}^k_{\mathcal{P}}}\in\mathcal{O}(\mathcal{G}^k_{\mathcal{P}})$, and then 2) to work on each SCC separately, before concatenating the obtained rankings. However, for a consensus ranking consistent with $<_{\mathcal{G}^k_{\mathcal{P}}}$, the relative positions of candidates in $B_i(\mathcal{G}^k_{\mathcal{P}})$ depend on the set of candidates in $B_{>i}(\mathcal{G}^k_{\mathcal{P}}):=B_{i+1}(\mathcal{G}^k_{\mathcal{P}})\cup\ldots\cup B_{\sigma(\mathcal{G}^k_{\mathcal{P}})}(\mathcal{G}^k_{\mathcal{P}})$ (but not on their order). The influence of $B_{>i}(\mathcal{G}^k_{\mathcal{P}})$ can be captured in the dynamic programming procedure by applying a modified version of Equation 4 separately for each subset $B_{\sigma(\mathcal{G}^k_{\mathcal{P}})}(\mathcal{G}^k_{\mathcal{P}})$ downto $B_1(\mathcal{G}^k_{\mathcal{P}})$. Formally, if $r^*$ is optimal for $k$-KAP, then:

$$\delta^k_{\mathtt{KT}}(r^*,\mathcal{P})=\sum_{i=1}^{\sigma(\mathcal{G}^k_{\mathcal{P}})} d^k_{\mathtt{KT}}(B_i(\mathcal{G}^k_{\mathcal{P}}))$$

[1] Or, equivalently, $\max_{S\in\Delta^m_{cc'}} w^k_{\mathcal{P}}(S,c',c)<0$.

where, for any subset $S \subseteq B_i(\mathcal{G}^k_{\mathcal{P}})$, $d^k_{\mathtt{KT}}(S)$ is defined by $d^k_{\mathtt{KT}}(\emptyset) = 0$ and ($B_{>i}$ stands for $B_{>i}(\mathcal{G}^k_{\mathcal{P}})$):

$$d^k_{\mathtt{KT}}(S) = \min_{c \in S}[d^k_{\mathtt{KT}}(S \backslash \{c\}) + \sum_{r \in \mathcal{P}_{S \cup B_{>i}}} \sum_{c' \succ_r c} \sum_{i=0}^{k-2} \binom{|S| + |B_{>i}| - \mathtt{rk}(c', r) - 1}{i}].$$

It amounts to replacing $S$ by $S \cup B_{>i}$ in the second summand of Equation 4 to take into account the existence of a consensus ranking where all the candidates of $B_{>i}$ are ranked after those of $B_i$. Let $r^*_i$ be a ranking of $B_i(\mathcal{G}^k_{\mathcal{P}})$ such that $\delta^k_{\mathtt{KT}}(r^*_{\geq i}, \mathcal{P}_{B_{\geq i}(\mathcal{G}^k_{\mathcal{P}})}) = d^k_{\mathtt{KT}}(B_i(\mathcal{G}^k_{\mathcal{P}})) + \ldots + d^k_{\mathtt{KT}}(B_{\sigma(\mathcal{G}^k)}(\mathcal{G}^k_{\mathcal{P}}))$, where $r^*_{\geq i}$ is the ranking obtained by the concatenation of rankings $r^*_i, \ldots, r^*_{\sigma(\mathcal{G}^k)}$ in this order. The ranking $r^*_{\geq 1}$ of $C$ is a consensus ranking w.r.t. the $k$-wise Kemeny rule. Given Theorem 3, the $k$-wise majority digraph thus seems promising to boost the computation of a consensus ranking. Unfortunately, the following negative result holds:

**Theorem 4.** *Given two candidates $c$ and $c'$ in a profile $\mathcal{P}$, determining if $\max_{S \in \Delta^m_{cc'}} w^k_{\mathcal{P}}(S, c, c') > 0$ is an NP-complete problem for any constant $k \geq 4$.*

*Proof.* For the membership part, note that given a set $S$, we can check in polynomial time if $w^k_{\mathcal{P}}(S, c, c') > 0$. Indeed, we can enumerate all sets in $\Delta^k_{cc'}(S)$ and for each ranking $r \in \mathcal{P}$ count how many are in $\Delta^k_r(S, c, c')$ or in $\Delta^k_r(S, c', c)$.

For the hardness part, we make a reduction from the set cover problem, known to be NP-complete [24]:

Set Cover Problem
*Instance:* A set of elements $\mathcal{X} = \{x_1, \ldots, x_p\}$, a collection $\mathcal{T} = \{T_1, \ldots, T_q\}$ of sets of elements of $\mathcal{X}$, and a positive integer $b$.
*Question:* Does there exist a subcollection $\mathcal{K} \subseteq \mathcal{T}$ of at most $b$ sets that covers $\mathcal{X}$ (i.e., such that $\bigcup_{T \in \mathcal{K}} T = \mathcal{X}$)?

We assume that no set in $\mathcal{T}$ contains $\mathcal{X}$ as otherwise the problem is trivial. Furthermore, we assume that no element in $\mathcal{X}$ is contained in all sets of $\mathcal{T}$ as otherwise this element could be discarded from the instance as any solution would cover this element. We now detail the preference profile that we create from an instance of the set cover problem.

*Set of candidates:* We will create a profile such that $\max_{S \in \Delta^m_{cc'}} w^k_{\mathcal{P}}(S, c, c') > 0$ iff the answer to the set cover problem is yes for the instance under consideration. More precisely,

we will show that if $\mathcal{X}$ cannot be covered by a subcollection $\mathcal{K} \subseteq \mathcal{T}$ with less than $b$ sets, then $w_{\mathcal{P}}^k(S, c, c') < 0$ for all $S$ in $\Delta_{cc'}^m$. Otherwise, if set $\mathcal{X}$ can be covered with a subcollection $\mathcal{K} \subseteq \mathcal{T}$ with less than $b$ sets, then there exists a set $S$ in $\Delta_{cc'}^m$ such that $w_{\mathcal{P}}^k(S, c, c') > 0$. In addition to candidates $c$ and $c'$, for each pair $(x, T) \in \mathcal{X} \times \mathcal{T}$ such that $x \in T$ we create a candidate $c_{x,T}$. Moreover, for each set $T \in \mathcal{T}$ we create a candidate $c_T$. In the sequel, we may call candidates $c_{x,T}$ *element candidates* and candidates $c_T$ *set candidates*. This process yields at most $pq+2$ candidates ($(p-1)q+q+2$). The candidates $c_{x,T}$ and $c_T$ will make the correspondence with the subcollection $\mathcal{K}$: the candidate $c_T$ will be in the set $S$ iff $T \in \mathcal{K}$ and the candidate $c_{x,T}$ will be in the set $S$ if $T$ is added to $\mathcal{K}$ in order to cover $x$.

*Set of voters:* For each pair $(x, T) \in \mathcal{X} \times \mathcal{T}$ such that $x \in T$, we create $2b$ voters $v_{x,T}^s$ ($s \in \{1, \ldots, 2b\}$) with the same ranking $r_{x,T}$ such that $\texttt{tail}_4(r_{x,T}) = c \succ c_{x,T} \succ c_T \succ c'$. For each element $x \in \mathcal{X}$, we create $2b$ voters $v_x^s$ ($s \in \{1, \ldots, 2b\}$) with the same ranking $r_x$ such that $\texttt{tail}_{u+2}(r_x) = c' \succ c_{x,T_{i_1}} \succ \ldots \succ c_{x,T_{i_u}} \succ c$ where $T_{i_1}, T_{i_2}, \ldots, T_{i_u}$ are the different sets in $\mathcal{T}$ that contain $x$. For each set $T \in \mathcal{T}$ we create $2b|T|+2$ voters $v_T^s$ ($s \in \{1, \ldots, 2b|T|+2\}$) with the same ranking $r_T$ such that $\texttt{tail}_3(r_T) = c' \succ c_T \succ c$. Lastly, we create $2q+1+2b$ voters $v^s$ ($s \in \{1, \ldots, 2q+1+2b\}$) with the same ranking $r$ such that $\texttt{tail}_2(r) = c \succ c'$. In the end, we obtain $O(bpq)$ voters. Note that $\mathcal{P}$ can be build with no unanimity dominance relationship between two candidates.

We now show that, given $k \geq 4$, we have $\max_{S \in \Delta_{cc'}^m} w_{\mathcal{P}}^k(S, c, c') > 0$ iff the answer to the set cover problem is yes.

First note that when $S = \{c, c'\}$, we have the following values for $\phi_{\mathcal{P}}^k(S, c, c')$ and $\phi_{\mathcal{P}}^k(S, c', c)$:

$$\phi_{\mathcal{P}}^k(\{c, c'\}, c, c') = 2b \sum_{T \in \mathcal{T}} |T| + 2q + 2b + 1$$
$$\phi_{\mathcal{P}}^k(\{c, c'\}, c', c) = 2b \sum_{T \in \mathcal{T}} |T| + 2q + 2bp.$$

Hence,

$$\begin{aligned} w_{\mathcal{P}}^k(\{c, c'\}, c, c') &= \phi_{\mathcal{P}}^k(\{c, c'\}, c, c') - \phi_{\mathcal{P}}^k(\{c, c'\}, c', c) \\ &= 2b(1-p) + 1 < 0, \end{aligned}$$

as $p \geq 2$ and $b \geq 1$.

Now let us look at how adding an element to $S$ starting from $S = \{c, c'\}$ modifies values $\phi_{\mathcal{P}}^k(S, c, c')$ and $\phi_{\mathcal{P}}^k(S, c', c)$. We will assume without loss of generality that we add set candidates before element candidates.

- If we assume $S$ is only composed of $c$, $c'$ and set candidates, then adding a candidate $c_T$ to it results in adding $2b|T|$ to $\phi^k_{\mathcal{P}}(S, c, c')$ and $2b|T| + 2$ to $\phi^k_{\mathcal{P}}(S, c', c)$.

- If we add a candidate $c_{x,T}$ to $S$, then we add $2b$ to $\phi^k_{\mathcal{P}}(S, c, c')$ if $c_T \notin S$ and $4b$ otherwise. Additionally, we add $2b$ to $\phi^k_{\mathcal{P}}(S, c', c)$ if there is no other $c_{x,T'} \in S$, otherwise we add to it something that is greater than or equal to $4b$. Note that we have used the fact that $k \geq 4$, as we would only add $2b$ to both values if $k$ was equal to 3.

From these observations, we can derive the following rules:

1. If $\{c_{x,T}, c_{x,T'}\} \subset S$ with $T \neq T'$, then $w^k_{\mathcal{P}}(S, c, c') \leq w^k_{\mathcal{P}}(S \setminus \{c_{x,T'}\}, c, c')$.
2. If $c_{x,T} \in S$ and $c_T \notin S$, then $w^k_{\mathcal{P}}(S, c, c') \leq w^k_{\mathcal{P}}(S \setminus \{c_{x,T}\}, c, c')$.
3. If $c_T \in S$ and $\forall x \in T, c_{x,T} \notin S$, then $w^k_{\mathcal{P}}(S, c, c') \leq w^k_{\mathcal{P}}(S \setminus \{c_T\}, c, c')$.

Stated differently, while rule 1 states that, to maximize $w^k_{\mathcal{P}}(S, c, c')$, we should keep no more than one candidate $c_{x,T}$ per element $x$, rules 2 and 3 state that there should be a candidate $c_{x,T}$ in $S$ iff there should also be candidate $c_T$.

Let us now consider a set $S \in \Delta^m_{cc'}$ verifying rules 1, 2 and 3 that includes one candidate $c_{x,T}$ for $s$ elements $\{x_{i_1}, \ldots, x_{i_s}\}$ and $v$ set candidates. Then:

$$\begin{aligned} w^k_{\mathcal{P}}(S, c, c') &= 2b(1-p) + 1 - 2v + 2bs \\ &= 2b(s-p) + 2(b-v) + 1. \end{aligned}$$

Note that if $S \neq \{c, c'\}$, then $v \geq 1$ and that by rule 1, $s \leq p$. If $s < p$, then $w^k_{\mathcal{P}}(S, c, c') \leq -2b + 2(b-v) + 1 = 1 - 2v < 0$. Hence, if $w^k_{\mathcal{P}}(S, c, c') \geq 0$ then $s = p$, which further implies that $v \leq b$. In this case, it is easy to see that the $v$ sets corresponding to the set candidates in $S$ form a valid set cover of $\mathcal{X}$ as $\{x_{i_1}, \ldots, x_{i_s}\} = \mathcal{X}$, $v \leq b$ and $S$ verifies rule 2. To summarize, making the assumption that $w^k_{\mathcal{P}}(S, c, c') \geq 0$, (In fact in this case $w^k_{\mathcal{P}}(S, c, c') > 0$) we have showed that we could build a valid set cover of $\mathcal{X}$ from $S$. Consequently, this implies that if the set cover instance admits no valid set cover, then $\max_{S \in \Delta^m_{cc'}} w^k_{\mathcal{P}}(S, c, c') < 0$.

Conversely, let us assume that there exists a subcollection $\mathcal{K} = \{T_{i_1}, \ldots, T_{i_v}\}$ with $v \leq b$ sets that covers $\mathcal{X}$. We consider a set $S$ such that $\{c_{T_{i_1}}, \ldots, c_{T_{i_v}}, c, c'\} \subset S$ and such that for each $x \in \mathcal{X}$, $S$ contains exactly one candidate $c_{x,T}$ where $x \in T$ and $T \in \mathcal{K}$. Then,

simple calcula show that:

$$\phi^k_{\mathcal{P}}(S,c,c') = 2b\sum_{T\in\mathcal{T}}|T| + 2b\sum_{T\in\mathcal{K}}|T| + 4bp + 2q + 1 + 2b,$$
$$\phi^k_{\mathcal{P}}(S,c',c) = 4bp + 2b\sum_{T\in\mathcal{T}}|T| + 2q + 2b\sum_{T\in\mathcal{K}}|T| + 2v.$$

Hence, $w^k_{\mathcal{P}}(S,c,c') \geq 1$ and $\max_{S\in\Delta^m_{cc'}} w^k_{\mathcal{P}}(S,c,c') > 0$. □

Hence, computing $\mathcal{G}^k_{\mathcal{P}}$ from $\mathcal{P}$ is NP-hard for $k \geq 4$. In contrast, $\mathcal{G}^3_{\mathcal{P}}$ can be computed in polynomial time. Indeed, given a set $S \subset C$ such that $\{c,c'\} \subseteq S$, adding an element $x \notin S$ to $S$ increases $\phi^3_{\mathcal{P}}(S,c,c')$ by one for each $r \in \mathcal{P}$ such that $c \succ_r c'$ and $c \succ_r x$. Let us denote by $\mathcal{P}_{c\succ c'}$ the set $\{r \in \mathcal{P} \text{ s.t. } c \succ_r c'\}$. A set $S^*$ maximizing $w^3_{\mathcal{P}}(S,c,c')$ is $S^* \coloneqq \{c,c'\} \cup \{x \in C \text{ such that } |\mathcal{P}_{c\succ c'} \cap \mathcal{P}_{c\succ x}| > |\mathcal{P}_{c'\succ c} \cap \mathcal{P}_{c'\succ x}|\}$. For instance, coming back to the weighted digraph $\mathcal{G}^3_{\mathcal{P}}$ shown on the right of Figure 1, a set $S^*$ maximizing $w^3_{\mathcal{P}}(S,c_4,c_3)$ in Example 5 is $\{c_3,c_4,c_5\}$, which yields $w^3_{\mathcal{P}}(S^*,c_4,c_3) = 4$ as reported previously.

Note that one can take advantage of the meta-graph of SCCs to trim the graph $\mathcal{G}^k_{\mathcal{P}}$ if one looks for a consensus ranking $r^*$ consistent with a specific order $<_{\mathcal{G}^k_{\mathcal{P}}} \in \mathcal{O}(\mathcal{G}^k_{\mathcal{P}})$. It may indeed happen that, for an edge $(c,c')$, the weight $w^k_{\mathcal{P}}(c,c') = w^k_{\mathcal{P}}(S,c,c') > 0$ corresponds to a set $S$ which contains candidates that will never be below $c$ in $r^*$. Conversely, the set $S$ may omit candidates that are necessarily below $c$ in $r^*$. These constraints can be induced by either unanimity dominance relations or by $<_{\mathcal{G}^k_{\mathcal{P}}}$. The following example illustrates this idea.

**Example 8.** *Let us refine the digraph $\mathcal{G}^3_{\mathcal{P}}$ previously obtained for the profile $\mathcal{P}$ of Example 5. The SCCs are $B_1 = \{c_1\}$, $B_2 = \{c_2\}$, $B_3 = \{c_3,c_4\}$ and $B_4 = \{c_5,c_6\}$ and $\mathcal{O}(\mathcal{G}^k_{\mathcal{P}}) = \{<_{\mathcal{G}^k_{\mathcal{P}}}\}$, where $1 <_{\mathcal{G}^k_{\mathcal{P}}} 2 <_{\mathcal{G}^k_{\mathcal{P}}} 3 <_{\mathcal{G}^k_{\mathcal{P}}} 4$. A set maximizing $w^3_{\mathcal{P}}(S,c_3,c_4)$ is $S = \{c_2,c_3,c_4\}$. This set contains $c_2$ while it is necessarily above $c_3$ in a consistent ranking. Conversely, candidates $c_5$ and $c_6$ are omitted while they are necessarily below $c_3$. By taking into account these constraints, we obtain that a set maximizing $w^3_{\mathcal{P}}(S,c_3,c_4)$ is $S = \{c_3,c_4,c_5,c_6\}$, for which $w^3_{\mathcal{P}}(S,c_3,c_4) = -4$. Hence, we can remove the arc $(c_3,c_4)$ from $\mathcal{G}^3_{\mathcal{P}}$. Similarly, it is possible to show that the arc $(c_6,c_5)$ can be removed from $\mathcal{G}^3_{\mathcal{P}}$. Thanks to these refinement steps, we can conclude that a consensus ranking is $r^* = c_1 \succ c_2 \succ c_4 \succ c_3 \succ c_5 \succ c_6$.*

## 5. A polynomial time 2-approximation algorithm

In this section, we provide a polynomial time 2-approximation algorithm for problem $k$-KAP, in the same spirit as the 2-approximation algorithm by Dwork et al. [9] for the

Kemeny aggregation problem. This latter algorithm relies on the Spearman distance.

**Definition 6.** *The Spearman distance $\delta_S(r, r')$ between two rankings $r$ and $r'$ is defined by*

$$\delta_S(r, r') = \sum_{c \in C} |\, \mathtt{rk}(c, r) - \mathtt{rk}(c, r')|.$$

The *Diaconis-Graham inequality* [25] states that the Kendall tau and Spearman distances remain within a constant factor of each other for all pairs of rankings, namely:

$$\delta_{\mathtt{KT}}(r, r') \leq \delta_{\mathtt{S}}(r, r') \leq 2\, \delta_{\mathtt{KT}}(r, r').$$

Note that the right bound is tight, as witnessed by the two rankings $c_1 \succ_r c_2$ and $c_2 \succ_{r'} c_1$, for which $\delta_{\mathtt{KT}}(r, r') = 1$ and $\delta_{\mathtt{S}}(r, r') = 2$.

Dwork et al. [9] have shown that a ranking $r_S$ minimizing the sum of Spearman distances to the rankings in a preference profile $\mathcal{P}$ (i.e., $\sum_{r' \in \mathcal{P}} \delta_{\mathtt{S}}(r_S, r') = \min_r \sum_{r' \in \mathcal{P}} \delta_{\mathtt{S}}(r, r')$) can be computed in polynomial time via a minimum cost matching algorithm. As a consequence of the Diaconis-Graham inequality, the sum of Kendall tau distances between $r_S$ and the rankings in $\mathcal{P}$ is in the worst case twice that of an optimal ranking for the Kemeny rule. Denoting by $r_{\mathtt{KT}}$ an optimal ranking for the Kemeny rule on $\mathcal{P}$, we have indeed:

$$\sum_{r' \in \mathcal{P}} \delta_{\mathtt{KT}}(r_{\mathtt{S}}, r') \leq \sum_{r' \in \mathcal{P}} \delta_{\mathtt{S}}(r_{\mathtt{S}}, r') \leq \sum_{r' \in \mathcal{P}} \delta_{\mathtt{S}}(r_{\mathtt{KT}}, r') \leq 2 \sum_{r' \in \mathcal{P}} \delta_{\mathtt{KT}}(r_{\mathtt{KT}}, r').$$

In order to obtain the same kind of result for the $k$-wise Kendall tau distance, we introduce a $k$-wise variant of the Spearman distance.

**Definition 7.** *The $k$-wise Spearman distance $\delta^k_{\mathtt{S}}(r, r')$ between two rankings $r$ and $r'$ is defined by:*

$$\delta^k_{\mathtt{S}}(r, r') = \sum_{c \in C} \sum_{i=\min\{\mathtt{rk}(c,r), \mathtt{rk}(c,r')\}}^{\max\{\mathtt{rk}(c,r), \mathtt{rk}(c,r')\}-1} N_{k-2}^{m-i-1}$$

*where $N_k^p = \sum_{i=0}^{k} \binom{p}{i}$ is the number of subsets of size less than or equal to $k$ within a set of size $p$.*

This definition is motivated by the fact that $N_{k-2}^{m-i-1}$ corresponds to the $k$-wise Kendall tau distances between a ranking and the ranking obtained by swapping candidates of ranks $i$ and $i+1$. Note that $\delta^2_{\mathtt{S}} = \delta_{\mathtt{S}}$. We have indeed:

$$\begin{aligned} \sum_{i=\min\{\mathtt{rk}(c,r), \mathtt{rk}(c,r')\}}^{\max\{\mathtt{rk}(c,r), \mathtt{rk}(c,r')\}-1} N_0^{m-i-1} &= \max\{\mathtt{rk}(c, r), \mathtt{rk}(c, r')\} - \min\{\mathtt{rk}(c, r), \mathtt{rk}(c, r')\} \\ &= |\, \mathtt{rk}(c, r) - \mathtt{rk}(c, r')|. \end{aligned}$$

because $N_0^{m-i-1} = 1$ for all $i$ in the summation.

We now state and prove the extension of the Diaconis-Graham inequality to our $k$-wise variants:

**Lemma 3.** *For any two rankings $r$ and $r'$: $\delta^k_{\mathtt{KT}}(r,r') \leq \delta^k_{\mathtt{S}}(r,r') \leq 2\delta^k_{\mathtt{KT}}(r,r')$.*

*Proof.* We first prove that $\delta^k_{\mathtt{KT}}(r,r') \leq \delta^k_{\mathtt{S}}(r,r')$. We recall that:

$$\delta^k_{\mathtt{KT}}(r,r') = |\{S \in \Delta^k : \mathtt{top}_r(S) \neq \mathtt{top}_{r'}(S)\}|$$

which can be reformulated:

$$\delta^k_{\mathtt{KT}}(r,r') = \sum_{c \in C} |\{S \in \Delta^k : \mathtt{top}_r(S) = c \text{ and } \mathtt{top}_{r'}(S) \neq c\}|.$$

Let us denote by $\Delta^k_c(r,r')$ the set $\{S \in \Delta^k : \mathtt{top}_r(S) = c \text{ and } \mathtt{top}_{r'}(S) \neq c\}$. For any $S \in \Delta^k_c(r,r')$, one of the followings holds true:

- $\mathtt{rk}(\mathtt{top}_r(S), r) \leq \mathtt{rk}(\mathtt{top}_{r'}(S), r') < \mathtt{rk}(\mathtt{top}_r(S), r')$,
- or $\mathtt{rk}(\mathtt{top}_{r'}(S), r) > \mathtt{rk}(\mathtt{top}_r(S), r) \geq \mathtt{rk}(\mathtt{top}_{r'}(S), r')$,
- or both (if $\mathtt{rk}(\mathtt{top}_r(S), r) = \mathtt{rk}(\mathtt{top}_{r'}(S), r')$).

$$\begin{aligned}\text{Consequently: } \delta^k_{\mathtt{KT}}(r,r') \leq & \sum_{c \in C} |\{S \in \Delta^k_c(r,r') : \mathtt{rk}(c,r) \leq \mathtt{rk}(\mathtt{top}_{r'}(S), r') < \mathtt{rk}(c,r')\}| \\ & + \sum_{c \in C} |\{S \in \Delta^k_c(r,r') : \mathtt{rk}(\mathtt{top}_{r'}(S), r) > \mathtt{rk}(c,r) \geq \mathtt{rk}(\mathtt{top}_{r'}(S), r')\}|\end{aligned}$$

because the union of the two sets is $\Delta^k_c(r,r')$ and $|A \cup B| \leq |A| + |B|$ for any two sets $A$ and $B$.

On the one hand, we have:

$$\begin{aligned}& \sum_{c \in C} |\{S \in \Delta^k_c(r,r') : \mathtt{rk}(c,r) \leq \mathtt{rk}(\mathtt{top}_{r'}(S), r') < \mathtt{rk}(c,r')\}| \\ \leq & \sum_{\substack{c \in C \\ \mathtt{rk}(c,r) < \mathtt{rk}(c,r')}} \sum_{i=\mathtt{rk}(c,r)}^{\mathtt{rk}(c,r')-1} N^{m-i-1}_{k-2}.\end{aligned}$$

The upper bound results from the fact that each candidate $c'$ such that $\mathtt{rk}(c,r) \leq \mathtt{rk}(c',r') < \mathtt{rk}(c,r')$ belongs to at most $N^{m-\mathtt{rk}(c',r')-1}_{k-2}$ subsets $S \in \Delta^k_c(r,r')$ such that $\mathtt{top}_{r'}(S) = c'$.

On the other hand, we have:

$$\begin{aligned}& \sum_{c \in C} |\{S \in \Delta^k_c(r,r') : \mathtt{rk}(\mathtt{top}_{r'}(S), r) > \mathtt{rk}(c,r) \geq \mathtt{rk}(\mathtt{top}_{r'}(S), r')\}| \\ = & \sum_{c' \in C} |\{S \in \Delta^k_{c'}(r',r) : \mathtt{rk}(c',r) > \mathtt{rk}(\mathtt{top}_r(S), r) \geq \mathtt{rk}(c',r')\}| \\ \leq & \sum_{\substack{c' \in C \\ \mathtt{rk}(c',r) > \mathtt{rk}(c',r')}} \sum_{i=\mathtt{rk}(c',r')}^{\mathtt{rk}(c',r)-1} N^{m-i-1}_{k-2} = \sum_{\substack{c \in C \\ \mathtt{rk}(c,r) > \mathtt{rk}(c,r')}} \sum_{i=\mathtt{rk}(c,r')}^{\mathtt{rk}(c,r)-1} N^{m-i-1}_{k-2}.\end{aligned}$$

The first equality results from the fact that varying $c$ and considering subsets $S$ such that $\mathtt{top}_r(S)=c$ and $\mathtt{rk}(\mathtt{top}_{r'}(S),r) > \mathtt{rk}(c,r) \geq \mathtt{rk}(\mathtt{top}_{r'}(S),r')$ is equivalent to varying $c'$ and considering subsets $S$ such that $\mathtt{top}_{r'}(S)=c'$ and $\mathtt{rk}(c',r) > \mathtt{rk}(\mathtt{top}_r(S),r) \geq \mathtt{rk}(c',r')$.

Hence:

$$\begin{aligned}\delta^k_{\mathtt{KT}}(r,r') \leq & \sum_{\substack{c\in C \\ \mathtt{rk}(c,r)<\mathtt{rk}(c,r')}} \sum_{i=\mathtt{rk}(c,r)}^{\mathtt{rk}(c,r')-1} N^{m-i-1}_{k-2} + \sum_{\substack{c\in C \\ \mathtt{rk}(c,r)>\mathtt{rk}(c,r')}} \sum_{i=\mathtt{rk}(c,r')}^{\mathtt{rk}(c,r)-1} N^{m-i-1}_{k-2} \\ = & \sum_{c\in C} \sum_{i=\min\{\mathtt{rk}(c,r),\mathtt{rk}(c,r')\}}^{\max\{\mathtt{rk}(c,r),\mathtt{rk}(c,r')\}-1} N^{m-i-1}_{k-2} = \delta^k_{\mathtt{S}}(r,r').\end{aligned}$$

We now prove that $\delta^k_{\mathtt{S}}(r,r') \leq 2\delta^k_{\mathtt{KT}}(r,r')$. In this purpose, we consider a sequence $r_0 = r, r_1, \ldots, r_\delta = r'$ of rankings, where $r_{j+1}$ is obtained from $r_j$ by swapping in $r_j$ the candidate $c'_j = \mathtt{top}_{r'}(\{c \in C : \mathtt{rk}(c,r_j) \neq \mathtt{rk}(c,r')\})$ with the previous candidate in $r_j$, denoted by $c_j$. This is like doing an in-place selection sort w.r.t. the order defined by $r'$, moving $c'_j$ to its place in $r'$ by successive swaps. Note the following things:

- at each step $\mathtt{rk}(c'_j,r') < \mathtt{rk}(c'_j,r_j)$ by definition of $c'_j$;
- $\delta = \delta_{\mathtt{KT}}(r,r')$ because each swap of consecutive candidates in $r_j$ decreases by exactly one the number of pairwise disagreements between $r_j$ and $r'$ ($c'_j \succ_{r'} c_j$ by definition of $c'_j$).

The proof is in two steps:

(*i*) We show that $\delta^k_{\mathtt{KT}}(r_j,r') = \delta^k_{\mathtt{KT}}(r_j,r_{j+1}) + \delta^k_{\mathtt{KT}}(r_{j+1},r')$. By induction, an immediate consequence is that:

$$\delta^k_{\mathtt{KT}}(r,r') = \sum_{j=0}^{\delta-1} \delta^k_{\mathtt{KT}}(r_j,r_{j+1}).$$

(*ii*) We define $D_j = \delta^k_{\mathtt{S}}(r_j,r')$. We show that $D_j - D_{j+1} \leq 2\delta^k_{\mathtt{KT}}(r_j,r_{j+1})$, which implies: $\delta^k_{\mathtt{S}}(r,r') = \sum_{j=0}^{\delta-1}(D_j - D_{j+1}) \leq 2\sum_{j=0}^{\delta-1} \delta^k_{\mathtt{KT}}(r_j,r_{j+1}) = 2\delta^k_{\mathtt{KT}}(r,r')$.

*Proof of* $(i)$. We have the following sequence of equalities:

$$\begin{aligned}
&\delta^k_{\mathtt{KT}}(r_j, r') - \delta^k_{\mathtt{KT}}(r_{j+1}, r') \\
&= |\{S \in \Delta^k : \mathtt{top}_{r_j}(S) \neq \mathtt{top}_{r'}(S)\}| - |\{S \in \Delta^k : \mathtt{top}_{r_{j+1}}(S) \neq \mathtt{top}_{r'}(S)\}| \\
&= |\{S \in \Delta^k : \mathtt{top}_{r_j}(S) = c_j \text{ and } \mathtt{top}_{r'}(S) = c'_j\}| \\
&\text{(because } r_j \text{ and } r_{j+1} \text{ only differ in the order of } c_j \text{ and } c'_j\text{)} \\
&= |\{S \in \Delta^k : \{c_j, c'_j\} \subseteq S \text{ and } S \setminus \{c_j, c'_j\} \subseteq \mathtt{Below}_{c'_j}(r_j)\}| \\
&\text{(because } c'_j \succ_{r'} c, \forall c \in \mathtt{Below}_{c'_j}(r_j), \text{ which implies that } \mathtt{Below}_{c_j}(r_j) \subseteq \mathtt{Below}_{c'_j}(r')\text{)} \\
&= |\{S \in \Delta^k : \mathtt{top}_{r_j}(S) \neq \mathtt{top}_{r_{j+1}}(S)\}| \\
&\text{(because } r_j \text{ and } r_{j+1} \text{ only differ in the order of } c_j \text{ and } c'_j\text{)} \\
&= \delta^k_{\mathtt{KT}}(r_j, r_{j+1}).
\end{aligned}$$

The result follows.

*Proof of* $(ii)$. As $r_j$ and $r_{j+1}$ only differ in the ranks of $c_j$ and $c'_j$, the difference $D_j - D_{j+1}$ is equal to:

$$\sum_{c \in \{c_j, c'_j\}} \left( \sum_{i=\min\{\mathtt{rk}(c,r_j), \mathtt{rk}(c,r')\}}^{\max\{\mathtt{rk}(c,r_j), \mathtt{rk}(c,r')\}-1} N_{k-2}^{m-i-1} - \sum_{i=\min\{\mathtt{rk}(c,r_{j+1}), \mathtt{rk}(c,r')\}}^{\max\{\mathtt{rk}(c,r_{j+1}), \mathtt{rk}(c,r')\}-1} N_{k-2}^{m-i-1} \right).$$

Note that $\mathtt{rk}(c_j, r_{j+1}) = \mathtt{rk}(c_j, r_j) + 1$ and $\mathtt{rk}(c'_j, r_{j+1}) = \mathtt{rk}(c'_j, r_j) - 1$ and $\mathtt{rk}(c'_j, r') \leq \mathtt{rk}(c'_j, r_{j+1})$. By swapping $c_j$ and $c'_j$ in $r_j$, the $k$-wise Spearman distance to $r'$ will thus decrease with respect to $c'_j$ by $N_{k-2}^{m-\mathtt{rk}(c'_j, r_j)}$. Regarding $c_j$, it may increase by $N_{k-2}^{m-\mathtt{rk}(c_j, r_j)-1}$ if $\max\{\mathtt{rk}(c_j, r_j), \mathtt{rk}(c_j, r')\} = \mathtt{rk}(c_j, r_j)$, or decrease by $N_{k-2}^{m-\mathtt{rk}(c_j, r_j)-1}$ otherwise. Hence, the largest possible decrease in $\delta^k_{\mathtt{S}}$ occurs in this latter case. We derive from this short analysis:

$$\begin{aligned}
D_j - D_{j+1} &\leq N_{k-2}^{m-\mathtt{rk}(c'_j, r_j)} + N_{k-2}^{m-\mathtt{rk}(c_j, r_j)-1} \\
&= 2N_{k-2}^{m-\mathtt{rk}(c'_j, r_j)} \text{ (as } \mathtt{rk}(c'_j, r_j) = \mathtt{rk}(c_j, r_j) + 1\text{)} \\
&= 2\delta^k_{\mathtt{KT}}(r_j, r_{j+1})
\end{aligned}$$

The last line results from the fact that, as stated previously, $N_{k-2}^{m-i-1}$ corresponds to the $k$-wise Kendall tau distance between a ranking and the ranking obtained by swapping candidates of ranks $i$ and $i+1$. □

The following result states that a consensus ranking for the $k$-wise Spearman distance can be computed in polynomial time in the numbers $n$ of voters and $m$ of candidates:

**Lemma 4.** *A consensus ranking for the $k$-wise Spearman distance can be computed in time $O(nm^3)$, for a preference profile with $n$ voters and $m$ candidates.*

*Proof.* Consider the complete bipartite graph with two independent sets $V_1$ and $V_2$, where:

- each vertex in $V_1$ corresponds to a candidate in $C$,
- each vertex in $V_2$ corresponds to a position in $\{1, \ldots, m\}$.

The edge $\{c, p\}$ between $c \in V_1$ and $p \in V_2$ is weighted by $\sum_{r' \in \mathcal{P}} \sum_{i=\min\{p, \mathtt{rk}(c,r')\}}^{\max\{p, \mathtt{rk}(c,r')\}-1} N_{k-2}^{m-i-1}$. Each perfect matching corresponds to a ranking $r$, where $\mathtt{rk}(c, r) = p$ if edge $\{c, p\}$ belongs to the matching. Determining a minimum weight matching in this graph thus amounts to solve the following minimization problem:

$$\min_r \sum_{c \in C} \sum_{r' \in \mathcal{P}} \sum_{i=\min\{\mathtt{rk}(c,r), \mathtt{rk}(c,r')\}}^{\max\{\mathtt{rk}(c,r), \mathtt{rk}(c,r')\}-1} N_{k-2}^{m-i-1}.$$

The two first sum operators can be swapped, which yields:

$$\min_r \sum_{r' \in \mathcal{P}} \sum_{c \in C} \sum_{i=\min\{\mathtt{rk}(c,r), \mathtt{rk}(c,r')\}}^{\max\{\mathtt{rk}(c,r), \mathtt{rk}(c,r')\}-1} N_{k-2}^{m-i-1} = \min_r \sum_{r' \in \mathcal{P}} \delta_{\mathrm{s}}^k(r, r').$$

Thus a minimum weight matching corresponds to a consensus ranking for the $k$-wise Spearman distance. Such a matching can be computed in time $O(|V_1|^3)$ by the Hungarian algorithm, thus in $O(m^3)$ as $|V_1| = m$. The computation of the complete bipartite graph itself is performed in $O(nm^3)$ (time required for computing the weights of edges). The overall complexity is therefore $O(nm^3)$. □

From Lemma 3, which implies that any consensus ranking for the $k$-wise Spearman distance is a 2-approximation of an optimal consensus ranking for the $k$-wise Kemeny rule, and Lemma 4, which establishes that a consensus ranking for the $k$-wise Spearman distance can be computed in $O(nm^3)$, we deduce the main result of this section:

**Theorem 5.** *There exists a 2-approximation algorithm for $k$-KAP which runs in $O(nm^3)$ time.*

Note that determining whether the bound 2 is tight remains an open problem (as is the case for $k = 2$, to our knowledge).

## 6. Numerical Tests

The numerical tests[2] we carried out had several objectives:

- to evaluate the computational performance of the dynamic programming approach of Section 3,
- to evaluate the impact of parameter $k$ on the set of consensus rankings,
- to assess the efficiency of the preprocessing technique of Section 4,
- to study the practical approximation ratio of the polynomial-time 2-approximation algorithm proposed in Section 5.

*Generation of preference profiles.* The preference profiles are generated according to the Mallows model [26], using the Python package PrefLib-Tools [27] in most tests (the PerMallows R package [28] was used for the tests of the 2-approximation algorithm). This model takes two parameters as input: a reference ranking $\sigma$ (the mode of the distribution) and a dispersion parameter $\phi \in (0, 1)$. Given these inputs, the probability of generating a ranking $r$ is proportional to $\phi^{\delta_{\text{KT}}(r,\sigma)}$. The more $\phi$ tends towards 0 (resp. 1), the more the preference rankings become correlated and resemble $\sigma$ (resp. become equally probable, i.e., we are close to the *impartial culture assumption*). This model enables us to measure in a simple way how the level of correlation in the input rankings impacts our results. In all tests, the number $n$ of voters is set to 50 and the ranking $\sigma$ is set arbitrarily as the $k$-wise Kemeny rule is neutral. For each triple $(m, k, \phi)$ considered, the results are averaged over 50 preference profiles.

*Practicability of the dynamic programming approach.* We first evaluate our dynamic programming approach on instances with different values for $m$ and $k$. Note that the computational performance measured here is not impacted by the level of correlation in the input rankings as it does not change the number of states in dynamic programming nor the computation time to determine the optimal value in each state. Hence, we only consider instances generated under the impartial culture assumption, i.e., with $\phi \approx 1$.

Table 3 (Rows 3-5) displays the average, max and min running times obtained for some representative $(m, k)$ values. As expected, the running times increase exponentially with $m$. Conversely, parameter $k$ seems to have a moderate impact on the running times. The dynamic programming approach enables us to solve $k$-KAP in a time of up to 3 sec. (resp. 76 sec.) for $m \leq 14$ (resp. $m \leq 18$).

[2]Implementation in C++, except for the polynomial-time 2-approximation algorithm implemented in Python3. Unless otherwise stated, all times are CPU seconds on an Intel Core I7-8700 3.20 GHz processor with 16GB of RAM.

*Influence of $k$ on the set of consensus rankings.* Second, we study the impact of $k$ on the set of optimal solutions to $k$-KAP. Indeed, one criticism for the Kemeny rule is that there exists instances for which the set of consensus rankings is compounded of many solutions which are quite different from one another. Thus, we investigate if increasing $k$ helps in mitigating this issue. For this purpose, we consider the same instances as before and compute the average number of consensus rankings denoted by $|\mathcal{R}^*|_{\texttt{avg}}$. The results are displayed in the sixth row of Table 3. Interestingly, this measure decreases quickly with $k$. For instance, when $m=18$, $|\mathcal{R}^*|_{\texttt{avg}}$ is divided by 5 when $k$ increases from 2 to 9 and is below 2 when $k=m$. The intuition is that $\delta^k_{\texttt{KT}}$ becomes more fine-grained as $k$ increases.

Table 3: Average, max and min CPU times in seconds of the dynamic programming approach of Section 3 for varying values of $m$ and $k$ (Rows 3 to 5). Average number of consensus rankings for increasing values of $m$ and $k$ (Row 6).

| $m$ | 6 | | | 10 | | | 14 | | | 18 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $k$ | 2 | 3 | 6 | 2 | 5 | 10 | 2 | 7 | 14 | 2 | 9 | 18 |
| Average time | <0.01 | <0.01 | <0.01 | 0.07 | 0.08 | 0.08 | 2.52 | 2.54 | 2.61 | 70.93 | 72.26 | 74.95 |
| Max time | <0.01 | <0.01 | <0.01 | 0.80 | 0.08 | 0.09 | 2.64 | 2.60 | 2.64 | 71.57 | 73.57 | 75.38 |
| Min time | <0.01 | <0.01 | <0.01 | 0.70 | 0.07 | 0.08 | 2.49 | 2.49 | 2.57 | 70.27 | 71.91 | 74.33 |
| $\lvert\mathcal{R}^*\rvert_{\texttt{avg}}$ | 3.00 | 1.20 | 1.05 | 3.84 | 1.24 | 1.10 | 5.36 | 2.36 | 1.16 | 19.70 | 4.12 | 1.47 |

*Impact of the 3-wise majority graph.* We now study the impact of the preprocessing method proposed in Section 4 for $k = 3$. This preprocessing uses the $k$-wise majority digraph to divide $k$-KAP into several subproblems which can be solved separately by dynamic programming. Hopefully, when voters' preferences are correlated (i.e., for "small" $\phi$ values), these subproblems become smaller and more numerous, making the preprocessing more efficient. The results are shown in Table 4, where the results obtained without preprocessing are also given in the last column. The obtained running times are highly dependent on $\phi$. For instance, with $m=18$, the average running time for solving 3-KAP is above 1 minute if $\phi = 0.95$ while it is below 1 second if $\phi \leq 0.85$. This gap is necessarily related to the preprocessing step, since $\phi$ has no impact on the running time of the dynamic programming approach. To explain this significant speed-up, we display in Table 5 the average size of the largest SCC of the 3-wise majority digraph at the end of the preprocessing step. Unsurprisingly, this average size turns out to be correlated with $\phi$: when $\phi \leq 0.5$, the size of the largest SCC is almost always 1. Hence, the preprocessing step is likely to yield directly a consensus ranking. In contrast, when $\phi=0.95$, the average size of the largest SCC is close to $m$, thus the impact of the preprocessing is low.

Table 4: Average, max and min CPU times (in seconds) for the 3-wise Kemeny rule with preprocessing.

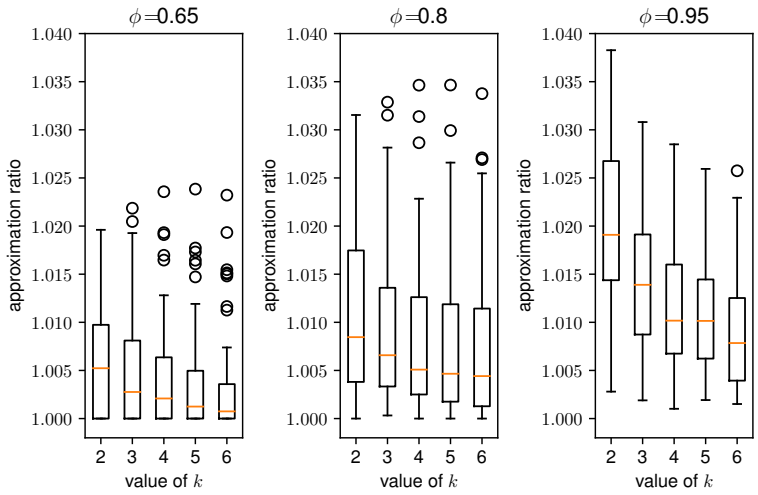
| $m$ | $\phi$ | 0.5 | 0.8 | 0.85 | 0.9 | 0.95 | w/o preproc. |
|---|---|---|---|---|---|---|---|
| 6 | Avg time | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 |
| | Max time | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 |
| | Min time | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 | <0.01 |
| 10 | Avg time | 0.03 | 0.03 | 0.04 | 0.07 | 0.10 | 0.07 |
| | Max time | 0.03 | 0.03 | 0.10 | 0.15 | 0.17 | 0.08 |
| | Min time | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.07 |
| 14 | Avg time | 0.09 | 0.09 | 0.11 | 0.94 | 2.21 | 2.52 |
| | Max time | 0.09 | 0.13 | 0.25 | 3.14 | 3.26 | 2.59 |
| | Min time | 0.08 | 0.08 | 0.09 | 0.10 | 0.26 | 2.49 |
| 18 | Avg time | 0.20 | 0.20 | 0.55 | 14.87 | 61.72 | 71.17 |
| | Max time | 0.31 | 0.21 | 8.46 | 79.87 | 80.11 | 71.61 |
| | Min time | 0.19 | 0.19 | 0.19 | 0.22 | 6.02 | 71.02 |

Table 5: Average size of the largest SCC after preprocessing.

| $m\backslash\phi$ | 0.47 | 0.81 | 0.85 | 0.88 | 0.95 |
|---|---|---|---|---|---|
| 6 | <1.10 | 1.84 | 1.88 | 2.72 | 3.28 |
| 10 | <1.10 | 1.64 | 3.28 | 5.32 | 8.20 |
| 14 | <1.10 | 2.68 | 3.84 | 9.12 | 12.91 |
| 18 | <1.10 | 2.84 | 4.27 | 9.80 | 17.44 |

*Polynomial-time 2-approximation algorithm.* Lastly, we study the practical approximation ratio of the proposed polynomial-time 2-approximation algorithm. The results are averaged over 50 instances randomly generated according to the Mallows model, for 50 voters and 12 candidates. Preliminary tests not reported here tended to show that this approximation ratio does not depend on the number of candidates. Furthermore, the approximation ratio becomes better when the number of voters increases which can be explained by the fact that the law of large numbers make dominant the reference ranking in the Mallows model, regardless of the aggregation rule. We report in Figure 3 the box plots obtained by varying the values of $k$ and $\phi$ (the closer to 1, the closer to the impartial culture assumption). Note that the CPU time to compute a consensus ranking for the $k$-wise Spearman distance (for 50 voters and 12 candidates) is below 12 milliseconds whatever the values of $k$ and $\phi$, using the `linear_sum_assignment` function of the SciPy Python library on an Intel Core i5 2.3 GHz dual core processor with 8GB of RAM. It is interesting to observe that the practical approximation ratio is much better than 2: the worst ratio over all generated instances is 1.04. For the same reason as for the number of voters, the more correlated the preferences in the profile, the better the approximation ratio. Furthermore, the greater the value of $k$, the better the approximation ratio, as is clear from the curves obtained.

Figure 3: Practical approximation ratio of the 2-approximation algorithm.




## 7. Conclusion

In this paper, we advocate using the results of *setwise* contests between candidates to design social welfare functions that are less myopic than those only based on pairwise comparisons. One natural such social welfare function is a $k$-wise generalization of the Kemeny rule, which returns a ranking minimizing the number of disagreements on top candidates of sets of cardinality lower than or equal to $k$. We have studied this $k$-wise Kemeny rule from both axiomatic and algorithmic viewpoints. In more detail, we established that determining a consensus ranking is NP-hard for any $k \geq 3$. Then, after proposing a dynamic programming procedure, we have investigated a $k$-wise variant of the majority graph, from which we developed a preprocessing step. Computing this graph is a polynomial time problem for $k = 3$ but becomes NP-hard for $k \geq 4$. The numerical tests show the practicability of the approach for up to 18 candidates. Lastly, we have designed a 2-approximation algorithm for the $k$-wise Kemeny aggregation problem. This approximation algorithm in fact returns a ranking minimizing a $k$-wise variant of the Spearman distance, and the worst case approximation ratio is then derived from an adaptation of the Diaconis-Graham inequality. The numerical tests suggest that, in practice, the $k$-wise Kemeny score of the returned ranking is often much better than a 2-approximation of the optimal score.

Several research directions could be further investigated. One of them would be to investigate the complexity of determining a consensus ranking for $\delta^k_{\mathtt{KT}}$ when $k = m$, because our hardness result only holds for *fixed* values of $k$. Secondly, note that $\delta^k_{\mathtt{KT}}$ focuses on "small" sets as we count disagreements on sets of size lower than or equal to $k$. An opposite

viewpoint would be to consider "large" sets by counting disagreements on sets of size greater than or equal to $m - k$. Note that for $k = 0$ this would lead to a rule similar to plurality. Another direction is to study the complexity of recognition problems related to the $k$-wise Kemeny rule [29], i.e., deciding if a given ranking is a consensus ranking for the $k$-wise Kemeny rule. Alternative definitions of $k$-wise majority graphs that are easier to compute for $k > 3$ are also worth investigating. Finally, other social welfare functions based on the results of setwise contests are worth investigating in our opinion, both from the axiomatic and the computational points of view.

## Acknowledgements

We wish to thank the anonymous reviewers for their useful comments that helped us to improve the presentation of the paper. This work has been partially supported by project ANR-14-CE24-0007-01 "CoCoRICo-CoDec" and the Italian MIUR PRIN 2017 project ALGADIMAR "Algorithms, Games, and Digital Markets".

## References

[1] H. Gilbert, T. Portoleau, O. Spanjaard, Beyond pairwise comparisons in social choice: A setwise kemeny aggregation problem, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020, 2020, pp. 1982–1989.

[2] H. Moulin, Axioms of cooperative decision making, number 15 in Econometric Society Monographs, Cambridge University Press, 1991.

[3] W. Cheng, E. Hüllermeier, A new instance-based label ranking approach using the mallows model, in: Proceedings of the 6th International Symposium on Neural Networks, ISNN 2009, Wuhan, China, May 26-29, 2009, Springer, 2009, pp. 707–716.

[4] S. Clémençon, A. Korba, E. Sibony, Ranking median regression: Learning to order through local consensus, in: Proceedings of the 29th international conference on Algorithmic Learning Theory, ALT 2018, Lanzarote, Canary Islands, Spain, April 7-9, 2018, 2018, pp. 212–245.

[5] S. Wang, J. Sun, B. J. Gao, J. Ma, Vsrank: A novel framework for ranking-based collaborative filtering, ACM Transactions on Intelligent Systems and Technology (TIST) 5 (2014) 51.

[6] B. N. Jackson, P. S. Schnable, S. Aluru, Consensus genetic maps as median orders from inconsistent sources, IEEE/ACM Transactions on computational biology and bioinformatics 5 (2008) 161–171.

[7] R. Fagin, R. Kumar, D. Sivakumar, Efficient similarity search and classification via rank aggregation, in: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, San Diego, California, USA, June 9-12, 2003, 2003, pp. 301–312.

[8] A. Altman, M. Tennenholtz, Axiomatic foundations for ranking systems, Journal of Artificial Intelligence Research 31 (2008) 473–495.

[9] C. Dwork, R. Kumar, M. Naor, D. Sivakumar, Rank aggregation methods for the web, in: Proceedings of the 10th international conference on World Wide Web, WWW 2001, Hong Kong, China, May 1-5, 2001, 2001, pp. 613–622.

[10] K. J. Arrow, A difficulty in the concept of social welfare, Journal of political economy 58 (1950) 328–346.

[11] P. C. Fishburn, Condorcet social choice functions, SIAM Journal on applied Mathematics 33 (1977) 469–489.

[12] W. S. Zwicker, Introduction to the theory of voting, in: F. Brandt, V. Conitzer, U. Endriss, J. Lang, A. D. Procaccia (Eds.), Handbook of Computational Social Choice, Cambridge University Press, 2016, pp. 23–56.

[13] K. A. Baldiga, J. R. Green, Assent-maximizing social choice, Social Choice and Welfare 40 (2013) 439–460.

[14] T. Lu, C. Boutilier, The unavailable candidate model: a decision-theoretic view of social choice, in: Proceedings of the 11th ACM conference on Electronic Commerce, EC 2010, Cambridge, Massachusetts, USA, June 7-11, 2010, 2010, pp. 263–274.

[15] J. G. Kemeny, Mathematics without numbers, Daedalus 88 (1959) 577–591.

[16] W. S. Zwicker, Cycles and intractability in a large class of aggregation rules, Journal of Artificial Intelligence Research 61 (2018) 407–431.

[17] R. Kumar, S. Vassilvitskii, Generalized distances between rankings, in: Proceedings of the 19th international conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010, 2010, pp. 571–580.

[18] H. Young, An axiomatization of Borda's rule, Journal of Economic Theory 9 (1974) 43–52.

[19] H. P. Young, A. Levenglick, A consistent extension of condorcet's election principle, SIAM Journal on applied Mathematics 35 (1978) 285–300.

[20] V. Conitzer, M. Rognlie, L. Xia, Preference functions that score rankings and maximum likelihood estimation, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, July 11-17, 2009, 2009, pp. 109–115.

[21] J. Bartholdi, C. A. Tovey, M. A. Trick, Voting schemes for which it can be difficult to tell who won the election, Social Choice and welfare 6 (1989) 157–165.

[22] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, F. A. Rosamond, Fixed-parameter algorithms for Kemeny rankings, Theoretical Computer Science 410 (2009) 4554–4570.

[23] I. Charon, O. Hudry, An updated survey on the linear ordering problem for weighted or unweighted tournaments, Annals of Operations Research 175 (2010) 107–158.

[24] R. M. Karp, Reducibility among combinatorial problems, in: Complexity of computer computations, Springer, 1972, pp. 85–103.

[25] P. Diaconis, R. L. Graham, Spearman's footrule as a measure of disarray, Journal of the Royal Statistical Society: Series B (Methodological) 39 (1977) 262–268.

[26] C. L. Mallows, Non-null ranking models. I, Biometrika 44 (1957) 114–130.

[27] N. Mattei, T. Walsh, Preflib: A library of preference data HTTP://PREFLIB.ORG, in: Proceedings of the 3rd international conference on Algorithmic Decision Theory, ADT 2013, Bruxelles, Belgium, November 13-15, 2013, 2013, pp. 259–270.

[28] E. Irurozki, B. Calvo, J. A. Lozano, et al., Permallows: An R package for Mallows and generalized Mallows models, Journal of Statistical Software 71 (2016) 1–30.

[29] O. Hudry, Complexity of computing median linear orders and variants, Electronic Notes in Discrete Mathematics 42 (2013) 57–64.