Axiomatic and computational aspects of discrete optimization problems in collective settings : from Multi-Agent Scheduling to Participatory Budgeting

Martin Durand


École doctorale nº 130 : Informatique, Télécommunications et Electronique
Sorbonne Université, Paris

Axiomatic and computational aspects of discrete optimization problems in collective settings

From Multi-Agent Scheduling to Participatory Budgeting

Thèse de doctorat en informatique
présentée publiquement par

Martin DURAND

le 23 Octobre 2023

Rapporteurs
Alessandro AGNETIS — Professor, University of Sienna, Italie
Jérôme LANG — Directeur de recherche CNRS
LAMSADE, Université Paris-Dauphine, France

Examinateurs
Nadia BRAUNER — Professeure, G-SCOP, France, Présidente du jury
Nicolas MAUDET — Professeur, LIP6, Sorbonne Université, France
Arianna NOVARO — Maîtresse de conférences, Université Paris I Panthéon-Sorbonne, France

Directrice de thèse
Fanny PASCUAL — Maîtresse de conférences, HDR,
LIP6, Sorbonne Université, France

Table of contents

To those who make me happy.

Acknowledgments

I would like, first and foremost, to thank my PhD advisor Fanny Pascual. I had the chance of meeting her a few years ago during my Bachelor's degree as a teacher in the algorithmic course, alongside with Olivier Spanjaard. From all the courses I was following back then, this one was my breath of fresh air. Both lectures and exercises sessions interested me a lot and I quickly became more and more fascinated with algorithms and their theoretical aspects. Following this path, I entered the Master's degree ANDROIDE, focused on operations research, decision, multi-agent systems and robotics. During this degree, my interest was focused on computational complexity and collective decision making. After working on a research project, supervised by Fanny, I asked her if she could supervise me for an internship, then a second a few months later and finally a PhD thesis. At every step of the way Fanny has been an excellent teacher and a benevolent supervisor.

By accepting to supervise me for this PhD thesis, Fanny allowed me enter the world of academia, a world in which I blossom and in which I feel deeply happy. For this, I will always be thankful to her. During the thesis we managed to find the proper rhythm, despite the lock-downs, the amount of time spent on administrative tasks and teachings. I always had enough freedom to explore the topics I enjoyed and always had enough guidance not to get lost. She has been kind, open-minded, patient, rigorous, hard-working, she had the right intuitions and always gave helpful feedback. For all these reasons, I always knew, going into a meeting, that it would be both an enjoyable and teaching moment.

I wish Fanny the best for the years to come and I will always be grateful to her.

I would also like to thank Olivier Spanjaard who, like Fanny, was first my teacher both in Bachelor's and Master's degree. He co-supervised, with Fanny, my Master's degree 6 months internship. During this time, as well as later, during the PhD thesis, it has been a pleasure working with him on research as well as teaching. Beyond this, he has always been available and helpful whenever I had an issue as well as cheerful in the day-to-day life.

I would also like to thank all the professors from the Master's degree ANDROIDE. I learnt a lot during these two years, and this is due to their skills as teachers as well as their dedication to build courses they are passionate about. This passion passes on. I think this thesis shows that I took a lot from all these lectures from the most theoretical to the most practical. Thank you to Pierre Fouillhoux, Patrice Perny, Bruno

Escoffier, Pierre-Henri Wuillemin, Evripidis Bampis, Thibaut Lust, Nicolas Maudet, Aurélie Beynier and Safia Kedad-Sidhoum.

More generally, it has always been a pleasure to work in LIP6, it is a very welcoming environment. In particular, the members of the Operations Research and the Decision teams have been my colleagues for 3 years now and the atmosphere was always cheerful. Thank you to Carola, Martin, Niklas, Koen, Georgii, Mara, Maria Laura, Anja and Alexis for sharing a part of this journey with me.

There are three colleagues I would like to thank just a little bit more, hopefully the others will not get jealous. Magdalena have been my office mate for several years. We worked on topics that are quite close and every single discussion I had with her regarding research has been illuminating. She is deeply passionate about her research and about the computational social choice field as a whole. She is nice, inspiring and unique in so many ways. She also gave me plenty of advice for the redaction of this manuscript as well as the template I used for this document (and this is one of the most valuable gift for a PhD student !).

I would also like to thank my two fellow PhD student companions François and Océane. We have been colleagues for two years but, beyond that, we spent a lot of time together outside of the office. François pushed me to do sports again, which is not an easy task, and Océane shared a little bit of my passion for movies. They have been funny, understanding and supportive friends and I will remember kindly the time I spent with them during these years.

I would also like to thank Déborah and Kévin for being supportive during this journey. They are two of my closest friends and, despite the circumstances, they have always been available and cheerful as they always had been.

Finally, I want to thank my family for being supportive, in all possible ways, especially my parents for hosting me and bearing with me for 3 additional years, which once more, is not an easy task ! I would like to thank my sister as well, she has been a role model for me for quite some time now.

All these persons, and many others, have been a part of my life during this journey. I have no regret about these three years, if I had to do it all over again, I would do it all over again without changing anything, thank you.

Introduction

This thesis focuses on several collective decision making problems, from multi agent scheduling to participatory budgeting. There are several agents, that can represent companies, citizens of a city, members of a research lab ..., for which a common solution has to be found. Such a solution can be a schedule of tasks of interest for the agents, a ranking of items that the agents have to sort or a selection of projects approved by the agents. Each agent has different interest over the possible solutions. This can be because the solution impacts directly the agents or because the agents express preferences over the possible solutions. Any solution can be evaluated thanks to different tools. We will mostly focus on fairness and efficiency. Fairness and efficiency can be formulated in different ways, depending on the context, from objective functions to binary properties. In all cases, our goal will be to find a solution that corresponds as much as possible to the interests or preferences of the agents. A solution is collectively satisfying if it is "close" to the preferences of the agents, according to some definition of closeness, or if the overall benefit of the agents is high. The solution should also be fair in the sense that no agent should be treated better than any other. We study different problems, especially scheduling problems, in which we have to find fair solution or fair decision making processes while guaranteeing some notion of efficiency.

Presentation of the chapters. This document is divided into seven chapters.

- In Chapter 1, we present both the scheduling and the computational social choice fields. We aim at introducing models and resolution concepts relevant to this thesis. We conclude the chapter with a short review of some multi-agent scheduling problems.
- Chapter 2 is dedicated to the Multi-Organization Scheduling Problem. It is a scheduling problem in which several organizations (or agents) have tasks and machines. Each organization has a "local" schedule in which it schedules its own tasks on its own machines. We consider that the organizations collaborate by sharing their machines in order to improve the quality of their solution. The goal is to find a schedule of all the tasks on all the machines (a task can be scheduled on a machine owned by another organization) which satisfies all the organizations. Our objective here is to study the tradeoff between efficiency, in terms of global performance, and fairness. Regarding fairness, we will at first consider a

rationality constraint which requires that, when the machines are shared among the organizations, each organization has a solution at least as satisfying as its local schedule. In other words, an organization cannot loose anything by sharing. This constraint ensures that organizations have an incentive to collaborate, but fulfilling it can impact the efficiency of the solution and our goal is to understand to which extent. In a final part, we will consider fairness as a main objective and formulate a new problem, by trying to find solutions not only fulfilling the rationality constraint but in which each organization gains as much as possible.

- Chapters 3 and 4 focus on the Collective Schedules problem. The collective schedules problem consists in computing a schedule of tasks shared between individuals. Individuals have preferences over the order of the shared tasks. This problem has numerous applications since tasks may model public infrastructure projects, events taking place in a shared room, or work done by co-workers. Our aim is, given the preferred schedules of individuals (voters), to return a consensus schedule.

  In Chapter 3, we propose an axiomatic study of the collective schedule problem, by using classic axioms in computational social choice and new axioms that take into account the duration of the tasks. We show that some axioms cannot be fulfilled by the same rule, and we study the axioms fulfilled by four rules: one which has been studied in the seminal paper on collective schedules [Pascual et al., 2018], one which generalizes the Kemeny rule, one which generalizes Spearman's footrule, and one which relies on a scheduling approach. From a computational point of view, we show that three of these rules solve NP-hard problems, but that it is possible to solve optimally these problems for small but realistic size instances, and we give an efficient heuristic for large instances. We conclude this chapter with experiments evaluating the quality of the heuristic and the computation time of the four rules.

  In Chapter 4, we will consider the setting in which all the tasks have the same length. We study several algorithms taking preferences as parameters and returning a collective solution. These algorithms are based on two main criteria, each one extending a classic scheduling criterion: a distance criterion and a binary criterion. These algorithms return a solution minimizing either the binary or the distance criterion. This work focuses on classic scheduling constraints, namely the release dates, the deadlines and precedence constraints. We will consider two settings, one in which preferences fulfill the constraints, and another one in which they do not. In both cases the goal is to study the complexity and the mathematical properties of the algorithms. Finally, we study a fast heuristic algorithm for a special case of our problem with regards to its approximation ratio for the distance criterion and whether or not the schedule returned by this heuristic fulfills the scheduling constraints.

- In Chapter 5 we study the preference aggregation problem with a probabilistic approach. A set of v voters express preferences over a set of n candidates. We

make the hypothesis that there exists a ground truth ranking, i.e. an objective way of ranking the candidates. The voters have a noisy perception of this ground truth and express their perception via their votes. In this chapter, we call "model" a probabilistic model which represents the noise. This model associates a conditional probability to each preference.We study in this chapter a non-utilitarian discrete choice model for preference aggregation and its application to voting.We propose an exact and a heuristic algorithm to compute the best estimation of the ground truth ranking according to our model. Numerical tests are presented to assess the efficiency of these algorithms, and measure the model fitness on synthetic and real data.

- Chapter 6 is dedicated to the Participatory Budgeting problem. In this problem, the objective is to select a set of projects that fits in a given budget. Voters express their preferences over the projects and the goal is then to find a consensus set of projects that does not exceed the budget. The aim of this chapter is to introduce models and algorithms for the Participatory Budgeting problem when projects can interact with each other. Our goal is to detect such interactions thanks to the preferences expressed by the voters. Through the projects selected by the voters, we detect positive and negative interactions between the projects by identifying projects that are consistently chosen together. In presence of project interactions, it is preferable to select projects that interact positively rather than negatively, all other things being equal. We introduce desirable properties that utility functions should have in presence of project interactions and we build a utility function which fulfills the desirable properties introduced. We then give axiomatic properties of aggregation rules, and we study three classical aggregation rules: the maximization of the sum of the utilities, of the product of the utilities, or of the minimal utility. We show that in the three cases the problems solved by these rules are NP-hard, and we propose a branch and bound algorithm to solve them. We conclude this chapter with experiments.

- Finally, Chapter 7 concludes this thesis and presents several research perspectives for each of the previous chapters.

Chapter 1

Preliminaries

This chapter aims at introducing basic notions and notations used throughout this thesis. Firstly, we introduce scheduling: we define what a scheduling problem is, present several resolution concepts and mention some algorithms to solve such problems. The problems we will study in Chapters 2, 3 and 4 are scheduling problems. Secondly, we introduce computational social choice and focus on the voting problem: we give an overview of different approaches and present voting rules as well as tools to evaluate such rules. The voting problem is central to this thesis and we will use the different approaches introduced in this chapter in Chapters 3 to 6. We conclude this chapter with a short review of several multi agent scheduling problems.

We start by introducing the scheduling field.

1.1 An introduction to scheduling

What is scheduling ? Scheduling [Błażewicz et al., 2001; Brucker, 2010; Pinedo, 2012] is a decision-making process dealing with the allocation of tasks to a set of given resources, also called machines. Scheduling problems represent many real life situations. Machines can represent any resource in a production or logistic process: machines in a workshop, airstrips in an airport, processing units in a computing environment, crew members in a company and so on. Tasks may represent steps in a production process, take-off and landings in an airport, execution of programs and so on. Each of the task has a processing time, namely an amount of time needed by a machine to process the task. A schedule is then an allocation of each task on a machine, each task having a starting time and a completion time. A task cannot start on a machine if the machine is processing another task at the same time. Scheduling is both a very active research area in operations research and a very well-implanted one since we can find publications from the 1950s (e.g. [Bellman, 1956]). At that time, scheduling was mostly about optimizing production lines in factories. However as previously mentioned, it can model a very wide range of real life problems.

Example 1.1.1: Gate affectation in an airport

We can represent the affectation of planes to gates in an airport as a scheduling problem. Each gate is represented by a machine. Each boarding and disembarkation of a plane is a task. Depending on the number of passengers, the duration of each of these steps can vary and therefore the tasks may have different processing times. The problem then consists in affecting tasks to machines such that no machine processes two tasks at the same time, since multiple planes cannot be affected to one gate.

We introduce a first set of notations. Scheduling deals with a set of m *machines*. There is a set $\mathcal{J} = \{t_1, \ldots, t_n\}$ of n *tasks*. Each task t_i has a *processing time* p_i, i.e. the amount of time needed by a machine to process task t_i. Given a schedule S, the *completion time* of a task t_i in schedule S, i.e. the time at which the processing of task t_i ends, is denoted by $C_i(S)$.

One common way of representing schedules is to use a Gantt chart. In such a chart, each row represents a machine, tasks represented on the same row are affected to the same machine. The x axis represents time, task represented on the left are processed before tasks represented on the right.

Example 1.1.2: Gantt chart

If we consider a very simple example in an airport having 3 gates $\{A, B, C\}$ and dealing with 10 flights $\{F1, F2, F3, F4, F5, F6, F7, F8, F9, F10\}$, either arriving at the airport or leaving from the airport, in a 12 hours time period. We can represent a potential schedule of the 10 flights on the 3 gates as follows:

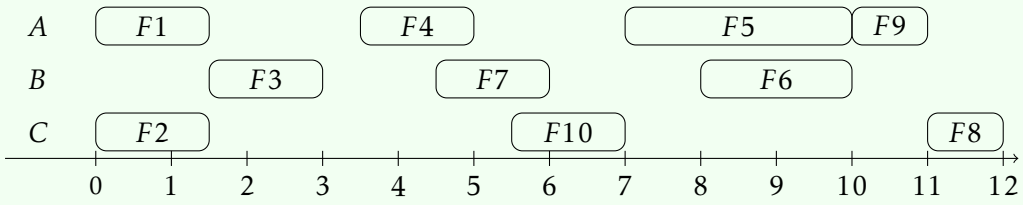


Figure 1.1: Gantt chart of a schedule

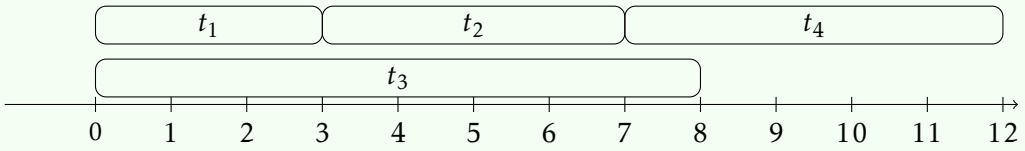
The schedule S represented in Figure 1.1 affects, for example, task $F6$ to the machine B between time 8 and time 10 and therefore $C_{F6}(S) = 10$, while task $F8$ is affected to machine C between time 11 and time 12, thus $C_{F8}(S) = 12$.

The makespan of a schedule S is the completion time of the last task to be processed in S, it is usually denoted by $C_{\max}(S)$. It represents the time from which the whole workload has been processed and all the machines are free. As an example, if tasks represent steps of a project, the makespan represents the time at which the project is

fully completed.

Example 1.1.3: Makespan

The makespan is very easy to read on a Gantt chart. Let us look at the following schedule S of 4 tasks on 2 machines.



It is easy to see that the task with the maximum completion time is t_4 with $C_4(S) = 12$, the makespan of S is then 12.

Constraints. In Example 1.1.2, it is not possible to schedule the tasks in any given way. Indeed, it is impossible to schedule a disembarkation before the plane actually reaches the airport or to schedule a boarding after the departure of the plane. Scheduling problems handle such situations with constraints, which can take different forms. The goal is then to find an allocation of the tasks on the machines fulfilling a set of constraints, such a schedule is called a *feasible schedule*. Some of the classic constraints include *release date*, i.e. the time from which it is possible to schedule a task and *deadline*, i.e. the last possible time a task can be completed at. The release date of task t_i is denoted by r_i while $\overline{d}_i$ denotes its deadline.

It is also possible to have a precedence relation P between tasks, $t_a P t_b$ means that task t_a has to be scheduled before task t_b, i.e. the starting time of task t_b has to be greater than or equal to the completion time of task t_a. This precedence relation over the set of tasks $\mathcal{J}$ is irreflexive, asymmetric and transitive.

Observation 1.1.1: Quick reminder

- Irreflexivity means that we cannot have $t_a P t_a$, i.e. the precedence relation cannot force to schedule a task before itself.
- Asymmetry means that if we have $t_a P t_b$ we have $\neg t_b P t_a$, i.e. if we need to schedule t_a before t_b then it is impossible that t_b has to be scheduled before t_a.
- Transitivity means that if we have $t_a P t_b$ and $t_b P t_c$ then we have $t_a P t_c$, i.e. if task t_a has to be scheduled before task t_b and task t_b has to be scheduled before task t_c then task t_a has to be scheduled before t_c.

Because the precedence relation fulfills these three conditions, it is a strict partial order.

This relation can be represented with a precedence directed graph in which each vertex i represents a task t_i and there is an arc going from vertex i to vertex j if task t_i has to be scheduled before task t_j. Since the relation is transitive, irreflexive and asymmetric, this directed graph does not contain any cycle.

Objectives. Some problems only require to find a feasible schedule (which can already be a NP-hard problem), while some others aim at finding a feasible schedule which optimizes some function. There is a wide range of different objective functions: minimizing the makespan, minimizing the sum of the completion times of the tasks, minimizing the delay between an expected due date for a task and its actual completion time .... The objective functions we study in this thesis rely on two main notions the makespan and the due dates.

Finding a schedule which minimizes the makespan is a NP-hard problem, even if there are only 2 machines [Garey and Johnson, 1979]. However, there are numerous polynomial time algorithms returning schedules which have a makespan very close to the optimal reachable makespan, as we will see later.

The second class of optimization functions that we will study are based on the notion of *due date*. We say that a task t_i has a due date d_i if it is expected to complete at time d_i. Note that contrarily to a deadline, the task does not have to be completed by time d_i, however if the task is not completed at time d_i, there will be a penalty. A common example to illustrate this notion is delivery. If a company is supposed to deliver a given product to a client by a given time, it is usually not a strict deadline, in the sense that it could be delivered later. However this delay can imply discounts on the price of the product or other less visible costs, like lower chances that the client will contact the company again. The objective functions we will use are derived from several usual scheduling optimization functions [Brucker, 2010]:

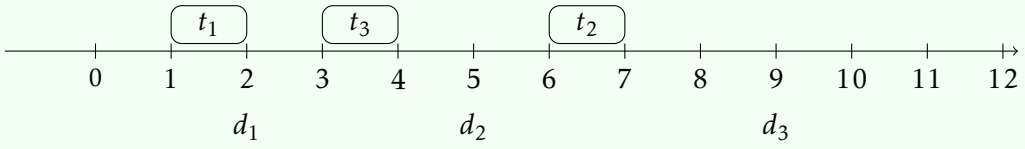
- Tardiness (T): the tardiness measures how late a task is in comparison to its due date. The tardiness $T_j(S)$ of a task t_j in a schedule S is defined as follows: $T_j(S) = \max\{0, C_j(S) - d_j\}$.
- Deviation (D): the deviation measures how far from its due date a task is, whether before or after. The deviation $D_j(S)$ of a task t_j in a schedule S is defined as follows: $D_j(S) = |C_j(S) - d_j|$.
- Unit time penalty (U): the unit time penalty counts 1 if the task is executed after its due date, 0 otherwise. It is defined as follows: $U_j(S) = \mathbb{1}_{C_j(S)>d_j}$.

These three functions corresponds to different interpretation of the due dates. The tardiness fits with the example mentioned above, we penalize tasks being late and the later the task, the higher the penalty. The unit time penalty is a binary criterion in the sense that it only measures if a task is late or not, it does not matter by how much. This is the case when executing a task after its due date can be done but is useless, even if it is done just after the due date. The deviation also penalizes tasks for being early. This

allows us to model situations in which scheduling a task earlier than its due date is costly. For example, imagine the due date being the time at which a company plans to send manufactured goods. Then it may want the goods to be produced as close to the date as possible, since producing them earlier means that they have to be stored, which can have a cost and also takes space in warehouses.

Example 1.1.4: Due date criteria

Let us consider an instance with 3 tasks $\{t_1, t_2, t_3\}$, each of processing time 1 and each having a due date. Task t_1 has a due date $d_1 = 2$, task t_2 has a due date $d_2 = 5$ and task t_3 has a due date $d_3 = 9$. Let us now consider a schedule S of the three tasks on one machine:



Task t_1 completes exactly at its due date, so both its tardiness, deviation and unit time penalty are 0. Task t_2 completes at time 7, which his higher than its due date $d_2 = 5$. Its deviation and tardiness are of 2 and its unit time penalty is 1. Task t_3 completes before its due date, its tardiness and unit penalty are 0 while its deviation is 5 because it is completed 5 units of time earlier than its due date.

Graham notation. As mentioned above the number of scheduling problems is very large and there are a lot of variations in each problem: How many machines are there ? Are all the machines identical ? Do the tasks have release dates ? Do all the tasks have the same processing time ? Do we want a schedule satisfying constraints or are we looking for a schedule minimizing the makespan, or some cost function ? Graham et al. [1979] introduced a convention on scheduling problems, they can be denoted using the following notation: $(\alpha|\beta|\gamma)$.

Component α. The α gives information on the machines. It can be a number, 1 for example indicates that there is one single machine. It can also be a letter, indicating some property on the machines. For example P indicates that the machines are identical. This letter can be associated with a number, indicating the number of machines, $P2$ means that there are 2 identical machines. If no number is specified, then the number of machines m is not fixed and is a parameter of the problem. In this thesis, we will consider two contexts regarding machines 1 and P, in other words, we will either look at scheduling problems with one machine or with a set of identical machines.

Component β. The β gives information on the constraints or specificity of the set of tasks. Constraints like release dates and deadlines are specified here by indicating r_i or $\overline{d}_i$. Precedence constraints are specified by adding "prec" in the β section. There are

different types of precedence corresponding to more or less restricted forms of precedence graph from "chains" or "tree" to any type of graph. If the tasks have a given characteristic, for example if all the tasks have the same processing time, it is indicated by adding $p_i = p$. If they all have a processing time of exactly 1, we add $p_i = 1$. The scheduling problems we will study will either have no constraints at all or the constraints mentioned above, precedence, release dates and deadlines. We will consider in Chapter 4 a situation in which all tasks have a unit processing time.

Component γ. Finally, the γ part indicates the optimization function. Looking for any feasible solution is indicated by a dash "-" or a star "*". The makespan objective is represented by $C_{\max}$ while the objective consisting in minimizing the sum of the completion time of all the tasks is denoted by ΣC_i. The objectives for the total tardiness, deviation and unit time penalty are denoted by ΣT_j, ΣD_j, ΣU_j, meaning that we look for a schedule minimizing the sum of the tardiness, deviation or unit time penalty over all the tasks. These objective functions imply that tasks also have due dates so this is not indicated in the β part of the notation.

In this thesis we will sometimes use Graham notation to refer to known problems, in particular $(P||C_{\max})$ but also problems like $(1||\Sigma D_j)$ or $(1|p_j = 1, chains|\Sigma U_j)$ for example.

Algorithms. We complete this introduction to scheduling by reviewing a few common scheduling algorithms that we will use later. Firstly, it is interesting to notice that a lot of scheduling problems are known to be NP-hard. In such cases, in order to solve these problems exactly, a few common methods are used like constraint programming, linear programming or branch and bound methods. These are classic tools allowing to model a lot of different problems (not necessarily in scheduling) and to solve them optimally. Depending on the problem, these methods may be more or less efficient and in some cases these exact resolution methods cannot be run in reasonable time even when the instances have a small number of tasks and machines. In such cases, we have to use other algorithms.

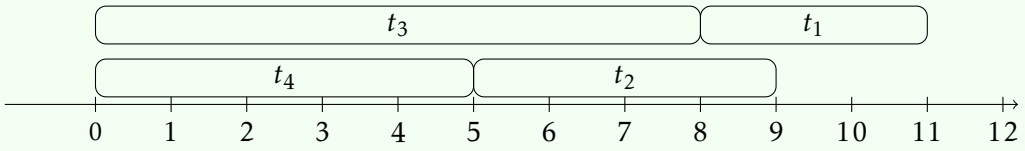
For some problems, it is also possible to use α-approximate algorithms. These algorithms do not solve the problems exactly but return a solution which is guaranteed not to be too bad in comparison to an optimal solution. For a minimization problem, like the makespan minimization problem, an algorithm is said to be α-approximate with regards to an objective function if it always return a schedule for which the objective value is at most α times the objective value for the optimal solution. A polynomial algorithm which is α-approximate for an NP-hard problem can be a good alternative to an exponential algorithm if the time available for the computation of a solution is limited and if α is not too large.

A first, very common, class of greedy scheduling algorithm are the list scheduling algorithms. The idea is to sort the tasks according to some criterion and then to schedule the tasks greedily in the given order, i.e. as soon as a machine is available, we schedule the first task in the list. Such algorithms are very fast to compute, as long as the criterion used to sort the tasks is simple to compute. They are also performing very well for several problems. For example, the algorithm scheduling tasks by in-

creasing processing time, also called SPT for Shortest Processing Time, is optimal for the problem $(P||\Sigma C_j)$. We now show a quick example of the scheduling algorithm LPT, standing for Longest Processing Time, which consist in sorting the tasks by decreasing processing time and scheduling the tasks in that order.

Example 1.1.5: LPT (Longest Processing Time) algorithm

Let us consider an instance with 2 identical machines and four tasks $\{t_1, t_2, t_3, t_4\}$ having processing times $p_1 = 3, p_2 = 4, p_3 = 8, p_4 = 5$. We sort the tasks by decreasing processing time, i.e. we will consider the tasks in the order t_3, t_4, t_2, t_1. The LPT algorithm then returns the following schedule:



This algorithm has several interesting properties. First as mentioned above, it is very fast to compute. Secondly it has a very good approximation ratio for the makespan minimization problem. It precisely have a ratio of $4/3 - 1/3m$, this means that we have a theoretical guarantee that the makespan of the schedule returned by LPT is at most $4/3 - 1/3m$ times the optimal makespan [Graham, 1969]. It performs even better in practice and the 4/3 ratio is only obtained for very specific instances. On a more general note, approximation is very commonly used in scheduling. Since a lot of problems are NP-hard, designing polynomial time algorithms which gives theoretical guarantees with regards to the optimization function seems like a good tradeoff between computation time and optimization.

If the reader is interested in the scheduling field, a more detailed presentation can be found in [Błażewicz et al., 2001; Brucker, 2010; Pinedo, 2012].

We now present the computational social choice field, another important research area in operations research as well as decision theory.

1.2 Computational social choice

Computational social choice [Brandt et al., 2016] focuses on collective decision making processes as voting, fair distribution of goods and more generally any process which consists in finding a common solution given the preferences of a set of agents. In the 1950s, research on collective decision processes was mostly conducted by economists and mathematicians who studied them from a normative angle, i.e. by studying their mathematical properties. This resulted in several very strong theoretical results, as we will see, but these works neglected the computational aspect of the processes. It is only from the 1980s that many computer scientists started investigating this field, bringing

a computational turn to the social choice theory field resulting in the computational social choice area. It heavily focuses on the decision process: how do we make sure that this process is fair and how do we make sure that the solution given by this process is satisfactory ? This section aims at introducing several common tools from computational social choice and especially the ones used in this thesis. We start by reviewing common methods for voting.

Notations. Voting deals with a set $V = \{v_1, \dots, v_v\}$ of v voters giving preferences over a set S of n candidates. The preference of voter v_i is denoted by R_i. The set of all these preferences is called the preference profile and is denoted by P. Depending on the context, these preferences can be given in different ways. The most commonly studied context is the one in which voters give full rankings over the candidates. We denote by X^S the set of all possible linear orders, or rankings, over the candidates in S. An aggregation rule r takes as an input a set of preferences and returns a collective solution. We can then describe r as $r : (X^S)^v \to X^S$.

Axioms. Axioms are desirable properties that an aggregation rule should follow. They are particularly studied in the theory of voting. Since one of the problems studied in this thesis, namely the collective schedules problem, is an extension of voting, we give detailed insights on axioms.

Let us start with an example. If we consider a public election, no voter should have an a priori greater impact than any other voter. This axiom is called anonymity (since voters are treated equally as if they were all interchangeable). Processes which favour a given voter do not fit well in election contexts. Therefore a way to validate an aggregation rule, is to check whether it fulfills axioms, like anonymity, or not. It would definitely be unacceptable for a population that the voting system used to elect the people's representatives does not satisfy anonymity, so any rule which does not fulfill anonymity cannot be used for public elections. Note that an axiom can be relevant in some contexts but not in others. For example, if stakeholders have to vote on a given issue relative to a company, then a stakeholder owning a very small part of the company should not have the same weight than the major stakeholder: in that case anonymity is not relevant. An important part of the work regarding axioms is to identify which ones are relevant in a given context. Given this information, we then try to find a rule fulfilling these axioms among the existing ones or we can design a new one fulfilling these properties. Axioms are very varied and can cover a lot of different aspects of the process from equal treatment, to resistance to manipulation. It is particularly interesting to look at combinations of axioms. Let us consider the three following axioms:

- Unanimity: if all voters prefer candidate a to candidate b, then in the ranking returned by the rule candidate a has to be ranked higher than candidate b.
- Independence of irrelevant alternatives: if every voter's preference on candidates a and b remains identical, then modifying the voters preferences on other pairs of candidates, e.g. a and c, b and d or c and d, will not modify the ranking of

a relatively to b in the ranking returned by the rule. In other words, whether a is ranked before b or b before a in the returned ranking only depends on the preferences of the voters over the pair a and b.

- Non-dictatorship: no single voter has the power to always decide the returned ranking.

These three axioms seem quite natural. However a very famous result, namely Arrow's impossibility theorem [Arrow, 1950], states that it is impossible for an aggregation rule to fulfill these three axioms.

Theorem 1.2.1: Arrow's impossibility theorem

No aggregation rule can fulfill both unanimity, independence of irrelevant alternatives and non-dictatorship.

This result shows that it is impossible to find an aggregation rule fulfilling this set of three natural axioms. Arrow's theorem states that we either have to chose a rule that could return a solution in which b is ranked above a, even if all voters prefer a to b, a rule for which the position of a relative to b depends not only on whether a is preferred to b by the voters, or a rule for which there is a dictator. None of these is ideal but we have to choose, rules often give up on the independence of irrelevant alternatives since non-dictatorship and unanimity seem more important. It also shows a very interesting aspect of the axiomatic study of rules. Since it is impossible to have everything at the same time, it is important to design rules that fit in different contexts. Having this toolbox (or axiom combination box) allows to pick and choose which rule seems to be the best in a given context. In that sense, when comparing two rules, if the second one does not fulfill an axiom fulfilled by the first, it may allow the second to fulfill another axiom which was not fulfilled by the first.

When studying the collective schedules problem, in Chapter 3, we will see that fulfilling some axioms may be incompatible with some other desirable properties.

Satisfaction as an optimization function. Independently from axioms, decision process can fulfill some other properties relative to an objective function we want to optimize. For example, if we manage to express how satisfied an agent is with a given solution, then a process finding the solution which maximizes the satisfaction of the agents is interesting in itself. Such a decision process can also fulfill some axioms (it is the case most of the time) and it is worth studying because it gives us some guarantee over the satisfaction of the agents. The Kemeny rule [Kemeny, 1959] is a classic voting system. It relies on a metric called the Kendall-Tau distance. The Kendall-Tau distance aims at measuring the difference between two rankings, supposedly one we are trying to evaluate and one which is given by a voter.

Definition 1.2.1: Kendall-Tau distance

Let R and R' be two rankings of the same set of n candidates $S = \{a, b, \dots n\}$. The Kendall-Tau distance between R and R' $\Delta_{KT}(R, R')$ is defined as follows:

$$\Delta_{KT}(R, R') = \sum_{(a,b) \in S^2} \mathbb{1}_{a \prec_R b, b \prec_{R'} a}$$

where $a \prec_R b$ means that a is ranked above b in ranking R.

For each pair of candidates (a, b), the Kendall-Tau distance counts 1 if a is ranked before b in R and b is ranked before a in R' or if b is ranked before a in R and a is ranked before b in R'. Intuitively, the Kendall-Tau distance counts the number of pairs on which rankings R and R' disagree.

Example 1.2.1: Kendall-Tau distance

Let us consider two rankings R and R' of 3 candidates a, b and c.

- Ranking R: $a \prec_R b \prec_R c$
- Ranking R': $b \prec_{R'} c \prec_{R'} a$

We can see that in ranking R the candidate a is ranked above candidate b and c. In ranking R', candidates b and c are ranked above candidate a. The Kendall-Tau distance then counts one disagreement on the pair (a, b) and one disagreement on the pair (a, c). Candidate b is ranked above candidate c in both rankings, the Kendall-Tau distance does not count any disagreement for this pair. The total Kendall-Tau distance $\Delta_{KT}(R, R')$ between R and R' is thus 2.

To evaluate a potential ranking R, it is possible to compute the Kendall-Tau distance of this ranking with every preference of the profile to obtain a measure of how far the ranking R is from the set of preferences given by the voters. The Kemeny rule precisely does that: it sums the Kendall-Tau distance of the ranking R to each preference of the voters to obtain an overall score for the ranking R. The Kendall-Tau distance between a ranking R and a preference profile P is then $\Delta_{KT}(R, P) = \sum_{v_i \in V} \Delta_{KT}(R^*, R_i)$. The higher this distance is, the further ranking R is from the set of preferences.

Definition 1.2.2: Kemeny rule [Kemeny, 1959]

The Kemeny rule returns a ranking R^* such that:

$$\Delta_{KT}(R^*, P) = \min_{R \in X^S} \Delta_{KT}(R, P)$$

Example 1.2.2: Kemeny rule

Let us consider 3 candidates $\{a, b, c\}$ and 9 voters with preferences, expressed as rankings, as follows:

- 3 voters have preferences $a \prec b \prec c$.
- 2 voters have preferences $c \prec a \prec b$.
- 2 voters have preferences $b \prec a \prec c$.
- 2 voters have preferences $c \prec b \prec a$.

Ranking $R = a \prec b \prec c$ has no disagreement with the first set of preferences. It disagrees on pairs (a, c) and (b, c) with the second set of preferences, so we count 2 times 2 disagreements. It disagrees on pair (a, b) with the third set of preferences, i.e. 2 times 1 disagreement, and on all pairs with the last set of preferences which amounts to 2 times 3 disagreements. The total Kendall-Tau distance with the profile is then $2 \cdot 2 + 2 \cdot 1 + 2 \cdot 3 = 12$, which is the lowest possible. The Kemeny rule returns ranking R.

The Kemeny rule fulfills several axioms, but independently of that, if we consider that the Kendall-Tau distance is an interesting way of measuring the distance between two rankings, then the solution returned by the Kemeny rule has an intrinsic value. It is the "closest" to the preferences of the voters according to the Kendall-Tau distance. This way of defining voting rules is common: we start by defining a notion of difference between a solution and a preference and we then aggregate these differences measure to obtain a global score for any possible solution, then returning a solution (in that case a ranking) minimizing this difference.

Another example we can mention is the Spearman rule [Diaconis and Graham, 1976], based on the Spearman correlation coefficient, which states that the distance between two rankings is the sum of the differences of the position of each candidate in both rankings.

Example 1.2.3: Spearman correlation coefficient

Let us consider the rankings R and R' from Example 1.2.1

- Ranking R: $a \prec_R b \prec_R c$
- Ranking R': $b \prec_{R'} c \prec_{R'} a$

The Spearman correlation coefficient ρ between rankings R and R' is computed

as follows:

$$\begin{aligned}\rho(R,R') &= \sum_{c\in S} |pos_c(R) - pos_c(R')| \\ &= |pos_a(R) - pos_a(R')| + |pos_b(R) - pos_b(R')| + |pos_c(R) - pos_c(R')| \\ &= |1-3| + |2-1| + |3-2| = 4\end{aligned}$$

Just like the Kemeny rule, the Spearman rule returns a ranking minimizing the overall Spearman correlation coefficient with the preference profile.

Among the rules proceeding in this way, some use a specific class of metrics to measure the difference between rankings. If this metric is a distance, in the mathematical sense, then the rule which consists in returning the ranking minimizing the distance with the preference profile necessarily fulfills certain axioms [Elkind et al., 2010, 2011]. This is the case of the Kemeny and Spearman rules. We will follow this principle when dealing with collective schedules and participatory budgeting.

On a final note, we mention here that both the Kemeny and the Spearman rules return a ranking minimizing the sum of the distances between the returned ranking and the preferences of the voter. However, instead of the sum, we could use other aggregators like the maximum, meaning that the returned ranking has to have the lowest maximum distance to a preference, i.e. the voter being the most unsatisfied with the solution has to be as satisfied as possible; or the product, meaning that instead of summing the distances to obtain a global distance with the set of preferences, we multiply them. These two other aggregators are known to be fairer than the sum, since they make sure that the least satisfied agents are not too unsatisfied. There are also other, more complex, aggregators, like OWA (Ordered Weighted Average) [Yager and Kacprzyk, 2012], that we will not use in this thesis but that could be interesting in the contexts studied in this thesis. We will use the sum in Chapters 3 and 4 but we will also use the product and the minimum in Chapter 6.

Probabilistic approach to voting. Another approach to voting consists in focusing less on the aggregation rule and more on the voters' behaviour. Instead of designing rules and studying their properties, it is possible to try and describe the way voters behave by using a probabilistic model [Elkind and Slinko, 2016; Xia, 2019]. The idea is the following one:

1. we suppose that there is an objective ground truth, which can consist in a ranking or anything that allows us to rank the candidates;

2. we suppose that voters do not have exact knowledge of this ground truth, they have only an imperfect perception of it, the votes are then interpreted as noisy observations of the ground truth;

3. by making assumptions on what this noise is, we can estimate how likely each ground truth is given the votes;

4. we return the best possible estimation of the ground truth, according to the votes and the assumptions on the voters behaviour.

Such an approach is called probabilistic because the assumptions on the noise give us a probabilistic model. The idea is to associate a probability with every possible vote, i.e. if we consider that the ground truth is a given ranking R, then a voter has a probability $p(R'|R)$ of observing another given ranking R'. This means that given a ground truth, we can estimate the probability of observing the preferences expressed by the voter and finally, we can estimate which ground truth is the most likely. The main objective of this approach is to give a tool that evaluates how well we can explain the behaviour of voters: the better a probabilistic model fits to the preferences we observe, the better it is. One final aspect is that it is possible, for certain aggregation rules, to find a corresponding probabilistic model: the rule returns a ranking which is a best estimation of the ground truth [Young, 1988; Conitzer et al., 2009]. In such cases, we can both have axiomatic guarantees about the rule and an evaluation of how well the model fits to the preferences of the voters. We will use this approach in Chapter 5.

Complexity. Another way to evaluate a decision process is to look at its complexity. The complexity of a process is given as a function of the size of the input. For example in elections, we have a number of voters v and a number of candidates n. The complexity of a process returning a solution to the election would then be a function of v and n. Since real life elections can gather the preferences of millions of voters over a large set of candidates, it is important for a voting system to be able to handle large size instances. Finding a ranking minimizing the Kendall-Tau distance is known to be NP-hard. However several resolution techniques, like dynamic programming [Betzler et al., 2009], have been used to find an optimal Kemeny ranking. These techniques do have an exponential complexity, but this complexity increases exponentially only with the number of candidates, so it may be possible to run the Kemeny rule if the number of candidates is not too large. On the other hand, some rules, like the Spearman rule, can be solved in polynomial time. Indeed, finding an optimal ranking for the Spearman rule can be reduced to an assignment problem, which can be solved in $O(n^3)$[Edmonds and Karp, 1972]. Once more, knowing the context in which the rule is going to be applied is key. If the number of candidate and voters is small, it may be better to go for a rule with high complexity but high theoretical guarantees, but in contexts in which the instance to solve has a very large size, it is probably better to have a rule with low complexity.

Observation 1.2.1: An issue that is hard to evaluate

Beyond all these theoretical aspects, there is also a practical one which may be even more important but which researchers may struggle with. No matter how good a rule is, how well it fits in a given context it is also essential that voters understand how the rule works. A voting rule has to be trusted in order for voters to be willing to participate, and it is easier for a rule to be trusted if it is

understood. We do have a lot of tools to measure complexity, in terms of number of operations, but very few to measure the "simplicity" of a rule for the voters. We can mention the experimental studies which directly ask voters about which systems seems to be the best according to them [Rosenfeld and Talmon, 2021]. But beyond these empirical studies, not much has been done to evaluate this criterion. In this thesis we will study several aggregation rules, some of them very "simple" and intuitive and some others way harder to explain to voters or agents. Most of the time the simpler rules lack some theoretical guarantees that the more complex rules give. However, in practice, using simpler rules may be the best option if we want the citizens to participate in the voting process.

Multi-winner voting Multi-winner voting consists in selecting a set of k winning candidates among the n candidates of S given the preferences of the voters [Faliszewski et al., 2017]. In classic contexts, a solution is either a unique candidate, who wins an election for instance, or a complete ranking of the candidates. In this case, we want a subset of candidates. When electing members of a parliament, it makes sense to look for a subset of candidates that is representative of the population. This idea of representation is formulated by the proportionality property. On the other hand, when choosing a committee of experts, it may be better to aim for the k "best" candidates, regardless of representation issues. Just like the classic voting problem presented earlier, there is a wide range of axioms corresponding to different contexts and that rules fulfill or not. In recent years, an extension of multiwinner voting has been widely studied: the Participatory Budgeting problem Aziz and Shah [2021]. In this problem, the objective is to select a set of projects that fits in a given budget. Voters express their preferences over the projects and the aim is then to find a consensus set of projects that does not exceed the budget. When all the projects have the same cost, we fall back to the multiwinner voting problem, since the number of projects that can be chosen is known. We will study this problem in Chapter 6.

Other resolution concepts. We conclude this section by mentioning other usual concepts in social choice problems that are not necessarily based on preference aggregation. One of these problems is the allocation of goods [Bouveret et al., 2016]. A set of m goods, e.g. car, house, money, candies, ..., has to be given to a set of n agents. Each agent gets a certain satisfaction from each good, depending on what she values. A solution is then an allocation of each good to an agent, each agent may receive several goods. The aim is often to find an allocation that is both fair and efficient. The efficiency can be measured by looking at the sum of the satisfaction of the agents or by looking at an optimal solution, where optimality is defined as the fact that no agent can improve its solution without another one decreasing her satisfaction. Fairness on the other hand is measured using different properties. The most commonly used is envy-freeness, which states that no agent should be more satisfied if she had the goods given to another agent instead of the good she has. Another notion of interest is proportion-

ality which states that each agents should be at least as satisfied with the goods she gets than the satisfaction she would get by having all the goods divided by the number of agents. A proportional allocation does not always exist, neither does an envy-free allocation. Some papers also focus on the trade off between efficiency and fairness [Aziz et al., 2023] in fair allocation.

We complete this chapter with a short review of different scheduling problems when several agents are involved.

1.3 Multi-agent scheduling

Several multi-agent problems have been studied in the literature. There are many different setups depending on the number and characteristics of machines, agents and tasks. The aim of this section is not to present an extensive review of multi agent scheduling problems studied in the literature but to introduce resolution concepts for such problems.

Competitive agents scheduling on a common machine. In this setting several agents each own a subset of tasks. The goal for each agent is to schedule her tasks on a common machine and to optimize an objective function on her set of tasks. This objective function can for example be the minimization of the sum of the completion times of her tasks [Perez-Gonzalez and Framinan, 2014]. In their book, Agnetis et al. [2014] give a detailed analysis of a large class of problems in this setting, as well as in other settings, as we will see later. They describe several solution concepts:

- *Pareto optimal solutions*: Since there are several objective functions – one for each agent, the notion of optimal solution is not as straightforward as it is when dealing with one objective only. In multi-objective optimization problems, a solution S is said to be Pareto optimal if there exists no other solution which is at least as good for all objective functions and strictly better for at least one objective function. To solve the problem we can either look for a Pareto-optimal solution or the set of all Pareto-optimal solutions.

- *Linear combination of criteria*: Among the Pareto optimal solutions, some maximize certain functions aggregating the different objectives. For example let us consider a two objectives setting: the first objective consists in minimizing some function f_1 and the second objective consists in minimizing some other function f_2. Then, it is possible to consider a function $f = \alpha_1 f_1 + \alpha_2 f_2$. The solution minimizing function f is Pareto optimal. The coefficient α_1 and α_2 allow to give more or less weight to each objective function.

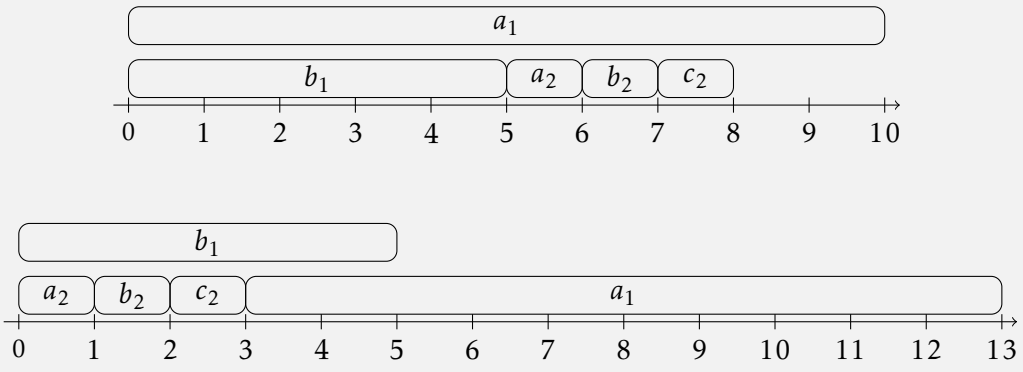
- *Epsilon-constraint*: This approach consists in setting a minimum quality expected for all objectives but the objective f_k and to optimize according to f_k. The solution found is then the best solution for f_k among the solutions that are good enough for the remaining objectives. Among all the Pareto optimal solutions, some may be

very unbalanced. For example a solution which is only optimizing one objective function f_k is Pareto optimal, since it is impossible to improve any other objective without deteriorating the value of f_k. However it is possible, and often the case, that such a solution is very unsatisfactory for the other objective functions. In such case, it is interesting to have some guarantee over the minimum quality of a solution on each objective function.

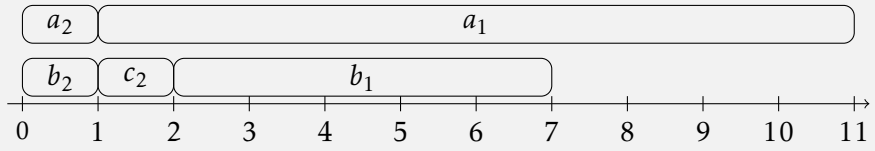
In a recent paper, Agnetis et al. [2019] use another solution concept: the Kalai-Smorodinski fairness, when two agents are competing on a single machine. In a minimization problem, the utility that agent i gets from a solution S is defined as the difference between the value of the agent's objective in the worst possible solution for the agent and the value of her objective in S. If f_i is the objective function for the agent i and S_i^∞ is the worst possible solution for the agent, then $u_i = f_i(S_i^\infty) - f_i(S)$. We denote by S_i^* the best possible solution for agent i. The utility u_i is then normalized by dividing it by $f_i(S_i^\infty) - f_i(S_i^*)$. A schedule is said to be Kalai-Smorodinski fair if it maximizes the minimum normalized utility among the agents. Using these normalized utilities allows all the satisfactions to be measured on a similar scale.

Observation 1.3.1: Kalai-Smorodinski fairness

This definition of fairness is extremely relevant in multi agent scheduling settings. Indeed, the different agents can have very different sets of tasks depending on the number and the processing time of the tasks. It is also interesting to look at the best and worst solutions for the agents since, depending on the instance structure, they can be very different or pretty close.
Let us consider a simple example. Two agents each have a subset of tasks and there are two common machines. Each agent wants to minimize her makespan, i.e. the completion time of her last task. Agent 1 owns two tasks, one of processing time 10, called a_1, and one of processing time 5, called b_1. Agent 2 owns three tasks of processing time 1, called a_2, b_2 and c_2. Among the solutions with no idle time the best for agent 1 gives her a makespan of 10 and the worst makespan of 13.



The best reachable makespan for agent 2 is 2 and the worst is 8 (as seen in the first schedule).



In this example, we can see that although a makespan of 10 is very satisfying for agent 1, it is unacceptable for agent 2, so comparing the objective function values is not relevant. Additionally the structure of the instance may limit the quality of solutions for one agent. In the example, agent 1 will always have a makespan of 10 at least because she has a task of processing time 10. Kalai-Smorodinski fairness allows to put the evaluation for all agents on a same scale and to make sure that this scale takes the instance structure into account. In Chapter 2, we will study a definition of agent satisfaction that is close, although not identical, to this definition of satisfaction.
There is however one drawback to this definition: it requires to compute the best and worst possible values of the objective function for each agent and this may be an NP-hard problem. It is the case in Chapter 2, where agents aim at minimizing their makespan and this problem is NP-hard.

We also mention that there exist other settings, in which some tasks may be of interest to several agents at the same time, in which there may be several parallel machines or in which the processing time of tasks may vary [Agnetis et al., 2014].

Among these extensions, Saule and Trystram [2009] study a case in which there are several agents competing to schedule their jobs on a set of common machines. Each agent has her own objective function. They propose several algorithms to solve this problem, some using combinatorial optimization techniques as well as some greedy algorithms, including one which is a constant approximation of a Pareto-optimal solution.

Multiple agents, each having one task Several papers look at fairness among tasks. Each agent is supposed to have one task and the goal is to design fair processes. In their paper, Niu et al. [2022] study a simple problem, the minimization of the sum of the completion times on a single machine. An optimal solution for this problem is obtained by scheduling the tasks by increasing processing times. However, such an algorithm is unfair in the sense that agents are not treated equally because of the processing time of the tasks. A fair process can be obtained by randomization, in this case each agent has an equal chance for her task to be processed first. The authors study the trade off between efficiency, in terms of the sum of completion times, and fairness in terms of expected completion time of each task.

Multiple agents with multiple tasks Using a game theory approach, Cohen and Pascual [2015] study a problem in which agents own several tasks and can choose to which machine they want to affect which task. Each machine has a scheduling policy, namely a way to order the tasks scheduled on the machine. The goal for each agent is to minimize the average completion time of her task. And the goal of the system is to minimize the average completion time of all tasks. Each agent can have different strategies, namely ways of spreading her tasks on the different machines. The authors look at equilibrium in such a setting.

We mention the Multi Organization Scheduling Problem [Pascual et al., 2007] in which several organizations each own tasks and machines. They have local schedules, i.e. schedules of their tasks on their machines. We look at what happens when the organizations share their machines. The goal is then to find a schedule of all the tasks on all the machines and in which each organization has a solution at least as good as its local schedule. We will give a detailed presentation of the Multi-Organization Scheduling Problem in Chapter 2.

We also quickly introduce the Collective Schedules problem [Pascual et al., 2018] in which a set of tasks common to a (potentially large) set of agents has to be scheduled. Each agent has her own preferences regarding the order of the tasks and the goal is to find a consensus schedule which satisfies at most the agents. This problem is an extension of the voting problem introduced earlier. We will focus on Collective Schedules in Chapters 3 and 4.

Chapter 2

Efficiency and Equity in the Multi-Organization Scheduling Problem

The Multi-Organization Scheduling Problem (MOSP) is a scheduling problem in which several organizations (or agents) have tasks and/or machines. Each organization has a "local" schedule in which it schedules its own tasks on its own machines. We consider that the organizations collaborate by sharing their machines in order to improve the quality of their solution. The goal is to find a schedule of all the tasks on all the machines (a task can be scheduled on a machine owned by another organization) which satisfies all the organizations. Our objective here is to study the tradeoff between efficiency, in terms of global performance, and fairness, by making sure each agent benefits from sharing the machines. Regarding fairness, we will at first consider a rationality constraint which requires that each organization has a solution at least as satisfying as its local schedule when sharing the machines. In other words, an organization cannot loose anything by sharing. This constraint ensures that organizations have an incentive to collaborate, however fulfilling it can impact the efficiency of the solution and our goal is to understand to which extent. In a final part, we will consider fairness as a main objective and formulate a new problem, by trying to find solutions not only fulfilling the rationality constraint but in which each organization gains as much as possible.

This work has been published in [Durand and Pascual, 2021].

2.1 Introduction

Cost constraints, as well as environmental issues, make the sharing of machines between independent organizations (such as laboratories or universities) a very interesting solution. Sharing machines allow organizations which need to execute tasks to use the machines of organizations which do not need machines at this time, decreasing

the completion time of the tasks without having to invest in new machines. But cooperation is even more than sharing unused machines with organizations who need to schedule tasks: cooperation can benefit simultaneously several organizations which all have tasks to compute, by allowing a better placement of the tasks, as we will see in the sequel. The Multi Organization Scheduling Problem (MOSP) [Pascual et al., 2007] deals with several organizations which each owns both a set of identical parallel machines and a set of sequential tasks to execute. The objective is to minimize the maximum completion time of the last task completed on the machines shared by the organizations, called *global makespan*, given that no organization should increase the completion time of its tasks in the shared system, compared to the case where it executes its own tasks on its own machines. This last constraint is called the *rationality constraint*, and ensures that all the organizations have incentive to share their machines.

Besides analyzing the best possible benefit that organizations can mutually have by sharing their machines, our aim is to focus on the *efficiency* of algorithms (where the efficiency is thought in term of makespan – the date at which all the tasks have been computed), and on the *equity* of algorithms for MOSP (even if the rationality constraint is fulfilled, the benefit should be spread among all organizations). These two aspects may be antagonist, and our aim is to see to which extent, since what we want would be a schedule with a small makespan and in which machines are shared with equity. We will start by reviewing existing work on MOSP, and continue by presenting our results and the structure of the chapter.

2.1.1 Related work.

The Multi Organization Scheduling Problem [Pascual et al., 2007, 2009] has been introduced with parallel rigid tasks (tasks that need to be executed in parallel on several machines) and has mainly been studied from an approximation viewpoint. The best approximate algorithm is a 3-approximation algorithm when the organizations schedule locally the tasks in decreasing order of their heights (the height of a task is the number of machines needed to execute the task), or a 4-approximation algorithm in the general case [Dutot et al., 2011]. For sequential tasks (tasks that need to be executed on one machine only), the best known algorithm is a 2-approximate algorithm [Cohen et al., 2010] (in the sequel of this section, all the papers – as well as our results – deal with sequential tasks). Note that all these bounds are not only approximation ratios, since they are in fact upper bounds of the ratio, in the worst instance, $\frac{C_{\max}(\mathcal{S})}{OPT^{\mathcal{T}}}$, where $C_{\max}(\mathcal{S})$ is the makespan in a solution returned by the algorithm and $OPT^{\mathcal{T}}$ is the smallest possible makespan that can be obtained by scheduling the same set of tasks on the same set of machines (this last schedule does not necessarily fulfill the rationality constraint). Lower bounds on such a ratio have also been given: it has been proved that there is no algorithm with a ratio smaller than 2 when the tasks are parallel [Pascual et al., 2009], or when the tasks are sequential and when it is required that, in the returned schedule, the machines of each organization schedule their own tasks before scheduling the tasks of other organizations [Cohen et al., 2010, 2011b]. A lower bound of $\frac{3}{2}$ is known in

the general case for sequential tasks Pascual et al. [2007]; Cohen et al. [2010] – we will improve this bound in the sequel.

Several variants have been studied: for example, organizations may choose themselves on which machines to schedule their tasks knowing that each machine schedule the tasks of its organization first Cohen et al. [2011b]. Despite most works deal with minimizing the overall makespan while each organization wishes to minimizes its own makespan, other objectives have also been studied: the aim can be to minimize the average completion time of tasks [Cohen et al., 2010, 2011b], or the energy needed to schedule the tasks [Cohen et al., 2014]. These papers usually show that the problem is NP-hard and then give approximation algorithms or heuristics.

Some papers also consider a relaxed version of MOSP: it is assumed that the organizations tolerate a bounded degradation on the makespan of their own tasks, and the aim is to minimize the global makespan. This problem is denoted by $(1+\alpha)$-MOSP [Ooshita et al., 2009] when it is assumed that each organization accepts to increase the maximum completion time of its tasks by a factor at most $(1+\alpha)$. A $\frac{3}{2}$-approximate algorithm for 2-MOSP has been given [Cordeiro et al., 2011]. Other work include additional constraints on the machines [Chakravorty et al., 2013]. The closest work in spirit to what we will do in Section 2.4 is a study of $(1+\alpha)$-MOSP on unrelated machines [Ooshita et al., 2009, 2012]. In this setting, Ooshita et al. show that, when there is no cooperation ($\alpha = 0$), the makespan can be m times higher than in the optimal makespan without the rationality constraint. When $\alpha > 0$, the authors also give a $(2+\frac{2}{\alpha})$-approximate algorithm for $(1+\alpha)$-MOSP.

There is few work on fairness issues when some organizations own tasks and machines. In an experimental work, Cohen et al. [2011b] look at the fairness (using stretch and Jain Index) of schedules returned by some algorithms, and they show that the best results are obtained by algorithm ILBA [Pascual et al., 2009]. In another work, Skowron and Rzadca [2013] model the fair scheduling problem as a cooperative game and use the Shapley value to determine an ideal fair schedule. To calculate the contribution of an organization, they determine how the presence of this organization influences the performance of other organizations. For unit-size tasks they give a fully polynomial-time randomized approximation scheme, and they show this problem is NP-hard and hard to approximate in the general case.

Other works about fairness in scheduling are mainly about how to schedule tasks of different users on a set of shared machines. In this context [Agnetis et al., 2014], several agents own an individual set of tasks and the objective is to schedule the tasks of all agents on a set of common machines. Each agent has her own objective function, the goal is then to find a solution in which each agent is satisfied given that the objectives of the different agents can be antagonist. The aim is then to find a Pareto optimal solution, i.e. a solution in which improving the satisfaction of an agent necessarily deteriorates the satisfaction of another agent, and if possible to find a fair one.

2.1.2 Overview of our results

In this chapter, we consider that N organizations $O_1, \ldots, O_N$ share m machines, and that each organization O_i has its own set of tasks $\mathcal{T}_i$. Each organization O_i wishes to minimize its makespan, i.e. the date at which all its tasks (the tasks of $\mathcal{T}_i$) have been completed. If each organization O_i schedules its own tasks (and only its owns tasks) on its own machines, these tasks are completed at a date which will be called the *local makespan* of O_i. We consider that this schedule is given by the organizations, they can either use a heuristic or solve the problem optimally, even though $(P||C_{\max})$ is a NP-hard problem. We have two objectives, which can be antagonists. First, we would like to return a schedule which is as efficient as possible, and thus which minimizes the global makespan while not increasing the local makespans . Second, we would like to return a fair schedule. Our results are as follows.

In Section 2.3 we show that cooperation can permit to decrease the makespan of each organization by a factor N (but no more). This shows that cooperation can benefit to all the organizations simultaneously, and not only to some organizations which own many tasks or few machines. In this section, we also give a polynomial time algorithm with resource augmentation: for a fixed $\epsilon > 0$, and a fixed number of organizations, it returns a solution $(1+\epsilon)$-approximate in which each organization has a makespan at most $(1+\epsilon)$ times its local makespan.

In Section 2.4, we relax the rationality constraint by considering $(1+\alpha)$-MOSP: we assume that each organization agrees to complete its last task at a date at most $(1+\alpha)$ times its local makespan. We are interested by the trade off between the value of α and the value of the (global) makespan. We first show that an algorithm which returns schedules which minimize the makespan can have to increase a local makespan by a factor $m-1$, which is certainly unacceptable for the agents. We then focus on the ratio than can be obtained for the global makespan, for a fixed α: we give a lower bound of the necessary increase of the makespan in $(1+\alpha)$-MOSP with respect to the optimal makespan without the rationality constraint. If $\alpha = 0$, $(1+\alpha)$-MOSP corresponds to MOSP (no organization should get a makespan higher than its local makespan). In this case, the obtained lower bound shows that it is not possible to obtain an algorithm which outputs 2-approximate schedules for the makespan and which fulfills the rationality constraint. This improves the lower bound of $\frac{3}{2}$ given in Pascual et al. [2007]; Cohen et al. [2010].

In Section 2.5, we define the gain of an organization as the ratio between its local makespan minus its makespan in the schedule returned over its local makespan. Since we want to fulfill the rationality constraint, this gain will be at least 0, but the higher this gain is, the higher an organization will be satisfied by the schedule returned. We are interested by getting fair schedules: we introduce the problem which consists in returning schedules which maximize the minimal gain of an organization. For the unit tasks case, i.e. the case in which all tasks have the same processing time, we give a polynomial time optimal algorithm for this problem. For the general case, we show that this problem is NP-complete, and even hard to approximate, and we give an heuristic which outputs, in practice, schedules close to the optimal ones.

We conclude this work by giving a few research direction in Section 2.6. Before starting to present our technical results, we start, in Section 2.2, by introducing notations and defining formally our problem.

2.2 Preliminaries

2.2.1 Notations

By $\mathcal{O} = \{O_1, ..., O_N\}$ we denote the set of N independent organizations sharing m identical machines $\{1, \ldots, m\}$ and n tasks. Each organization O_i, with $i \in \{1, \ldots, N\}$ owns $m_i \geq 1$ machines, and a set $\mathcal{T}_i$ of $n_i \geq 0$ tasks. If $n_i > 0$, these tasks are denoted by $t_i^1, \ldots, t_i^{n_i}$. Tasks are sequential: each task t_i^j is executed on a single machine, during a processing time (also called length) $p_i^j > 0$. We denote by $m = \Sigma_{i=1}^N m_i$ the total number of machines, and by $n = \Sigma_{i=1}^N n_i$ the total number of tasks. We denote by $\mathcal{T} = \cup_{i=1}^N \mathcal{T}_i$ the set of all the tasks.

Given a task j, and a considered schedule S, we denote by $C_j(S)$ the *completion time* of task j in schedule S, i.e. the date at which its execution ends. Preemption is not allowed: once a task starts to be executed, it will be executed until its completion.

MOSP takes as input the local schedules of the organizations. The local schedule of Organization O_i is a schedule of the n_i tasks of O_i on the m_i machines of O_i. This schedule may minimize the makespan of O_i, or not (this problem is indeed NP-hard [Garey and Johnson, 1979]): Organization O_i computes itself its local schedule and gives it to a central entity. We will denote by S_{loc}^i the *local schedule* of Organization O_i, and we will denote by C_{loc}^i the makespan of this schedule (this will be called the *local makespan* of O_i).

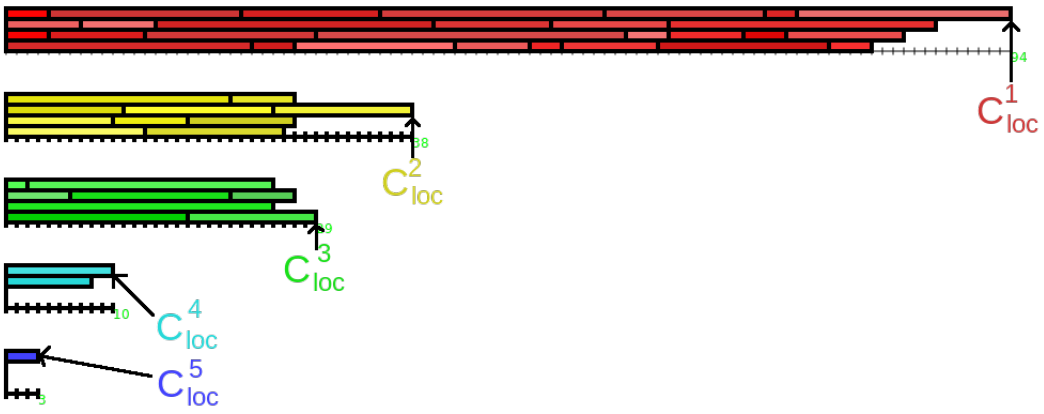


Figure 2.1: Example of a set of local schedules and local makespans for $N = 5$. Time on the x axis, organizations on the y axis.

Given a schedule S of the n tasks on the m machines, we will denote by $C_{\max}^i(S)$

the completion time of the last task of Organization O_i in S, also called the *makespan of Organization* O_i in S. Given a schedule S we will denote by $C_{\max}(S)$ the completion time of the last task in S. Therefore $C_{\max}(S) = \max_{i\in\{1,\dots,N\}} C^i_{\max}(S)$, and is called the *global makespan* (also called the makespan of S).

Given $i \in \{1,\dots,m\}$, we will denote by $L_i(S)$ the *load of machine* i in schedule S: this is the sum of the processing times of the tasks assigned to machine i in S. The *total load* is the sum of the processing times of all the tasks of the schedule ($\Sigma_{i=1}^{N}\Sigma_{j=1}^{n_i} p_i^j$).

2.2.2 Problem statement

The objective of each organization O_i is to minimize $C^i_{\max}(S)$, the date at which all its tasks are completed in the returned schedule S. The multi-organization scheduling problem (MOSP) consists in scheduling the n tasks of all the organizations, on the m machines of the organizations, in order to minimize the global makespan with the additional constraint that no organization has a makespan larger than the makespan of its local schedule:

$$\text{minimize } C_{\max}(S) \text{ such that, for each } i \in \{1,\dots,N\}, C^i_{\max}(S) \leq C^i_{loc}.$$

The set of these additional constraints is called the *rationality constraint*: it ensures that each organization will have incentive to accept the schedule returned by the central entity (or trusted third party), since it will not be able to get a better makespan if it schedules its own tasks on its own machines.

Given an instance I, we will denote by S^* an optimal solution for MOSP, and we will denote by OPT the makespan of such a solution. In the sequel, we will be interested in comparing OPT, or the makespan returned by an algorithm, to the best solution without the rationality constraint, that we will denote $S^{*(\bar{r})}$. In such a solution, all the tasks are scheduled on all the machines in order to minimize the global makespan: this is an optimal solution of the classical scheduling problem ($P\|C_{\max}$). We will also denote by $OPT^{(\bar{r})}$ the makespan of $S^{*(\bar{r})}$.

2.3 Interest of cooperation and algorithm

In this section, we measure to what extent cooperation can reduce the makespans of organizations, with respect to a schedule made of the local schedules only. We will show in Section 2.3.1 that on some instances, cooperation can decrease simultaneously all the makespans. In Section 2.3.2, we present an algorithm which returns a schedule whose makespan is at most $(1+\epsilon)$ times the makespan of an optimal schedule of MOSP, while the makespan of each organization in this schedule is at most $(1+\epsilon)$ times its local makespan.

2.3.1 Cooperation can decrease all the makespans

If the local makespan of Organization O_i is much larger than the local makespan of the other organizations, then, by load balancing tasks of O_i on the machines of all the organizations, the makespan of O_i may decrease a lot. In this section, we show that MOSP is more than load balancing tasks of heavy loaded organizations on the machines of less loaded organizations: there are instances for which all the organizations can simultaneously benefit of cooperation. Figure 2.2 shows an instance with $N = 3$ organizations in which all organizations can benefit a lot from sharing their machines. Proposition 2.3.1 shows that all the organizations may together reduce their makespans up to a factor N by cooperating with each other.

Proposition 2.3.1: Best case scenario

In an optimal schedule for MOSP, all the organizations may decrease simultaneously their makespans up to a factor N, with respect to their local makespans (which are assumed to be optimal). This is the best possible bound : there is no instance where each organization can decrease its makespan by a factor larger than N.

Proof. Let us first exhibit an instance where each organization improves its makespan by a factor as close as wished of N. We consider an instance where each of the N organizations owns a single machine (therefore $m = N$). For each $i \in \{1,\dots,N\}$, Organization O_i owns mx^{i-1} tasks of length 1 (thus Organization O_1 owns m tasks, while Organization O_N owns mx^{N-1} tasks). The local makespan of Organization O_i is therefore mx^{i-1}.

Let us now consider the following schedule, S, optimal for MOSP : on each machine, there are one task of Organization O_1, followed by x tasks of Organization O_2, followed by x^2 tasks of Organization O_3, and so forth. The schedule ends on each machine with x^{N-1} tasks of O_N. For each $i \in \{1,\dots,N\}$, the makespan of O_i in S is $1 + \Sigma_{j=2}^{i} x^{j-1}$. Therefore, each organization O_i decreases its makespan, from S_{loc}^i to S, by a factor $\frac{C_{loc}^i}{C_{\max}^i(S))} = \frac{mx^{i-1}}{1+\Sigma_{j=2}^{i} x^{j-1}}$. This tends towards $m = N$ when x tends towards to the infinity. An example of such an instance when $N = 3$ is shown in Figure 2.2.

Let us now show that there is no instance where cooperation can make each organization decrease its makespan by a factor larger than N. By contradiction, let us assume that there exists an instance I for which there is a schedule S in which the makespan of each organization is decreased by a factor larger than N with respect to its local makespan (assumed to be optimal). Note that there is in I at least one organization O_i such that $m_i \geq \frac{m}{N}$ (otherwise, we would have $\Sigma_{i=1}^{N} m_i < m$). By hypothesis, the makespan of O_i in S is $C_{\max}^i(S) < \frac{C_{loc}^i}{N}$. We now show that this implies that it is possible for O_i to obtain a schedule of its tasks on its machines with makespan smaller than C_{loc}^i.

Indeed, let us consider the following schedule of the tasks of O_i on m_i machines : compute between time 0 and $C_{\max}^i(S)$ the tasks scheduled in S on the first m_i machines, and then the tasks scheduled in S on machines $m_i + 1, 2m_i$ between time $C_{\max}^i(S)$ and

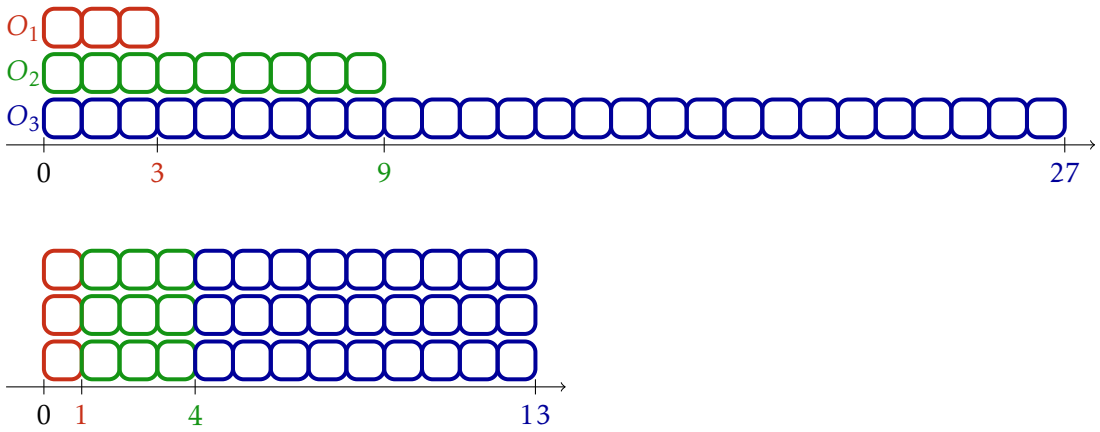


Figure 2.2: Example of best case instance when $N = 3$ before (top) and after (bottom) sharing machines

$2C^i_{\max}(S)$, etc. (tasks scheduled in S on machines $(x-1)m_i+1, xm_i$ are scheduled between time $(x-1)C^i_{\max}(S)$ and $xC^i_{\max}(S)$, as they are scheduled in S : a task starting at time t on machine j will be scheduled at time $t \mod C^i_{\max}(S)$, on machine $(j \mod m_i)$ if $j \mod m_i \neq 0$ and on machine m_i otherwise). This schedule is a feasible schedule of makespan at most $NC^i_{\max}(S) < C^i_{loc}$. Therefore, the local schedule of O_i was not optimal, a contradiction.

An example with $N = 3$ organizations can be found in Figure 2.3. The figure shows how to build, from a given schedule S, a local schedule for an organization O_i with $m_i \geq m/N$ such that the makespan of this local schedule is at most $NC^i_{\max}(S)$.

□

2.3.2 A PTAS with resource augmentation

In this section, we show that the polynomial approximation scheme (PTAS) presented by Hall and Shmoys [1989] for a scheduling problem can be used to get a PTAS with resource augmentation for our problem. More precisely: given a fixed $\epsilon > 0$, and a fixed number of organizations N, we will get a polynomial time algorithm which returns a schedule with a makespan at most $(1+\epsilon)OPT$, and in which the makespan of each organization is at most $(1+\epsilon)$ times its local makespan. The rationality constraint may thus be violated, but the increase of the makespans of the organizations is bounded, and may be acceptable if ϵ is small. Let us start by presenting the scheduling problem studied by Hall and Shmoys.

Scheduling problem with delivery times (SchedDT). The input of this problem consists in n_{DT} tasks $\{1,\ldots,n_{DT}\}$ and m_{DT} identical machines. Each task j has a processing time p_j (it must be processed without interruption for time p_j on any one of the m_{DT} machines), a release date r_j (the date at which it becomes available for processing), and a delivery time q_j. Each task's delivery begins immediately after its processing

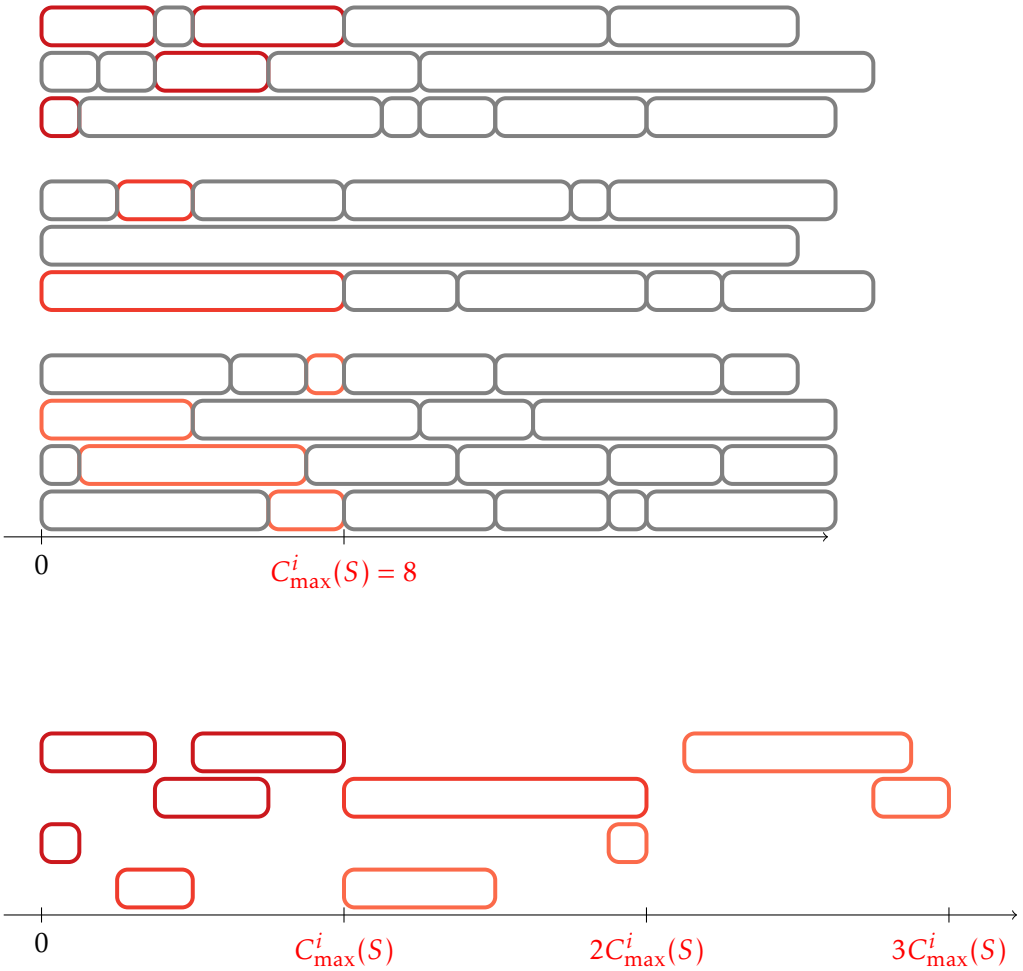


Figure 2.3: Example showing that it is impossible to decrease its makespan by a factor higher than N. In this case $N = 3$, O_i has $m_i = 4$ machines and owns the colored tasks. The other tasks are owned by the other organizations. A schedule S of all the tasks on all machines (top) and a potential local schedule of makespan at most $NC^i_{\max}(S)$.

has been completed, and all tasks may be delivered simultaneously. Therefore, for a given schedule S in which task j starts at time σ_j, the completion time of task j is defined as $C_j(S) = \sigma_j + p_j + q_j$. The aim is to minimize, over all possible schedules, the makespan $C_{\max}(S) = \max_{j\in\{1,\dots,n\}} C_j(S)$. In the sequel, we will denote this problem as SchedDT. As noted by Hall and Shmoys, this problem is equivalent to the scheduling problem with release dates (r_j) and due dates (d_j) – and without delivery times – in which the objective is to minimize the maximum lateness, where the lateness of task j is $L_j = \sigma_j + p_j - d_j$. This last problem is denoted as $(P|r_j|L_{max})$, using Graham's notation for scheduling problems. However, while this problem is inapproximable in polynomial time if $P \neq NP$, there exists a PTAS for SchedDT. Let us now give a high level description of this PTAS, that we will use for our problem is the sequel. The details can be found in the original paper [Hall and Shmoys, 1989].

High level description of the PTAS for SchedDT**.** This PTAS is a generalization of the PTAS of Hochbaum and Shmoys [1987] for problem $(P||C_{\max})$. The principle of Hall and Shmoys's algorithm is the following one. It assumes that there are a lower bound LB and an upper bound UB of the optimal makespan OPT_{DT} of SchedDT, such that $LB \leq OPT_{DT} \leq UB \leq 2OPT_{DT}$. It then does a dichotomic search with a target value T on this interval: for each target value, the algorithm either builds a schedule of makespan at most $T(1+\epsilon)$, or it assures that there is no schedule of makespan at most T. At the end of the dichotomic search, the schedule found with the smallest value of T which lead to a feasible schedule is returned.
Before this, a preprocessing step consists in rounding the input: the releases dates are rounded down to obtain a fixed number of distinct ones. The same thing is done for delivery times. Tasks are partitioned into two sets: large tasks (tasks whose processing times are larger than or equal to a given number δ function of ϵ), and small tasks (smaller than δ). Large tasks are rounded down so that there is a fixed number of different processing time for the large tasks. Given that, for large tasks, there are a fixed number of different values of q, r and p, there is now a fixed number τ_1 of different types of large tasks. Small tasks are "glued" into small components of size δ and of common values r and q (once these values have been rounded): there is now a fixed number τ_2 of different types of small components (which gather small tasks). Let X be the set of possible types of tasks ($|X| = \tau_1 + \tau_2$). A machine configuration indicates, for each type of task $t \in X$ how many tasks of type t are on the machine. Given the size of the large tasks, we can upper bound the maximum number of large tasks per machine in a schedule with a makespan smaller than $2OPT_{DT}$ and show that the number of relevant machine configurations is fixed (let us denote by γ this number). For a given schedule, x_l indicates the number of machines with configuration l: vector $x = (x_1, \dots, x_\gamma)$ defines an outline for the schedule. Therefore, the number of relevant outlines is at most m^γ, a polynomial in m. The order of tasks on machine is based on a generalization of the Jackson's rule [Jackson, 1955], a polynomial time optimal algorithm for $(1||L_{max})$ (where the aim is to minimize the maximum lateness on a single machine), when there are release date. This algorithm schedules the tasks by increasing due date. This problem, $(1|r_j|L_{max})$, is solved in polynomial time [Hall and Shmoys, 1989]. The algorithm tries

every relevant outline. If at least a schedule with makespan at most T is found, the algorithm outputs the best schedule – a schedule, with rounded tasks of makespan at most T. When the tasks take back their true values, this becomes a schedule of makespan at most $(1+\epsilon)T$. By doing a dichotomic search over T, this algorithm returns a $(1+\epsilon)$-approximate solution for the SchedDT [Hall and Shmoys, 1989]. Let us now see how we can use it to get a PTAS with resource augmentation for MOSP.

Algorithm for our problem. Let I be an instance of MOSP, and T an integer (T will be a target makespan). We create an instance $I'(T)$ of SchedDT from I and T in the following way. We fix $n_{DT} = n$ and $m_{DT} = m$. For task t_i^j (the j-th task of Organization i), which is of length p_i^j in I, we create in $I(T)'$ a task t_k, with $k = (\Sigma_{x=1}^{i-1} n_x)+i$ (i.e. to each task of I is associated a task in $I'(T)$). We set: $p_k = l_i^j$, $r_k = 0$, and $q_k = \max\{0, T - C_{loc}^i\}$. The idea is the following one: tasks are available at date 0, and a task of Organization O_i should be scheduled before the local makespan of O_i, C_{loc}^i. Whereas the lengths will be rounded, we will not round down the values q in the PTAS if the number of organizations is fixed (in this case, there will be a fixed number of sizes q – at most N, the number of organizations –, and this will allow us to better bound the deterioration of the local makespans of the organizations). Once this reduction has been done, we use the above described PTAS of Hall and Shmoys with instance $I'(T)$ (the only differences between the original PTAS and our utilization of it is that the values q are not rounded – if N is fixed –, and that the instance $I'(T)$ slightly differs at each step of the dichotomic search since the values q are a function of T).

We do a dichotomic search over the target makespan T in the interval $[LB, UB]$, where $LB = \max\left\{\max_{i,j} p_i^j, \frac{\Sigma_{i=1}^N \Sigma_{j=1}^{n_i} p_i^j}{m}\right\}$ and UB is the makespan of the schedule returned by a greedy 2-approximate algorithm for MOSP [Cohen et al., 2011b] (UB is the makespan of a schedule without idle times, so we have $UB \leq \frac{\Sigma_{i=1}^N \Sigma_{j=1}^{n_i} p_i^j}{m} + \max_{i,j} p_i^j \leq 2LB$). Note that $\max_{i,j} p_i^j$ and $\frac{\Sigma_{i=1}^N \Sigma_{j=1}^{n_i} p_i^j}{N}$ are lower bounds of OPT (since $\max_{i,j} p_i^j$ is the length of the longest task, and $\frac{\Sigma_{i=1}^N \Sigma_{j=1}^{n_i} p_i^j}{m}$ is the average load of a machine), and thus the maximum of the two $LB = \max\left\{\max_{i,j} p_i^j, \frac{\Sigma_{i=1}^N \Sigma_{j=1}^{n_i} p_i^j}{m}\right\}$ is a lower bound of OPT. Let us denote ApproxViaDT(ϵ) this algorithm.

Proposition 2.3.2: PTAS approximation ratio

Let $\epsilon > 0$. If the number of organizations is fixed, Algorithm ApproxViaDT(ϵ) returns a schedule of makespan at most $(1+\epsilon)OPT$ in which each organization $i \in \{1, \ldots, N\}$ has a makespan at most $(1+\epsilon)C_{loc}^i$.

Proof. Let us first show that Algorithm ApproxViaDT(ϵ), returns a schedule of makespan at most $(1+\epsilon)OPT$.

Let us consider an instance I of MOSP, and let us denote by OPT the value of its optimal makespan. Let us consider a target makespan T examined at a given step of the dichotomic search. For this value T, the algorithm either returns a schedule of value $T(1+\epsilon)$, or assures that there is no schedule of value at most T. If $T = OPT$, then, $OPT_{DT} \leq OPT$, where OPT_{DT} is the makespan of an optimal solution of instance $I'(T)$ of the scheduling problem with delivery times. Indeed, let us consider an optimal schedule for MOSP, and let us view it from the viewpoint of SchedDT. For each task k of Organization O_i, $q_k = \max\{0, OPT - C^i_{loc}\}$. In a feasible schedule for MOSP, the execution of this task k will end at most at time C^i_{loc}, and thus its completion (as defined in the scheduling problem with delivery times) will be at most at time $C^i_{loc} + q \leq OPT$. Therefore, there is a feasible schedule of makespan OPT for instance $I'(OPT)$.

Note that, during the dichotomic search of ApproxViaDT(ϵ), if there is a solution for instance $I'(T)$ for problem SchedDT then there is no solution for instances $I'(T')$ with $T' < T$ (by construction) and there are solutions for instances $I'(T')$ with $T' > T$ since a solution for instance T will be a solution for instance T' (the values q increase at most by $T' - T$, while the makespan also). Therefore, ApproxViaDT(ϵ), which returns a schedule $(1+\epsilon)$-approximate for SchedDT, will return a solution of makespan at most $(1+\epsilon)OPT$.

Let us now show that in the returned solution, the makespan of each organization is at most $(1+\epsilon)C^i_{loc}$. Recall that the makespan of the schedule returned by the PTAS for SchedDT (for the final target makespan T) is at most $(1+\epsilon)T \leq (1+\epsilon)OPT$. Recall also that the values q have not been rounded, and that the value q of a task of O_i is 0 if $C^i_{loc} > T$ and $T - C^i_{loc}$ otherwise. The schedule of the tasks of types in X (tasks with rounded sizes, or small tasks glued into small components) has a makespan at most T. The factor $(1+\epsilon)$ is obtained when we replace these tasks by the tasks with their real lengths. Since the values q have not been rounded down, a task of O_i will end, when considering the schedule with the rounded sizes, at time at most C^i_{loc} if $T < C^i_{loc}$, and at most $T - q = C^i_{loc}$ otherwise: in both cases, its execution ends at time at most C^i_{loc}. By replacing the tasks of X by the true tasks, each completion time may be increased by factor $(1+\epsilon)$. Therefore, we obtain a schedule in which the execution of each task of O_i ends at most at time $(1+\epsilon)C^i_{loc}$. This concludes the proof. □

Note that, if the number of organizations is not fixed, we can use the same algorithm, by rounding the values q (as in the original PTAS for SchedDT). This will return a schedule of makespan at most $(1+\epsilon)OPT$ and in which each organization has a makespan at most $C^i_{loc} + \epsilon OPT$.

2.4 Efficiency vs. increase of the local makespans

In this section, we study how the aim of minimizing the makespan is in opposition with the rationality constraint. We start, in Section 2.4.1, to show that if we want to return a schedule optimal for the makespan, then we may have to increase the local makespans up to a factor $m - 1$. Since it is unlikely that the organizations agree to

increase their local makespan of such a large factor, in Section 2.4.2, we assume that each organization agrees to increase its makespan by a factor $(1+\alpha)$, with $\alpha \geq 0$. We then look at the increase of the makespan in function of α (when $\alpha = 0$, the problem is MOSP, the higher α is, the more relaxed the rationality constraint is).

Note that, contrarily to what we have done in Section 2.3.2, in this section, we compare the makespan of an optimal solution of $(1+\alpha)$-MOSP to the optimal makespan *without the rationality constraint*, $OPT^{(\bar{r})}$. The algorithm of Section 2.3.2 returns a schedule close to OPT, the optimal solution of $MOSP$, but not necessarily close to $OPT^{(\bar{r})}$ (this can be very different, since, as we will see in the sequel, OPT, can be twice larger than $OPT^{(\bar{r})}$).

2.4.1 The aim is to minimize the makespan: impact on the local makespans.

We first show that in the specific case in which there are two organizations, each one having one machine, we have $OPT = OPT^{(\bar{r})}$.

Proposition 2.4.1: Particular case - $N = 2$

When $N = 2$ and $m_1 = m_2 = 1$, any optimal solution for MOSP is also optimal for $(P||C_{\max})$.

Proof. We assume $N = 2$, $m = 2$ and $m_1 = m_2 = 1$. Let us assume without loss of generality that $C^1_{loc} \leq C^2_{loc}$, and let us consider $S^{*(\bar{r})}$, an optimal schedule for $(P||C_{\max})$ for such an instance. In $S^{*(\bar{r})}$, let us schedule on each machine the tasks of O_1 before the tasks of O_2. By construction, this schedule minimizes the makespan, since each machines still runs the same tasks, just in a different order, the final task still completes at the same time. Organization O_2 does not increase its makespan (otherwise the local schedules would have a makespan smaller than OPT, which is not possible); and O_1 does not increase its makespan neither since its jobs are at the beginning of the schedule on each machine and it only had one machine for its local schedule. □

This is the best case: the rationality constraint does not prevent from obtaining the best schedule concerning the makespan. This is however not always the case when $m > 2$. The following proposition shows that, in order to get a schedule minimizing the makespan, an organization may have to increase its makespan up to a factor $m-1$.

Proposition 2.4.2: Cost of efficiency

In a schedule which minimizes the makespan of the tasks of $\mathcal{T}$ on m machines, an organization may necessarily increase its makespan up to a factor $m-1$ (compared to its local makespan), but never up to a factor larger than m. This holds even if there are two organizations.

Proof. Let us assume, without loss of generality that the organizations are indexed by non decreasing local makespan, i.e. $C^1_{loc} \leq C^2_{loc} \leq \cdots \leq C^N_{loc}$. Let us consider an

optimal schedule of the tasks $\mathcal{T}$ for problem $(P||C_{\max})$. In this schedule, we reorder the tasks such that on each machine the tasks are scheduled by increasing number of their organizations (i.e. tasks of O_1 are scheduled before the one of O_2, and so forth). Let us denote by $\mathcal{O}$ the schedule obtained. This schedule stays an optimal schedule since the load on each machine, and thus the makespan, do not change. Let us show that for each $i \in \{1,\dots,N\}$, $C^i_{\max}(\mathcal{O}) \leq mC^i_{loc}$. Let us consider a given machine j and a task x of O_i on machine j. If it is not the first task on machine j, task x is preceded by tasks of $\{O_1,\dots,O_i\}$ on j. The load of the tasks which precede x (plus the length of x) is thus at most $\sum_{k=1}^{i} m_k C^k_{loc}$ (since the load of each organization O_k is at most $m_k C^k_{loc}$). Since the organizations are indexed by non decreasing local makespans, $\sum_{k=1}^{i} m_k C^k_{loc} \leq \sum_{k=1}^{i} m_k C^i_{loc} \leq mC^i_{loc}$. The completion time of each task of O_i in $\mathcal{O}$ is at most mC^i_{loc}. Therefore $C^i_{\max}(\mathcal{O}) \leq mC^i_{loc}$.

Let us now output an instance in which an organization has to increase its makespan up to a factor $m-1$. Consider the instance with two organizations, where O_1 has $m-1$ machines and $m-1$ tasks of length 1 (its local makespan is thus 1), and where O_2 has $m-1$ tasks of length $m-1$ and 1 machine. An optimal schedule of these tasks on m machines has a makespan of $m-1$. Indeed, in such a schedule, the tasks of O_1 are necessarily scheduled on the same machine and are completed at time $m-1$: the makespan of O_1 is increased by a factor $m-1$. This instance is showed in Figure 2.4.

□

Note that the bound of $m-1$ can be increased up to m if the organizations are allowed to own tasks but no machine. The instance showing this is almost the same than the one in the proof above (O_1 owns m machines and m tasks of length 1 and O_2 has $m-1$ tasks of length m).

We have seen that what we could call "the price of efficiency", the factor at which a local makespan may have to increase to get an optimal schedule for the makespan, is between $m-1$ and m, which is high. We can assume that organizations may accept to increase their makespans in order to get an efficient schedule, but only if this does not increase to much. In the following section, we assume that each organization agrees to increase a little bit its makespan: given a fixed value α it will accept a schedule in which its makespan is increased by a factor at most $(1+\alpha)$ compared to its local makespan.

2.4.2 The aim is to minimize the increase of the local makespans: impact on the makespan.

Let $\alpha \geq 0$. We now assume that each organization O_i agrees to have a makespan at most equal to $(1+\alpha)C^i_{loc}$. If $\alpha = 0$, this is the MOSP. Otherwise, it means that each organization agrees to increase a little bit its makespan (the higher α is, the higher an organization agrees to increase its makespan). We call $(1+\alpha)$-MOSP, the problem where we wish to minimize the makespan with these relaxed constraints:

$$\text{minimize } C_{\max}(S) \text{ such that, for each } i \in \{1,\dots,N\}, C^i_{\max}(S) \leq (1+\alpha)C^i_{loc}.$$

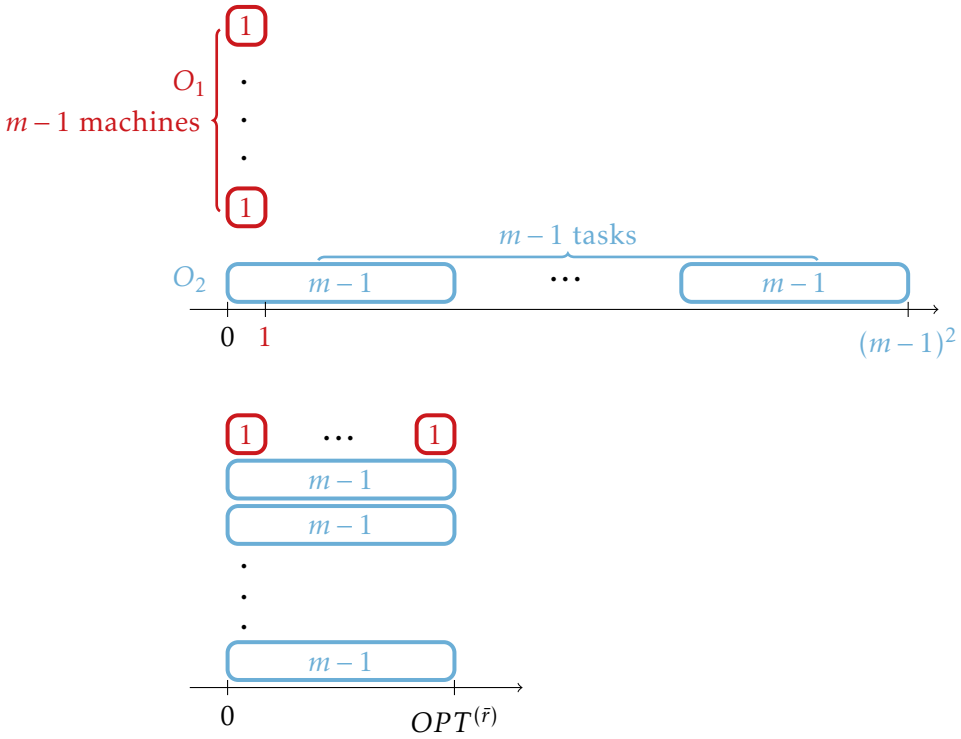


Figure 2.4: Example of an instance in which O_1 has to increase its makespan by a factor $m-1$ in an optimal solution for $(P||C_{\max})$ with regards to its local makespan. Local schedules (top), optimal schedule for $(P||C_{\max})$ (bottom).

Our aim is to give a lower bound on the approximation ratio of an algorithm for $(1+\alpha)$-MOSP with respect to the optimal makespan $OPT^{(\bar{r})}$: this will show what we loose, in term of makespan, due to the relaxed rationality constraint.

The bound we introduce in Proposition 2.4.3 implies, as we will see with Corollary 2.4.1, that when $\alpha = 0$ (in the usual MOSP context), there is no algorithm less than 2-approximate. We also show in Figure 2.6 how this ratio evolves when the number of machines and α increase.

Proposition 2.4.3: Relaxed rationality constraint

Let $\alpha \geq 0$, $\varepsilon > 0$. If each organization accepts to increase its makespan by a factor $(1+\alpha)$, there is no $(\max_{k\in\{\left\lfloor\sqrt{\frac{\alpha m^2+m}{1+\alpha}}\right\rfloor,\left\lceil\sqrt{\frac{\alpha m^2+m}{1+\alpha}}\right\rceil\}}\left(1+\frac{(m-k)(k(1+\alpha)-m\alpha-1)}{k(m-1)}\right)-\varepsilon)$-approximate algorithm with respect to the global makespan.

Proof. Given m machines, and $k \in \{1,\ldots,m-1\}$, let us consider the following set of tasks: k tasks of length $xk(m-1)$ (these tasks are said *large*) and $n_{small} = (m-1)xk(m-k)$ tasks of length 1 (these tasks are said *small*). The optimal makespan of these tasks is $OPT = xk(m-1)$: it is obtained when each large task is alone on a machine, and the small tasks are scheduled on the $(m-k)$ remaining machines.

Let us now assume that Organization O_1 owns $m-1$ machines and all the small tasks, and that Organization O_2 owns one machine and all the large tasks. The local makespan of O_1 is then $C^1_{loc} = xk(m-k) \leq OPT$, and the local makespan of O_2 is $C^2_{loc} = xk^2(m-1) \geq OPT$.

Let S be a schedule in which each organization increases its makespan by a factor at most $(1+\alpha)$. In S, each task of O_1 (small task) is completed at the latest at time $\lfloor(1+\alpha)C^1_{loc}\rfloor = \lfloor(1+\alpha)xk(m-k)\rfloor$. Therefore, on $m-k$ machines, there are at most $\lfloor(1+\alpha)xk(m-k)\rfloor$ tasks of length 1, and the other small tasks are on the k remaining machines. The minimal number of small tasks to schedule on the k remaining machines is $n_{small} - (m-k)\lfloor(1+\alpha)xk(m-k)\rfloor = (m-1)xk(m-k) - (m-k)\lfloor(1+\alpha)xk(m-k)\rfloor$. On one of these k machines, there is at least $1/k$ of these tasks, that is $(m-1)x(m-k) - \frac{(m-k)\lfloor(1+\alpha)xk(m-k)\rfloor}{k} \geq (m-1)x(m-k) - (1+\alpha)x(m-k)^2$. If there are at least two large tasks on the same machine, the makespan is at least equal to $2(xk(m-1)) = 2OPT$. Otherwise, there are at most one large task by machine. The makespan of such a schedule is then at least the length of a large task plus the length of the small tasks. This is larger than or equal to $xk(m-1) + (m-1)x(m-k) - (1+\alpha)x(m-k)^2$. The approximation ratio is thus at least $\frac{xk(m-1)+(m-1)x(m-k)-(1+\alpha)x(m-k)^2}{xk(m-1)} = 1 + \frac{(m-k)(k(1+\alpha)-m\alpha-1)}{k(m-1)}$.

By deriving $f(k) = 1 + \frac{(m-k)(k(1+\alpha)-m\alpha-1)}{k(m-1)}$ (with $k \in [1,+\infty)$), we find that the value of k which maximizes $f(k)$ is $k = \sqrt{\frac{\alpha m^2+m}{1+\alpha}}$.

Since $f(k)$ is an increasing function between $[1, \sqrt{\frac{\alpha m^2+m}{1+\alpha}}]$ and a decreasing function

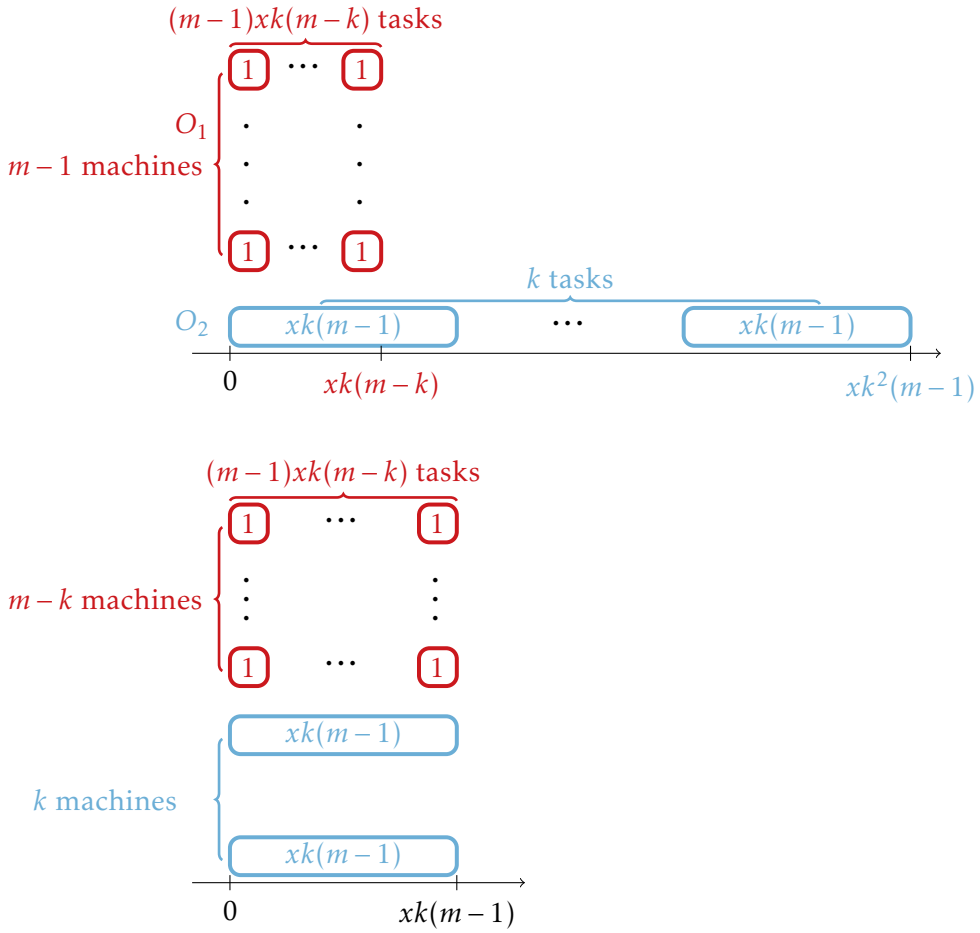


Figure 2.5: Instance giving the approximation ratio. Local schedules (top) and optimal solution for $(P\|C_{\max})$ (bottom).

in $[\sqrt{\frac{\alpha m^2+m}{1+\alpha}}, +\infty)$, the maximum value of $f(k)$ when k is an integer is:

$$\max_{k\in\left\{\left\lfloor\sqrt{\frac{\alpha m^2+m}{1+\alpha}}\right\rfloor,\left\lceil\sqrt{\frac{\alpha m^2+m}{1+\alpha}}\right\rceil\right\}} \left(1+\frac{(m-k)(k(1+\alpha)-m\alpha-1)}{k(m-1)}\right).$$

□

When $\alpha = 0$, the value of k which maximizes the ratio ($f(k)$) is $\lceil\sqrt{m}\rceil$ or $\lfloor\sqrt{m}\rfloor$. When $\sqrt{m}$ is an integer, there is no algorithm for MOSP which returns $\left(1+\frac{m-2\sqrt{m}+1}{m-1}-\varepsilon\right)$-approximate schedules with respect to the global makespan. This tends towards 2 when m tends towards the infinity, which leads to the following corollary.

Corollary 2.4.1: Cost of rationality

Let $\epsilon > 0$. There is no algorithm which returns schedules which fulfill the rationality constraint, and which is $(2-\epsilon)$-approximate with respect to the global makespan $OPT^{(\vec{r})}$.

This bound improves the previous one, $\frac{3}{2}$, which had been given by Pascual et al. [2007] for two organizations and by Cohen et al. [2010] for more than two organizations. Furthermore, Cohen et al. [2011a] show that no approximation algorithm for MOSP has a ratio asymptotically better than 2 w.r.t. the global makespan (when m tends towards the infinity) when we add the constraint that on the returned schedule, each machine schedules the tasks of its organization (if any) before the tasks of other organizations. This constraint is thus not necessary to obtain the asymptotic ratio of 2.

When m tends towards the infinity and $\alpha > 0$ the value of k maximizing $f(k)$ is then $m\sqrt{\frac{\alpha}{\alpha+1}}$. In that case we can express the approximation ratio depending on only α as $2+2\alpha-(\alpha+1)\sqrt{\frac{\alpha}{\alpha+1}}-\frac{\alpha}{\sqrt{\frac{\alpha}{\alpha+1}}}$. The value $\sqrt{\frac{\alpha}{\alpha+1}}$ quickly increases with α and tends towards 1 when α tends towards the infinity. This means that this ratio is close to 2 when α is close to 0, and it quickly decreases and tends towards 1.

Figure 2.6 shows the lower bound given in Proposition 2.4.3. This ratio is given as a function of α (Left), or of the number of machines, m (Right). The higher m is, the higher the ratio is. When α increases, this ratio decreases quickly. The first points of the curves in Figure 2.6 Left shows the lower bound of the ratio between the best makespan in a schedule satisfying the rationality constraint, and the best makespan without this constraint (as seen above, this ratio tends towards 2 when m increases). This ratio when $\alpha = 0$ can also be seen in the blue curve of Figure 2.6 Right.

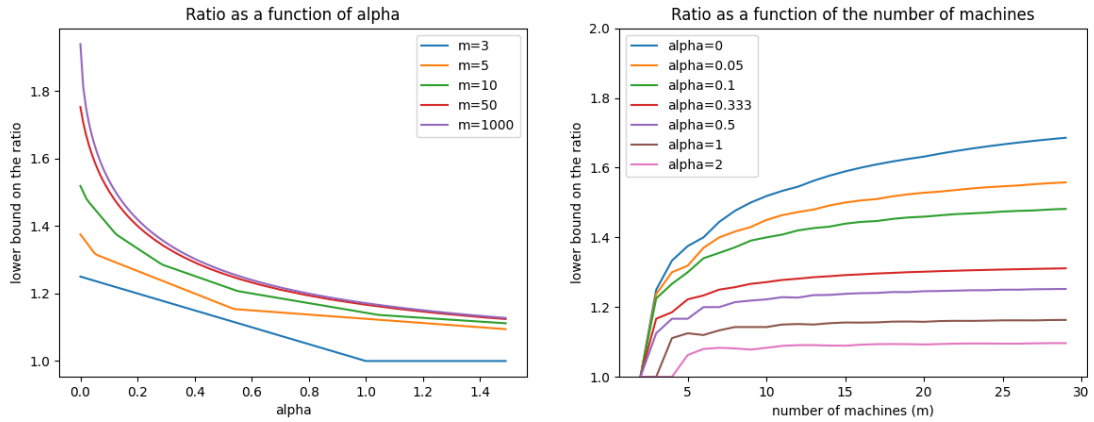


Figure 2.6: Each organization accepts to decrease its makespan by a factor $(1+\alpha)$. Lower bound on the ratio between the best possible makespan when no organization increases its makespan by a factor larger than $(1+\alpha)$, and the optimal makespan.

We end this section by mentioning that we can easily adapt the algorithm described in Section 2.3.2 to the case of $(1+\alpha)$-MOSP: whereas, for a target makespan T, we had set the delivery time of a task of Organization O_i to $q = \max\{0, T - C^i_{loc}\}$ (so that this task is completed at time C^i_{loc} in the returned schedule of rounded tasks), we fix this value to $q = \max\{0, T - (1+\alpha)C^i_{loc}\}$ in the case of $(1+\alpha)$-MOSP. We thus get, for any fixed $\epsilon > 0$, a polynomial time algorithm returning a schedule of makespan at most $(1+\epsilon)$ times the makespan of an optimal solution of $(1+\alpha)$-MOSP, and in which the makespan of each organization is at most $(1+\epsilon)(1+\alpha)$ its local makespan.

In the previous sections, we have assumed either that the rationality constraint should be fulfilled (but we then had as only objective function to minimize the global makespan, and the gains for the organizations – the decrease of their makespans – in the returned schedule could be very different), or we have even assumed than we can relax (in a bounded way) the rationality constraint to get a schedule with an even smaller makespan. In the following section, we focus on fairness issues: we will keep the rationality constraint, and our focus will not be to decrease the makespan, but to get schedule in which *all* the organizations decrease their makespans by a factor as large as possible.

2.5 Max Min Gain

2.5.1 Problem statement

Let us first define the gain $g^i(S)$ of Organization O_i in a schedule S: $g^i(S)$ represents how much Organization O_i has decreased its makespan in the schedule S in comparison

to its local schedule:

$$g^i(S) = \frac{C^i_{loc} - C^i_{\max}(S)}{C^i_{loc}}.$$

Note that this value is 0 when organization O_i has the same makespan in S and in its local schedule (the ratio would be 1 if O_i got a makespan of 0 in schedule S). Expressing the gain in that way allows to have a scale from 0 to 1 for all organizations, 0 meaning that the organization does not gain anything in comparison to its local schedule and 1 being a (potentially unreachable) situation in which an organization gets a makespan of 0. Intuitively, if the makespan of an organization is divided by $x \geq 1$, then its gain is $\frac{x-1}{x}$. For example if an organization has a makespan 2 times smaller in S in comparison to its local makespan, then its gain is $1/2 = 0.5$

The Maximal Minimal Gain problem, denoted as MaxMinGain, takes the same input as MOSP. The output is a schedule of the n tasks of all the organizations on the m machines of the organizations, in order to maximize the minimum gain among the organizations. The returned schedule is thus $S^* = \arg\max_S \min_{i\in\{1,\dots,N\}} g^i(S)$

Note that the schedule S_{loc} which is made of N local schedules has a minimum gain of 0, which means that an optimal schedule for MaxMinGain always has a minimum gain larger than or equal to 0 and satisfies the rationality constraint.

Given a considered instance I, we will denote by S^* an optimal solution for the problem MaxMinGain, and we will denote by OPT the minimum gain among the organizations in such a solution. In the sequel, we will be interested in comparing the makespan $C_{\max}(S^*)$, or the makespan returned by an algorithm, to the best solution without the rationality constraint, that we will denote by $S^{*(\bar{r})}$.

Note that our definition of the gain is very close to the definition of utility used in a paper by Agnetis et al. [2019]. In the work of Agnetis et al., two agents, each one owning a subset of tasks, share a single machine. The two agents A and B have different objective functions f^A and f^B. The utility of agent A in a schedule S is defined as $f^A_\infty - f^A(S)$, where f^A_∞ denotes the value of f^A when the subset of tasks of A is scheduled after the subset of tasks owned by B, which is the worst case for A. Even though the contexts are different, the idea is the same: we evaluate individual satisfaction by comparing a worst case for the agent to the current solution. In our case, for each organization O_i, we compare the makespan obtained by O_i in a schedule to the worst makespan O_i could have, and this worst makespan is its local makespan, C^i_{loc}, since the schedule should fulfill the rationality constraint.

2.5.2 Case of unit tasks

In this section, we show that problem MaxMinGain can be solved in polynomial time when all the tasks have the same length. Moreover, in this case, it is possible to find a schedule S which is optimal for MaxMinGain and optimal for problem $(P||C_{\max})$: the global makespan is minimized while the minimal gain of an organization is maximized. Let us now present the following algorithm which returns such a schedule. This algorithm, called LS-IM (for List Scheduling by Increasing local Makespan), is a

list scheduling algorithm: it greedily schedules all the tasks, considering the tasks by increasing local makespans of their owners:

Algorithm LS-IM

Sort the organizations by non decreasing local makespans. If two organizations have the same local makespan, sort them by non decreasing number of tasks, if these two organizations have the same number of tasks, sort them in any order. Let $O_{x_1}, \ldots, O_{x_N}$ be the result of this sort.
for i=1 **to** N **do**
  for *each task* $t^j_{x_i} \in \mathcal{T}_{x_i}$ **do**
    schedule $t^j_{x_i}$ on the first available machine;
  end
end

Algorithm 1: List scheduling by increasing local makespans (Algorithm LS-IM)

Observation 2.5.1: Complexity of LS-IM

LS-IM as presented above only runs in pseudopolynomial time. Indeed, when all tasks have the same processing time, an instance can be described with 2 integers per organization: the number of tasks and the number of machines that it owns. In that case, the size of the instance is of $2N$ integers, whereas LS-IM runs in $O(n + N \log N)$ since we need to sort the organizations by increasing local makespan and then schedule the tasks one by one. It is possible to obtain the same schedule without scheduling the tasks one by one since all the tasks from the same organization are scheduled "together". We start by sorting the organizations by increasing local makespan. Then for each organization, we compute the euclidian division of its number of tasks over the total number of machines (n_i/m). The result of the division gives us a number of slot needed on all the machines while the rest ($n_i \mod m$) gives us the number of additional tasks to be scheduled. These tasks can be scheduled as a block since no tasks from another organization should be scheduled in the middle of these tasks. This algorithm runs in $O(N \log N + N \log^2 n)$ and is thus a polynomial time algorithm.

Figure 2.7 shows an instance with local schedules on top and the schedule obtained by running LS-IM on bottom. It is easy to see that, in the schedule obtained with LS-IM, to increase the gain of organization O_3 (with green tasks), it would be necessary to delay one of the tasks from O_2 (yellow tasks) after the makespan of O_3. This would greatly lower the gain of O_2 and also lower the overall minimum gain. We now show that this applies more generally and that LS-IM is optimal for MaxMinGain when all tasks have the same processing time.

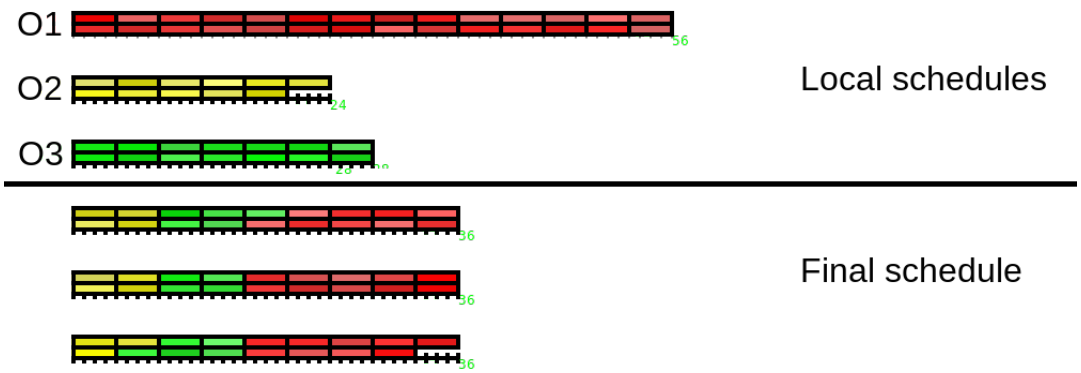


Figure 2.7: Example of an execution of LS-IM on an instance with $N = 3$ organizations

Proposition 2.5.1: Unit task MaxMinGain

When all the tasks have the same processing time, Algorithm LS-IM returns a schedule which is optimal for MaxMinGain and optimal for $(P||C_{\max})$.

Proof. Let us assume that the organizations are labelled such that $C^1_{loc} \leq C^2_{loc} \leq \cdots \leq C^N_{loc}$ and that, for all $l \in \{2,\ldots,N\}$, if $C^l_{loc} = C^{l-1}_{loc}$, then $n_l \geq n_{l-1}$. We also assume all the tasks have the same length.

Let us suppose, for the sake of contradiction, that the schedule S returned by algorithm LS-IM is not optimal for MaxMinGain. Let O_k be an organization which gets a minimal gain in S. In order to increase the gain of O_k, we should build a schedule S' in which $C^k_{\max}(S') < C^k_{\max}(S)$. Tasks all have the same length and there is no idle time in S: there is not enough slots so that all the tasks of $O_1,\ldots,O_k$ are completed before time $C^k_{\max}(S)$. Therefore, in S', a task of O_l, with $l < k$ will be completed at time at least $C^k_{\max}(S)$. The gain of O_l in S' will thus be at most $\frac{C^l_{loc}-C^k_{\max}(S)}{C^l_{loc}} \leq \frac{C^k_{loc}-C^k_{\max}(S)}{C^k_{loc}}$: the minimal gain in S' is smaller than or equal to the minimal gain in S. Therefore, S is optimal for MaxMinGain.

Schedule S has no idle time and all the tasks are the same length, therefore if a task t^j_i starts at time t in S, all machines are busy at least until t, which means that at least one tasks has to start at t or later, this is in particular true for the last task executed in S: the schedule S is then also optimal for $(P||C_{\max})$. □

We showed that, in the particular case where all tasks have the same processing time, we can find a polynomial time algorithm which builds a schedule which both minimizes the global makespan and maximizes the minimal gain of an organization. In this special case we do not have to compromise between global optimization and individual satisfaction. Unfortunately, this result does not hold in the general case, as we will see in the following section.

2.5.3 General case

In this section, we study MaxMinGain in the general case. We fist show that MaxMinGain is NP-hard and hard to approximate.

Proposition 2.5.2: MaxMinGain inapproximability

If $P \neq NP$, problem MaxMinGain is NP-hard and inapproximable in polynomial time, even if there are only two organizations and two machines.

Proof. Let $r > 1$. By contradiction, let us assume that $P \neq NP$ and that there exists a polynomial time r-approximate algorithm for MaxMinGain. We will show that this algorithm allows us to solve the NP-complete Partition problem. The Partition problem is the following one: given a set $S = \{a_1, \ldots, a_k\}$ of k positive integers such that $\Sigma_{i=1}^{k} a_i = 2B$, is it possible to partition S into two subsets S_1 and S_2 such that $\Sigma_{a_i \in S_1} a_i = \Sigma_{a_i \in S_2} a_i = B$?

We will exhibit an instance for which the maximum minimal gain is strictly greater than 0 if and only if there is a yes answer to the Partition problem. Note first that, if this is true, then our r-approximate algorithm allows us to solve the Partition problem. Indeed, if there is a yes answer to the Partition problem then the maximal minimal gain is $OPT > 0$: a r-approximate algorithm should return a solution in which the gain of each organization is a least $rOPT > 0$. If the answer to the Partition problem is 'no' then $OPT = 0$, and any algorithm, including the r-approximate algorithm, will return a solution with minimal gain 0. Therefore, the r-approximate algorithm permits to determine whether the answer to the partition problem is positive or not. Since this r-approximate algorithm is a polynomial time algorithm, this implies that $P = NP$, a contradiction.

Let us now consider the following instance of MaxMinGain, and show that, for this instance, there is a yes answer to the Partition problem if and only if the maximum minimal gain, OPT, is strictly greater than 0. There are two organizations, each one having a single machine. Organization O_1 owns k tasks $t_1^1, \ldots, t_k^1$ such that for each $i \in \{1, \ldots, k\}$, the processing time of task t_i^1 is equal to a_i. Organization O_2 owns 2 tasks, each of length $B+1$. The local makespan of O_1 is thus $2B$, while the local makespan of O_2 is $2B+2$. Figure 2.8 shows such an instance.

Let us first consider that answer of the Partition problem is 'yes'. Therefore, there exists a partition (S_1, S_2) of the tasks of O_1 such that $\sum_{t_i^1 \in S_1} p_{t_i^1} = \sum_{t_i^2 \in S_2} p_{t_i^2} = B$. By scheduling the tasks of S_1 followed by a task of O_2 on a machine, and the tasks of S_2 followed by the second task of O_2 on the second machine, the makespan of O_1 is B, while the makespan of O_2 is $B + (B+1) = 2B+1$. Since the local makespan of O_1 is $2B$ and the local makespan of O_2 is $2B+2$, both organizations have a gain strictly greater than 0 (O_1 decreases its makespan by a factor 2, and O_2 by a factor $\frac{B+2}{B+1}$).

Let us now consider that the answer of the MaxMinGain problem is 'yes'. This implies that the makespan of organization O_2 is equal to or lower than $2B+1$. It also means that the makespan of organization O_1 is equal to or lower than $2B-1$. Since

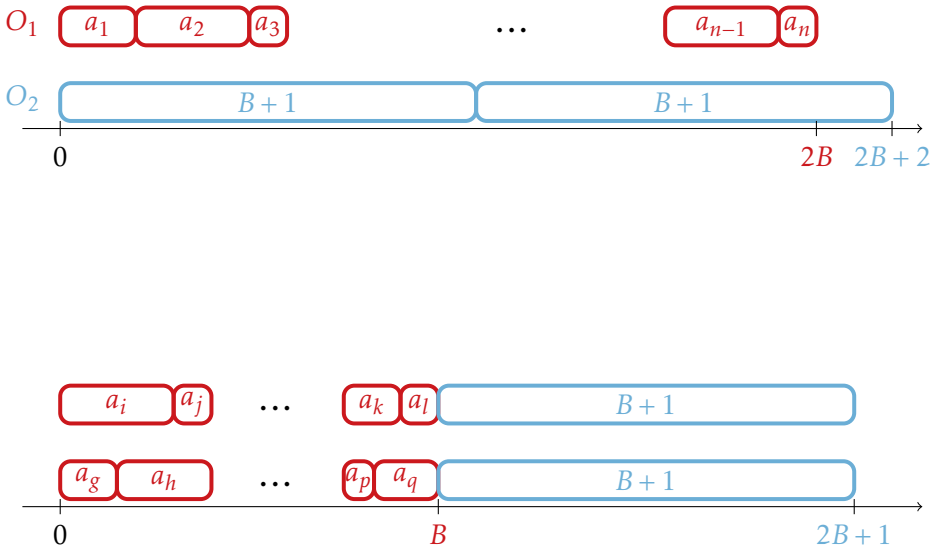


Figure 2.8: Local schedules (top). A solution with a minimum gain strictly greater than 0 (bottom). This can only be achieved if we can partition the tasks of O_1 into two sets, each of total processing time B.

O_2 has two tasks of processing time $B+1$ and O_2 has a makespan of $2B+1$ or less, its two tasks need to be scheduled on different machines and to start at the latest at time B. Organization O_1 has a total load of $2B$. Since tasks of O_2 start at the latest at time B and each occupy $B+1$ units of time on a machine, it means that there are at most B units of time on each machine between 0 and $2B-1$ for tasks of O_1, and this only if tasks of O_2 start exactly at time B. To have a makespan lower than or equal to $2B-1$ organization O_1 then needs to schedule its $2B$ load of tasks such that there is a load of total processing time B on each machine, otherwise either the task of O_2 starts later and O_2 has a gain of 0 or a task of O_1 starts after a task of O_2 and the gain of O_1 is 0. Since the processing times of the tasks of O_1 are the same than the values of the Partition problem, if in a schedule, the tasks of O_1 are split in such a way that there is a load of B units of time on each machine, it means that it is possible to partition the integers of the Partition problem in two subsets, each of total sum B. Therefore, the answer to the Partition problem is then 'yes'.

We have shown that the minimal gain is strictly greater than 0 if and only if the answer of the Partition problem is 'yes': this concludes the proof. □

Let us now show that MaxMinGain is strongly NP-hard. This implies that there is no pseudo-polynomial algorithm to solve it. The proof however supposes that the number of machines is not fixed, whereas the previous proof holds when $m = 2$.

Proposition 2.5.3: MaxMinGain strong NP-hardness

Problem MaxMinGain is strongly NP-hard, even if there are only two organizations.

Proof. Let us reduce the NP-complete problem 3-Partition to the decision version of MaxMinGain. The 3-Partition problem is the following one: given a set $S = \{a_1, \ldots, a_{3k}\}$ of $3k$ positive integers such that $\Sigma_{i=1}^{3k} a_i = kB$, with $B \in \mathbb{N}$, is it possible to partition S into k subsets $\{S_1, \ldots, S_k\}$ such that for each $i \in \{1, \ldots, k\}, \Sigma_{a_j \in S_i} a_j = B$?

Our problem is the following one: given the local schedules of N organizations, and given a value X between 0 and 1, is is possible to create a schedule S of all the tasks on all the machines such that the gain of any organization is at least X, meaning: $\frac{C_{loc}^i - C_{\max}^i(S)}{C_{loc}^i} \geq X, \forall i$?

We create an instance of the MaxMinGain problem from the instance of 3-Partition as follows: there are two organizations, O_1 and O_2. Organization O_1 owns one machine and $3k$ tasks $t_1, \ldots, t_{3k}$ such that for each $i \in \{1, \ldots, 3k\}$, the processing time of task t_i is equal to a_i. Organization O_2 owns $k-1$ machines and k tasks, each of length kB. We set $X = \frac{k-1}{2k}$. Such an instance is shown in Figure 2.9.

Let us show that there is a yes answer to the 3-Partition problem if and only if the answer of the corresponding instance of MaxMinGain is also 'yes'.

Let us first consider that the answer of the 3-Partition problem is 'yes': there exists a partition $(S_1, \ldots, S_k)$ of the tasks of O_1 such that for each $i \in \{1, \ldots, k\}, \Sigma_{a_j \in S_i} a_j = B$. For each $i \in \{1, \ldots, k\}$, by scheduling on machine i the tasks corresponding to the numbers of S_i followed by a task of O_2, the makespan of O_1 is B, while the makespan of O_2 is $B + kB = (k+1)B$. Since the local makespan of O_1 is kB and the local makespan of O_2 is $2kB$, the gain of O_1 is $\frac{k-1}{k} \geq \frac{k-1}{2k}$ and the gain of O_2 is then $\frac{2kB - B(k+1)}{2kB} = \frac{k-1}{2k} = X$: the answer to the decision problem of MaxMinGain is 'yes'.

Let us now consider that the answer to the decision problem of MaxMinGain is 'yes'. The gain of each organization is at least $X = \frac{k-1}{2k}$. This means that the makespan of O_i in S is at most $C_{loc}^i(1-X)$, that is $kB - kB\frac{k-1}{2k} = B(\frac{k+1}{2})$ for O_1 and $2kB - 2kB\frac{k-1}{2k} = (k+1)B$ for O_2. Thus the global makespan is at most $(k+1)B$. Therefore, there is necessarily one task of O_2 on each of the k machines. Since the global makespan is at most $(k+1)B$ and since the total load is $(k+1)kB$, then there is necessarily a load of $(k+1)B$ on each of the k machines. The load due to the tasks of O_2 on each machine is kB, so the load due to the tasks of O_1 is B on each machine. It is therefore possible to partition the numbers $t_1, \ldots, t_k$ into k sets of weight B: the answer of the 3-Partition problem is 'yes'. □

We showed that when the tasks have the same lengths, there is always a schedule which is both optimal for MaxMinGain and for the minimization of the makespan (problem $(P||C_{\max})$). It is easy to note that, in the general case, we can obtain an optimal solution S^* of MaxMinGain that is also 2-approximate for $(P||C_{\max})$. Since every schedule with no idle time is 2-approximate for $(P||C_{\max})$, we can obtain such a schedule from S^* by removing idle times between tasks and by advancing any task on a machine

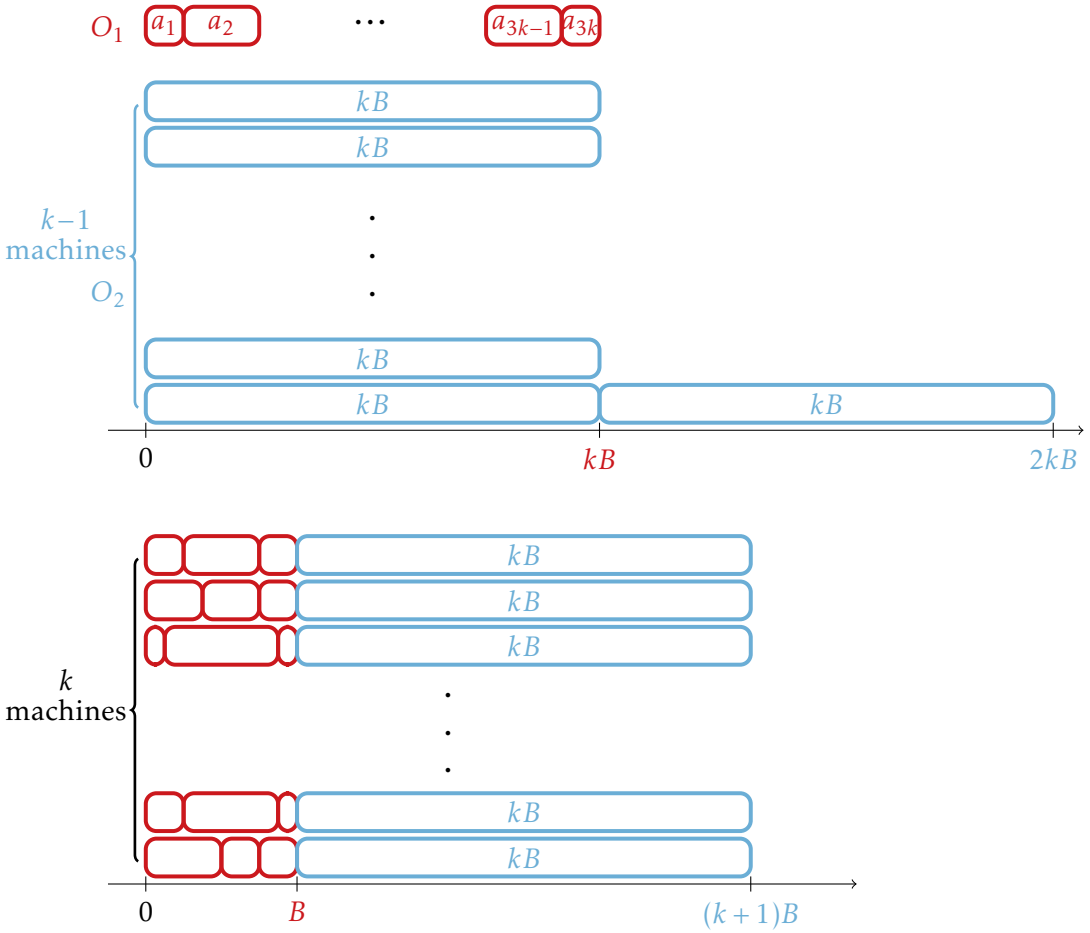


Figure 2.9: Local schedules (top). A solution with a minimum gain of X (bottom). This can only be achieved if we can partition the tasks of O_1 into k sets, each of total processing time B.

available before the starting time of the task. By doing this, we do not delay any task, so every organization has at least the same gain as in S^*, and this new schedule is thus still optimal for MaxMinGain. This schedule does not contain any idle time before that the last task starts to be executed, and is thus 2-approximate for $(P||C_{\max})$.

Let us now show that there is no algorithm which is optimal for MaxMinGain and which has an approximation ratio smaller than 2 for MOSP. Naturally, this implies that no algorithm can be optimal for MaxMinGain and have an approximation ratio smaller than 2 for $(P||C_{\max})$.

Proposition 2.5.4: Cost of fairness

Let $m \geq 4$ and $\epsilon > 0$. There is no algorithm which is optimal for MaxMinGain and $(2 - \frac{7}{m+3} - \epsilon)$-approximate for MOSP.

Proof. Let us consider the following instance, with two organizations. Organization O_1 owns $m-1$ machines and m^2 tasks of length 1. Organization O_2 owns one machine, 2 tasks of length $m-1$ and one task of length 3. O_1's local makespan is $m+2$ and O_2's local makespan is $2m+1$.

In an optimal schedule S_{MMG} for MaxMinGain, the m^2 tasks of O_1 are scheduled first, followed by three tasks of O_2 on three different machines. Indeed, in S_{MMG}, the makespan of O_1 is m and the makespan of O_2 is $2m-1$, which is also the global makespan. O_1's gain is $\frac{m+2-m}{m+2} = \frac{2}{m+2}$ and O_2's gain is $\frac{2m+1-2m-1}{2m+1} = \frac{2}{2m+1}$. Since O_2 has the minimum gain, in order to increase the minimum gain we should decrease O_2's makespan. This is only possible if a task of O_1 is delayed, being completed at time at least $m+1$ instead of m. The gain of O_1 would then be at most $\frac{m+2-(m+1)}{m+2} = \frac{1}{m+2}$, which is smaller than the minimal gain in S_{MMG}. Schedule S_{MMG} is thus optimal for MaxMinGain. We can also note that S_{MMG} is one with the smallest makespan among the optimal schedules for MaxMinGain.

Let us now consider S_{MOSP}, an optimal schedule for MOSP. In S_{MOSP}, the two tasks of O_2 of length $m-1$ are scheduled at time 0, first and $(m-1)(m-2)$ tasks of O_1, so that the load of every machine is $m-1$. Then, m tasks of O_1 are scheduled between $m-1$ and m. Schedule S_{MOSP} ends by the last task of O_2 (of length 3) at time m, and the remaining $2(m-1)$ tasks of O_1 on the $m-1$ other machines. In S_{MOSP}, the makespan of O_1 is thus $m+2$ and the makespan of O_2 is $m+3$. Note that S_{MOSP} fulfills the rationality constraint and is optimal for $(P||C_{\max})$.

Let r be the ratio between the makespan in S_{MMG}, the best schedule (w.r.t the minimization of the makespan) among schedules optimal for MaxMinGain, and the makespan in S_{MOSP}, optimal for MOSP:

$$r = \frac{2m-1}{m+3} = 2 - \frac{7}{m+3}.$$

Therefore, for this instance, there is no optimal schedule for MaxMinGain which has an approximation ratio better than $(2 - \frac{7}{m+3})$ for MOSP. □

When m tends towards the infinity the ratio tends towards 2: it is thus impossible to find an optimal algorithm for MaxMinGain less than 2-approximate for the makespan minimization. We have showed that MaxMinGain is strongly NP-hard, hard to approximate and that, in the general case, ensuring a fair schedule can lead to low global efficiency. We will now propose a polynomial time heuristic which, in practice, returns good solutions for both the minimum gain and the global makespan.

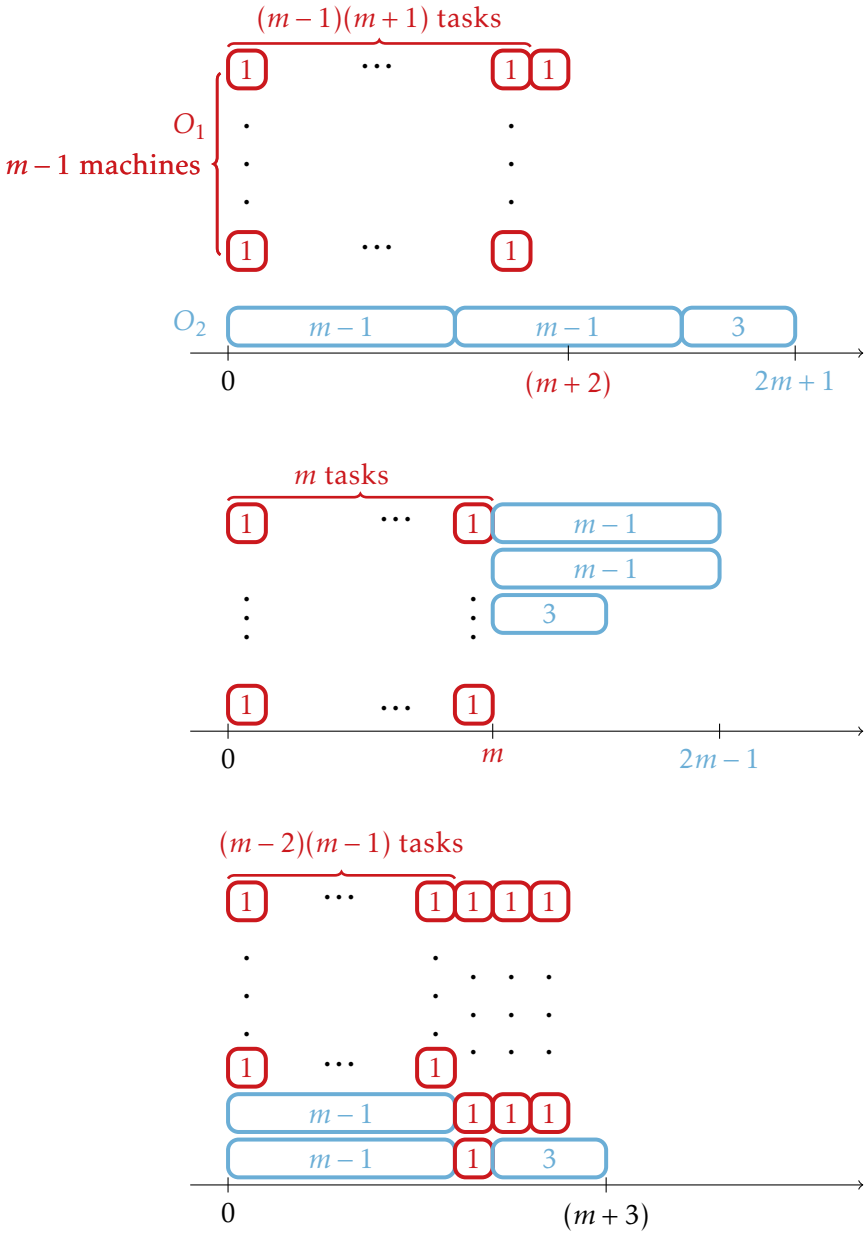


Figure 2.10: Instance giving the approximation ratio. Local schedules (top), optimal schedules for MaxMinGain (middle) and optimal solution for $(P||C_{\max})$ (bottom).

2.5.4 Heuristic

In this section, we propose a polynomial time heuristic for MaxMinGain. The idea behind this heuristic, called *MCEDD* (for MOSP Constrained Earliest Due Date), is to schedule the tasks by increasing local makespan. In such a schedule, tasks owned by the organization having the lowest local makespan are scheduled first, then the tasks of the organization having the second lowest makespan and so on. However, such a schedule does not necessarily fulfill the rationality constraint. The heuristic ensures that, at each step, the rationality constraint is fulfilled and tries to schedule tasks by increasing local makespan if it does not conflicts with this constraint.

Over this section we will illustrate the different steps of the algorithm with a running example. We will assume that the organizations are labelled in non decreasing order of their makespans: $C_{loc}^1 \leq \cdots \leq C_{loc}^N$.

Example 2.5.1: MCEDD execution: Local schedules

In the running example we will consider the following instance with five organizations:

As mentioned above, we will consider that the organizations are sorted by increasing local makespan, therefore O_1 is the organization with the dark blue tasks (on the bottom), O_2 is the organization with the light blue tasks above O_1 and so forth until O_5 at the top with the red tasks.

In this algorithm we will consider two subroutines. The first one is the list scheduling algorithm *LPT* (for Longest Processing Time), which schedules the tasks of an organization by non increasing processing time (as soon as a machine is available, the remaining task with the longest processing time is scheduled on this machine).

LPT Algorithm

Sort the tasks by non increasing processing time.
for *each task t in that order* **do**
  Find a machine m_i with a minimum load and schedule t on m_i
end

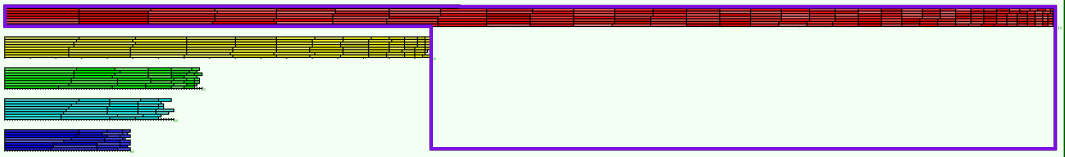
Algorithm 2: Longest Processing Time List Scheduling (LPT)

The second subroutine consists in delaying the tasks of an organization O_i in a way that no task ends after the local makespan of O_i and no task begins before the makespan of an organization with a lower makespan, unless it is scheduled on the machines of O_i. This means that the tasks are scheduled as late as possible either

between 0 and C^i_{loc} if the machine is owned by O_i or between the highest makespan among the organizations having a smaller makespan than O_i and C^i_{loc} if the machine is not owned by i (in practice, the tasks of O_i are scheduled using LPT from time C^i_{loc}, on all the machines and in the reverse order of time, under the constraint that no task should start on a machine of O_k (with $k \neq i$) before $C^{i-1}_{\max}(S)$).

Example 2.5.2: MCEDD execution: Delay subroutine

If we want to run this delay subroutine on O_5, the tasks of O_5 should be scheduled as late as possible within the purple border.



This subroutine aims at creating space at the beginning of the schedule for organizations with a low makespan without risking increasing the makespan of other organizations. It runs as follows:

Delay subroutine

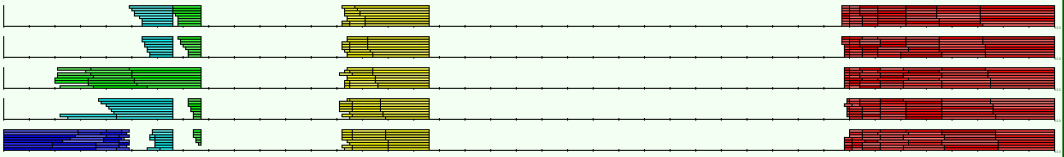
Parameters: Organization O_i, makespan $C^l_{\max}(S)$ of the organization O_l with the highest makespan among the organizations with a lower makespan than O_i
for *each machine* m_k **do**
  Set $l_{m_k} = C^i_{loc}$
end
Sort the tasks of O_i by non increasing processing time
for *each task* t_j *owned by* O_i **do**
  Find a machine m_k with maximum l_{m_k} and such that either $l_{m_k} - p_{t_j} \geq C^l_{\max}(S)$ or m_k is owned by O_i
  Schedule t_j on m_k such that it completes at time l_{m_k}
  $l_{m_k} \leftarrow l_{m_k} - p_{t_j}$
end

Algorithm 3: Delay subroutine

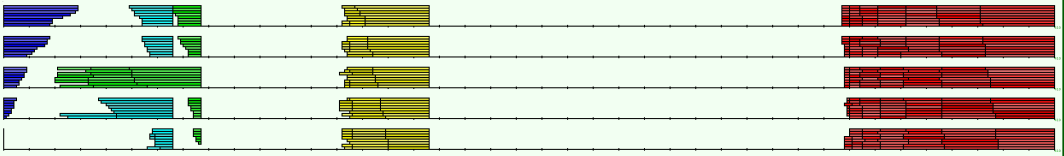
The MCEDD algorithm alternates between these two subroutines: it delays the tasks of the organizations with a high makespan, creating space at the beginning of the schedule. It then runs an LPT algorithm on the tasks of organizations with a small makespan. This LPT algorithm hopefully decreases the makespan of the organizations with a low local makespan, making it even lower. This means that there is more space for the delay subroutine, allowing to delay the tasks of the organizations with high makespans even more, creating more space and so forth.

Example 2.5.3: MCEDD execution: Alternating between the subroutines

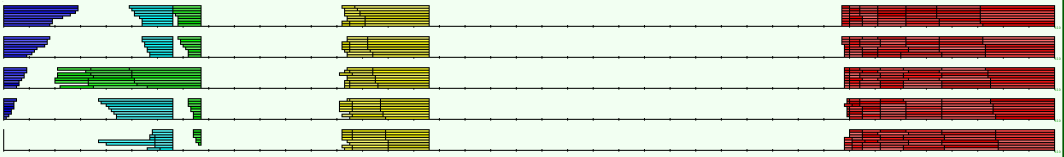
Starting from the local schedules, MCEDD starts by running the delay subroutine for all organizations except the one with the lowest local makespan, by decreasing local makespan, i.e. it delays the tasks of O_5, then O_4, then O_3 then O_2.



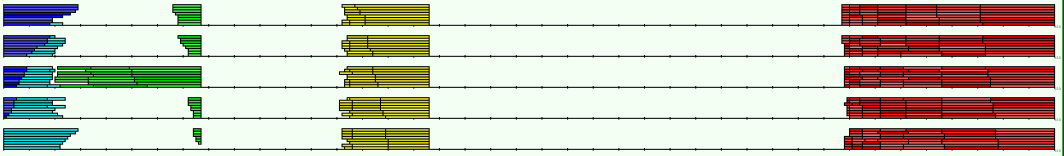
As we can see, organizations O_3 and O_2 cannot fully delay their tasks because the makespan of the organization below them, i.e. having the highest makespan among the organizations having a lower makespan, is too high. The MCEDD algorithm then runs an LPT algorithm on the tasks of O_1 on all the machines.



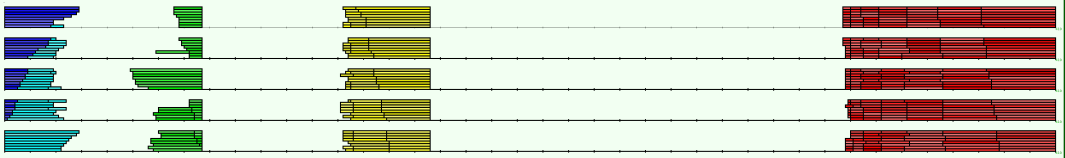
This operation lowers the makespan of O_1: we can therefore rerun the delay subroutine on O_2 with the new makespan of O_1.



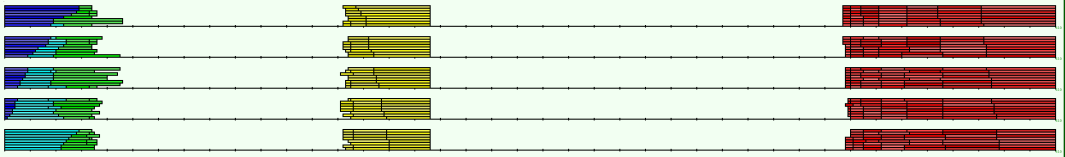
This delay may free some space on the machines owned by O_2 (although it is not the case here). The MCEDD algorithm then runs an LPT algorithm on the tasks of O_1 (since some space may have been freed) and then on the tasks of O_2.



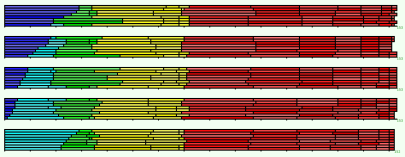
We rerun this sequence for organization O_3. First we delay its tasks:



Then we run LPT for organizations O_1,O_2 and O_3.



And we do the same for O_4 and O_5, giving us the final schedule.



The MCEDD algorithm is described as follows:

MOSP-Constrained Earliest Due Date algorithm(MCEDD)

for *i from* 2 *to* N **do**
  Use the delay subroutine with parameters i and C_{loc}^{i-1}
end
for *i from* 1 *to* N **do**
  for *j from* 1 *to* i **do**
    Use the LPT subroutine on tasks owned by O_j
  end
  if $C_{\max}^{i}(S) < C_{loc}^{i}$ *and* $i+1 \leq N$ **then**
    Use the delay subroutine with parameters $i+1$ and $C_{\max}^{i}(S)$
  end
end

Algorithm 4: MCEDD algorithm

Note that we decided to use LPT in order to schedule the tasks of an organization because of its low computational cost and its good approximation ratio, but it is possible to consider other scheduling algorithms.

Regarding complexity, we can sort the tasks by decreasing processing time as a preprocessing step, which costs $O(n \log n)$ operations. Thus the sorting steps in the

LPT and delay subroutines can be treated before starting MCEDD. Both LPT and the delay subroutine schedule greedily the task on the first available machine, finding such a machine costs $O(\log m)$ operations if we use a heap, meaning that scheduling all the n tasks in a subroutine costs $O(n \log m)$ operations. The MCEDD algorithm complexity is then determined by the two nested loops running through the organizations and running LPT. This amounts to $O(N^2 n \log m)$ operations. Adding the preprocessing part, the total complexity of MCEDD is in $O(n \log n + N^2 n \log m)$.

Note that this algorithm is 2-approximate for $(P||C_{\max})$ (and thus for MOSP) since it returns a schedule with no idle time before the start of the last task. Since MaxMinGain is hard to approximate, we have no approximation ratio for the minimum gain. Let us now evaluate this algorithm experimentally.

2.5.5 Experimental evaluation

In this section, we study the quality of the solution returned by our algorithm on randomly generated instances. To measure its efficiency, we will compare the makespan of the schedule returned by MCEDD with a lower bound of the optimal makespan. Let S be the schedule returned by our algorithm when executed on the instance I. We define:

$$s(I) = \frac{C_{\max}(S)}{\max(\overline{L}, p_{max}(I))}$$

where $\overline{L} = \Sigma_{i \in \mathcal{J}}^{n} pi/m$ is the average load of a machine and $p_{max}(I) = \max_{i \in \mathcal{J}} p_i$ denotes the largest processing time of a task in I. The value $\max(\overline{L}, p_{max}(I))$ is a lower bound of an optimal makespan for instance I.

To measure the equity of the returned solution we compare the minimum gain obtained in the scheduled returned by MCEDD with two upper bounds of the optimal minimum gain. We define :

$$s'(I) = \frac{\min\limits_{i \in \{1,\dots,N\}} g^i(S)}{\min\{UB1, UB2\}}$$

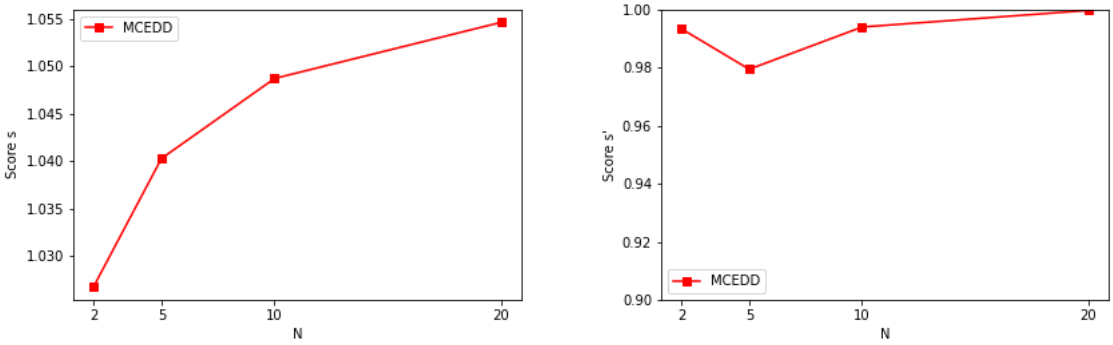
To compute the first upper bound $UB1$ we compute the gain any organization would get if it could schedule all its tasks at the beginning of the schedule and with preemption, i.e. tasks can be divided, this represent an upper bound of the gain this organization can get. We take the minimum of all these gains to obtain an upper bound on the maximum minimum gain. Formally, it is defined as follows:

$$UB1 = \min_{i \in \{1,\dots,N\}} \frac{C^i_{loc} - \max(\overline{L_i(I)}, p_{max}(O_i))}{C^i_{loc}}$$

where $\overline{L_i(I)} = \frac{\sum_{j=1}^{n_i} p_i^j}{m}$ is the average load of a machine if the only tasks in I were the one of O_i; $p_{max}(O_i)$ denotes the largest length of a task owned by O_i. We can note that $\max(\overline{L_i(I)}, p_{max}(O_i))$ is a lower bound of the best makespan that O_i could get. Then, the term $\frac{C^i_{loc} - \max(\overline{L_i(I)}, p_{max}(O_i))}{C^i_{loc}}$ is a higher bound of the gain O_i can get.

The second upper bound $UB2$ is an upper bound of the gain that the organization with the largest makespan can get. A lower bound of the best possible makespan is the average load of a machine $\overline{L}$. This means that at least one organization will get a makespan at least as large as $\overline{L}$. The gain of the organization which has this makespan is then lower than or equal than the gain the organization with the largest local makespan would get if its makespan in the final solution was the average load of a machine, i.e. $UB2 = \frac{C_{loc}^N - \overline{L}}{C_{loc}^N}$ (O_N is assumed to have the largest local makespan). Therefore, it is a higher bound of the minimum gain since at least one organization will get a gain of at most this value.

The local schedules are obtained with the LPT list scheduling. Instances are randomly generated thanks to a realistic generator [Lublin and Feitelson, 2003]. The authors have analyzed data from different sites regarding the workload and programmed a generator creating workloads similar to the ones observed in the data. We set the maximum task length to 50. Tasks are spread among the organizations following a zipf distribution; we set the number of elements of the distribution to N and s to 1.4267 which corresponds to the data observed by Iosup et al. [2006]. We create instances varying three parameters: the number of tasks n, the number of machines m and the number of organizations N. Machines are spread uniformly. We consider 9600 instances.
We will focus on the impact of the number of organizations on the quality of the solution returned by our algorithm on the tested instances.



(a) Variation of score s

(b) Variation of score s'

Figure 2.11: Experimental evaluation of MCEDD.

We see in Figure 2.11(a) that the score s increases with the number of organizations. This is consistent with the idea that the more organizations there are, the more difficult it is to satisfy each one of them. We also observe that the s score is below 1.055. This means that the schedule returned by our algorithm has on average a makespan

lower than 1.055 times a lower bound of optimal global makespan with no rationality constraint.

Figure 2.11(b) shows the variation of score s'. We note that s' is on average above 0.96. This means that the minimum gain in the schedule returned by our algorithm is higher than 0.96 times a higher bound of the optimal minimum gain.

Observation 2.5.2: Explaining the "drop" in the s' ratio

The value of the s' ratio may seem weird for 5 organizations, however we can explain this by looking at the two higher bounds we considered.

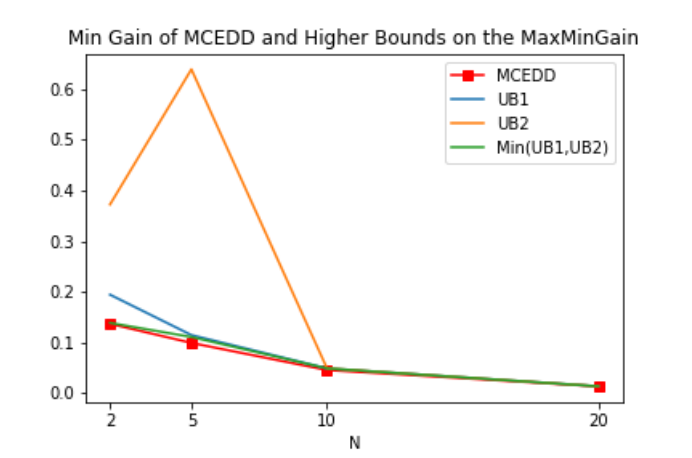


Figure 2.12: Average maximum minimum gain obtained by MCEDD in comparison to the average higher bounds

In this figure, we can see that the combination of the two bounds seem to be a very close higher bound of the optimal solution when $N = 2$, although none is informative enough on its own. However, when $N = 5$, the second higher bound does not seem to bring any information regarding the maximum minimum gain. When $N = 10$ or more, either bound seem to be satisfying with regards to the optimal solution. One potential explanation for the decrease in the s' score is that the bounds are not as informative when $N = 5$ in comparison to over values and not necessarily that MCEDD does not perform as well in such cases.

2.6 Conclusion

In this chapter, we have focused on two problems: MOSP and $(1+\alpha)$-MOSP for which we have studied the necessary tradeoff between efficiency (in term of low makespan) and the (relaxed) rationality constraint. We have also shown the interest of cooperation, that can benefit to all the organizations, and proposed an algorithm which re-

turns schedules $(1+\epsilon)$-approximate for MOSP while the makespans of the organizations are increased by at most a factor $(1+\epsilon)$. We then introduced problem MaxMinGain, for which we have also shown the necessary tradeoff between the minimization of the makespan and the minimization of the minimal gain (excepted if the tasks have all the same length, instances for which there is a polynomial time algorithm optimal for both objectives). We have shown that MaxMinGain is inapproximable in polynomial time if $P \neq NP$, but we have given a heuristic which, in practice, returns good schedules for both the minimization of the makespan and the maximization of the minimal gain.

Note that most results can be adapted if the tasks have released dates. Indeed, the "negative" results are still valid (this concerns complexity proofs, and results showing the necessary tradeoff between the global makespan and either the rationality constraint or the maximization of the minimal gain). The optimal algorithm for MaxMinGain with unit tasks can also be easily adapted. The PTAS of Hall and Shmoys [1989] works with release dates, and thus we can use its adaptation with release dates too: there is also in this case a $(1+\epsilon)$-approximate schedule for MOSP while the makespans of the organizations are increased by at most a factor $(1+\epsilon)$. Likewise, this algorithm can be adapted when machines are not necessary identical but can have a fixed number of different speeds.

Note also that in this chapter we have considered that each organization owns at least one machine, but results also hold if there are organizations with tasks but without any machine (in this case, they do not have "local makespan", and we do not apply the rationality constraint for these organizations).

Chapter 3

Collective schedules: analysis of four aggregation rules

The collective schedules problem consists in computing a schedule of tasks shared between individuals. Tasks may have different duration, and individuals have preferences over the order of the shared tasks. This problem has numerous applications since tasks may model public infrastructure projects, events taking place in a shared room, or work done by co-workers. Our aim is, given the preferred schedules of individuals (voters), to return a consensus schedule. We propose an axiomatic study of the collective schedule problem, by using classic axioms in computational social choice and new axioms that take into account the duration of the tasks. We show that some axioms are incompatible, and we study the axioms fulfilled by four rules: one which has been studied in the seminal paper on collective schedules [Pascual et al., 2018], one which generalizes the Kemeny rule, one which generalizes Spearman's footrule and one which relies on a scheduling approach. From an algorithmic point of view, we show that three of these rules solve NP-hard problems, but that it is possible to solve optimally these problems for small but realistic size instances, and we give an efficient heuristic for large instances. We conclude this chapter with experiments evaluating the quality of the heuristic and the computation time of the four rules.

The results presented in this chapter have been published in [Durand and Pascual, 2022].

3.1 Introduction

In this chapter, we are interested in the scheduling of tasks of interest to different people, who express their preferences regarding the order of execution of the tasks. The aim is to compute a consensus schedule which aggregates the preferences of the individuals, that we will call voters in the sequel.

This problem has numerous applications. For example, public infrastructure projects, such as extending the city subway system into several new metro lines, or simply re-

building the sidewalks of a city, are often phased. Since workforce, machines and yearly budgets are limited, phases have often to be done one after the other. The situation is then as follows: given the different phases of the project (a phase being the construction of a new metro line, or of a new sidewalk), we have to decide in which order to do the phases. Phases may have different duration – some may be very fast while some others may last much longer. In other words, the aim is to find a schedule of the phases, each one being considered as a task of a given duration. Note that tasks may not only represent public infrastructure projects, but they may also model events taking place in a shared room, or work done by co-workers (the schedule to be built being the order in which the events – or the work to be done – must follow each other). In order to get such a schedule, public authorities may take into account the preferences of citizens, or of citizens' representatives, which could be invited to express their preferences.

This problem, introduced by Pascual et al. [2018], takes as input the preferred schedule of each voter (the order in which he or she would like the phases to be done), and returns one collective schedule – taking into account the preferences of the voters and the duration of the tasks. We distinguish two settings. In the first one, each voter would like each task to be scheduled as soon as possible, even if he or she has preferences over the tasks. In other words, if this were possible, all the voters would agree to schedule all the tasks simultaneously as soon as possible. This assumption – the earlier a task is scheduled the better – , will be denoted by *EB* in the sequel. It was assumed by Pascual et al. [2018], and is reasonable in many situations, in particular when tasks are public infrastructure projects. However, it is not relevant in some other situations. Consider for example workers, or members of an association, who share different works that have to be done sequentially, for example because the tasks need the same workers, or the same resource (e.g. room, tool). Each work (task) has a given duration and can imply a different investment of each worker (investment or not of a person, professional travel, staggered working hours, ...). Each worker indicates his or her favorite schedule according to his or her personal constraints and preferences. In this setting, it is natural to try to fit as much as possible to the schedules wanted by the workers – and scheduling a task much earlier than wanted by the voters is not a good thing: assumption EB does not hold here. In this paper, our aim is to compute a socially desired collective schedule, with or without the EB assumption.

Observation 3.1.1: EB setting

In the EB setting, it could make sense to look for a schedule minimizing the sum of the completion times of the tasks. In that way, we make sure that as many tasks as possible are completed at the beginning of the schedule, leaving long tasks that can cause delay to be executed later. In a sense, an "efficient" schedule can be obtained with the SPT (Shortest Processing Time) list algorithm, i.e. by scheduling tasks by increasing processing time. However such a schedule can be unsatisfying for the voters, e.g. if all voters schedule the tasks by decreasing processing time. The collective decision process should take into account both the

preferences and the general idea that short tasks should be somehow favoured when dealing with an EB setting.

This problem generalizes the classical consensus ranking problem, since if all the tasks have the same unit length, the preferred schedules of a voter can be viewed as her preferred ranking of the tasks. Indeed, each (unit) task can be considered as a candidate (or an item), and a schedule can be considered as a ranking of the candidates (items). Computing a collective schedule in this case consists thus in computing a collective ranking, a well-known problem in computational social choice [Brandt et al., 2016].

Related work. Our work is at the boundary between computational social choice [Brandt et al., 2016] and scheduling [Brucker, 2010], two major domains in artificial intelligence and operational research.

As mentioned above, the collective schedule problem generalizes the collective ranking problem, which is an active field in computational social choice (see e.g. [Dwork et al., 2001; Skowron et al., 2017; Celis et al., 2018; Singh and Joachims, 2018; Biega et al., 2018; Geyik et al., 2019; Asudeh et al., 2019; Narasimhan et al., 2020]). In this field, authors often design rules (i.e. algorithms) which return fair rankings, and they often focus on fairness in the beginning of the rankings. If the items to be ranked are recommendations (or restaurants, web pages, etc.) for users, the beginning of the ranking is indeed probably the most important part. Note that this does not hold for our problem since all the planned tasks will be executed – only their order matters. This means that rules designed for the collective ranking problem are not suitable not only because they do not consider duration for the items, but also because they focus on the beginning of the ranking. This also means that the rules we will study can be relevant for consensus ranking problems where the whole ranking is of interest.

As mentioned earlier, the collective schedule problem has been introduced by Pascual et al. [2018] for the EB setting. In this paper, the authors introduced a weighted variant of the Condorcet principle [De Condorcet, 2014], called the PTA Condorcet principle (where PTA stands for "Processing Time Aware"), and they adapted previously known Condorcet consistent rules when tasks have different processing times. They also introduced a new rule, which computes a schedule which minimizes the sum of the tardiness of tasks between the preferred schedules of the voters and the schedule which is returned. They show that the optimization problem solved by this rule is NP-hard but that it can be solved for reasonable instances with a linear program.

Multi agent scheduling problems mainly focus on cases where (usually two) agents own their *own* tasks, that are scheduled on shared machines: the aim is to find a Pareto-optimal and/or a fair schedule of the tasks of the agents, each agent being interested by her own tasks only [Saule and Trystram, 2009; Agnetis et al., 2014]. We also mention the work of Elkind et al. [2022] in which the authors study the assignment of shared unit size tasks to specific unit size time slots. This latter work focus on fairness notions extended from the multi-winner voting problem and does not use scheduling notions.

We conclude this related work section by mentioning similarities between our prob-

lem and the participatory budgeting problem, which is widely studied [Aziz and Shah, 2021]. In the participatory budgeting problem, voters give their preferences over a set of projects of different costs, and the aim is to select a socially desirable set of items of maximum cost B (a given budget). The participatory budgeting problem and the collective schedules problems have common features. They both extend a classical optimization problem when users have preferences: the participatory budgeting problem approach extends the knapsack problem when users have preferences over the items, while the collective schedules problem extends the scheduling problem when users have preferences on the order of the tasks. Moreover, when considering unit items or tasks, both problems extend famous computational social choice problems: the participatory budgeting problem generalizes the multi winner voting problem when items have the same cost, and the collective schedules problem generalizes the collective ranking problem when tasks have the same duration. For both problems, because of the costs/lengths of the items/tasks, classical algorithms used with unit items/tasks may return very bad solutions, and new algorithms are needed. A recent work by Boes et al. [2021] proposes a framework for such collective problems, generalizing, among others, the collective schedules and the participatory budgeting problems. The authors also study how well-known collective decision rules can be extended to fit in these optimization contexts.

Overview of our results.

- In section 3.2, we present four rules to compute consensus schedules. We introduce a new one, that we will denote by PTA Kemeny, and which extends the well-known Kemeny rule [Kemeny, 1959] used to compute consensus rankings in computational social choice. The two next rules come from scheduling theory, and were introduced by Pascual et al. [2018]: they consist in minimizing the sum of the tardiness of tasks in the returned schedule with respect to the voters' schedules (rule ΣT), or in minimizing the sum of the deviation of tasks with respect to the voters' schedules (rule ΣD). Note that the ΣD rule is equal to the Spearman's footrule [Diaconis and Graham, 1976] when the tasks are unitary. The last rule (rule EMD) consists in scheduling tasks by increasing median completion time in the preferences of the voters.

- In section 3.3, we study the axiomatic properties of the above mentioned rules by using classical social choice axioms as well as new axioms taking into account the duration of the tasks. Table 3.1 summarizes our results. We also show incompatibilities between axioms: we show that a rule which is neutral, or which is based on a distance, both does not fulfill the PTA Condorcet consistency property, and can return a schedule with a sum of tardiness as far from the optimal as wanted.

- In Section 3.4, we show that the PTA Kemeny and ΣD rules solve NP-hard problems and we propose a fast heuristic which performs well regarding the ΣD and ΣT rules.

- In Section 3.5, we see that the PTA Kemeny and ΣD rules can be used for small but realistic size instances, and that the heuristic presented in the previous section returns schedules which are very close to the ones returned by ΣD. We also compare the performance of the rules on the sum of tardiness or deviations of the tasks in the returned schedules.

Let us now introduce formally our problem and present the four rules that we will study in the sequel.

3.2 Preliminaries

3.2.1 Definition of the problem and notations.

Let $\mathcal{J} = \{t_1, \ldots, t_n\}$ be a set of n tasks. Each task $t_i \in \mathcal{J}$ has a length (or processing time) p_i. We do not consider idle times between the tasks, and preemption is not allowed: a schedule of the tasks is thus a permutation of the tasks of $\mathcal{J}$. We denote by $X^{\mathcal{J}}$ the set of all possible schedules. We denote by $V = \{v_1, \ldots, v_v\}$ the set of v voters. Each voter $v_k \in V$ expresses her favorite schedule $S_k \in X^{\mathcal{J}}$ of the tasks in $\mathcal{J}$. The preference profile, P, is the set of these schedules: $P = \{S_1, \ldots, S_v\}$.

Example 3.2.1: Preference profile

Let us consider an instance with $n = 3$ and $v = 5$. The set $\mathcal{J} = \{t_a, t_b, t_c\}$ is the set of tasks and we have $p_a = 2, p_b = 4$ and $p_c = 1$. The first 2 voters schedule t_b first, then t_a and finally t_c. A second set of two voters schedule t_a, then t_b, then t_c. The final voter schedules t_c, then t_b and finally t_a. We represent such an instance with a Gantt chart as follows:

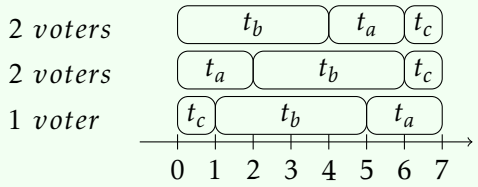


Figure 3.1: Example of a preference profile P

Given a schedule S, we denote by $C_i(S)$ the completion time of task t_i in S. We denote by $d_{i,k}$ the completion time of task t_i in the preferred schedule of voter k (i.e. $d_{i,k} = C_i(S_k)$ – here d stands for "due date" as this completion time can be seen as a due date, as we will see in the sequel. We denote by $t_a \succ_S t_b$ the fact that task t_a is scheduled before task t_b in schedule S. This relation is transitive, therefore, if, in a schedule S, task t_a is scheduled first, then task t_b and finally task t_c, we can describe S as $(t_a \succ_S t_b \succ_S t_c)$. Given a schedule S we will denote by $S_{(t_a \leftrightarrow t_b)}$ the schedule obtained from S by swapping the positions of the tasks t_a and t_b.

An *aggregation rule* is a mapping $r : (X^{\mathcal{J}})^v \to X^{\mathcal{J}}$ that associates a schedule S – the consensus schedule – to any preference profile P. We will focus on four aggregation rules that we introduce now: ΣD, ΣT, *PTA Kemeny* and *EMD*.

3.2.2 Four aggregation rules.

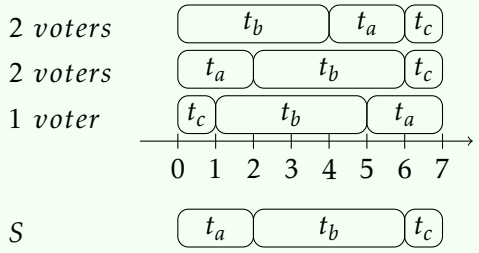
A) The ΣD rule. The ΣD rule is an extension of the Absolute Deviation (D) scheduling metric [Brucker, 2010]. This metric measures the deviation between a schedule S and a set of given due dates for the tasks of the schedule. It sums, over all the tasks, the absolute value of the difference between the completion time of a task t_i in S and its due date. By considering the completion time $d_{i,k}$ of task t_i in the preferred schedule S_k as a due date given by voter v_k for task t_i, we express the deviation $D(S, S_k)$ between schedule S and schedule S_k as $D(S, S_k) = \sum_{t_i \in \mathcal{J}} |C_i(S) - d_{i,k}|$. By summing over all the voters, we obtain a metric $D(S, P)$ measuring the deviation between a schedule S and a preference profile P:

$$D(S, P) = \sum_{S_k \in P} \sum_{t_i \in \mathcal{J}} |C_i(S) - d_{i,k}| \tag{3.1}$$

The ΣD rule returns a schedule S^* minimizing the deviation with the preference profile P: $D(S^*, P) = \min_{S \in X^{\mathcal{J}}} D(S, P)$.

Example 3.2.2: ΣD- computing deviation

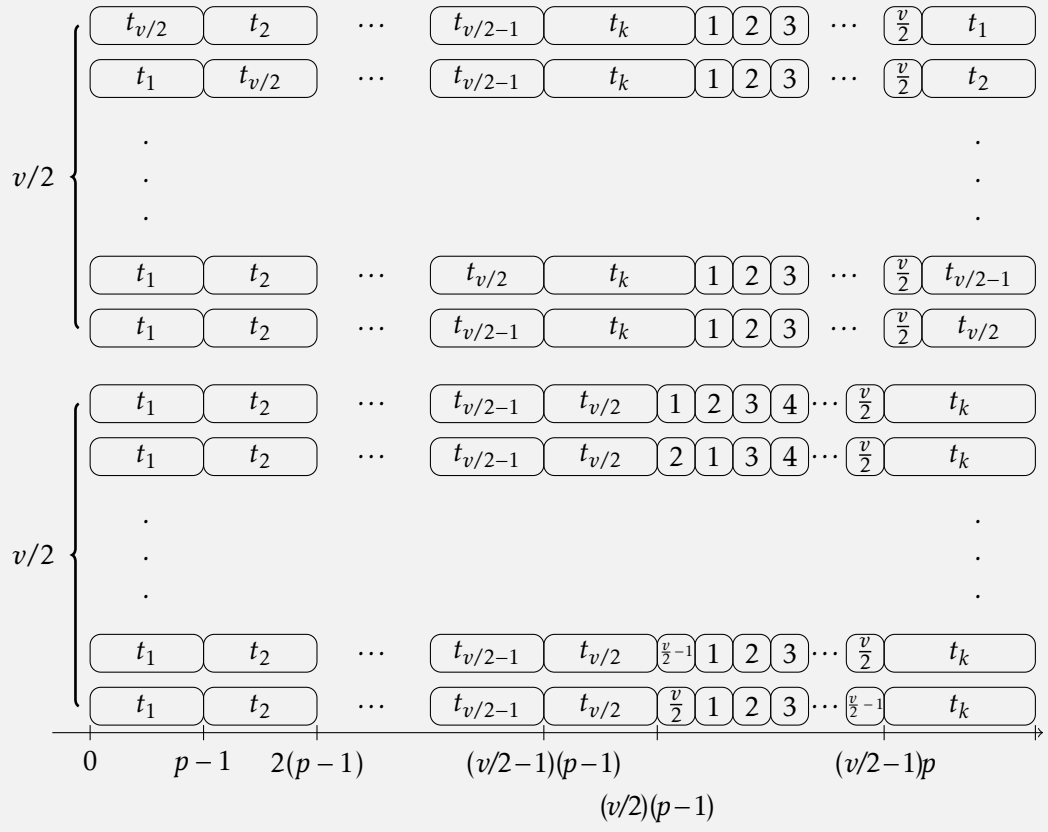
We consider the instance of Example 3.2.1 and the schedule $S = (t_a \succ_S t_b \succ_S t_c)$.



In S, t_a completes at time 2 whereas it completes at time 6 in the schedule preferred by the first two voters, we then count $|2-6|$ as the deviation relative to task t_a. Tasks t_b completes at time 6 in S and at time 4 in the schedule preferred by the first two voters, we count $|6-4|$ for the deviation relative to task t_b. Finally, task t_c completes at time 7 in both schedules, we do not count any deviation. The total deviation between these two schedules is then $|2-6| + |6-4| = 6$. We compute the deviation over all the schedules preferred by the voters and we sum to obtain the total deviation between schedule S and the preference profile P.

Observation 3.2.1: ΣD- Particular property

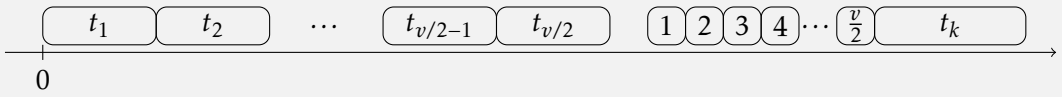
We mention here that contrarily to the other rules that we study in this chapter, the ΣD rule does not "naturally" schedule the tasks without idle time. Indeed, even if all the voters give schedules without idle times, the solution minimizing the sum of deviations may have some, as we see in the following example. There are v voters and $n = v + 1$ tasks. The tasks are of three types. Task t_k has a processing time p. Tasks t_1 to $t_{v/2}$ have a processing time of $p-1$. Tasks 1 to $\frac{v}{2}$ have a processing time of 1. There are v voters split in two groups. The first $v/2$ voters schedule all tasks t_1 to $t_{v/2}$ except one at the beginning of their schedule. The i^{th} schedule all these tasks except t_i. These voters then schedule tasks t_k then the tasks of length 1 in the same order $1, 2, \ldots$ then the task t_i (depending on the voter). The last $v/2$ voters schedule the tasks t_1 to $t_{v/2}$ by increasing index, then one task of processing time one: voter $v/2 + i$ schedule task i at this slot. They then schedule the remaining tasks of processing time one by increasing index and finally t_k. The detailed preference profile can be found below.



We give a few intuitions of why the optimal solution will have an idle time:

- In this instance, the tasks of processing time 1 are all scheduled most of the time in one given time slot. For example task i starts at time $(v/2-1)(p-1)+p$ in $v-i$ preferences. In order to minimize the deviation, a schedule should make sure that these tasks are assigned to their time slots.
- Tasks t_1 to $t_{v/2-1}$ are always scheduled at the same time except for the i^{th} voter who schedule task t_i at the end of the schedule. Because of that, a schedule minimizing deviation should put tasks t_1 to $t_{v/2-1}$ at the beginning of the schedule in that order.
- This leaves the positions of t_k and $t_{v/2}$. Since $t_{v/2}$ is scheduled either after the t tasks at the beginning of the schedule, or in the middle of the t tasks by all voters except one, and since task t_k is scheduled either after the t tasks or at the very end of the schedule the best option regarding deviation is to schedule $t_{v/2}$ after $t_{v/2-1}$.
- If we want each task of processing time 1 to be scheduled at its spot, it is necessary that there is an idle time after $t_{v/2}$, then task $1, 2, \ldots$ and finally t_k which completes one unit of time after the total load of the schedule.

The previous paragraph aims at giving the intuition of why the optimal solution has an idle time, but we can also verify numerically that the optimal solution is indeed:



Looking at tasks 1 to $v/2$, scheduling task i at its time slot in the preferences of the first $v/2$ voters creates a deviation of $i-1+i$ (there are $i-1$ voters for which the task is moved by one unit of time and one voter for which it is moved by i). Scheduling task i one unit of time earlier (which would remove the idle time) increases the deviation by one for $v-i$ voters who scheduled task i in the same slot and reduces it by one for the remaining i voters, this would then be an increase of $\sum_{i=1}^{v/2} v-2i$. On the other hand the deviation of t_k would decrease by v, since it would be one unit of time closer to all the preferences.So for any value of v such that $v < \sum_{i=1}^{v/2} v-2i$, the solution minimizing the total deviation has an idle time.
However, in this chapter we will only consider solutions without idle time. We can consider a lot of contexts in which the time horizon is precisely set and it is impossible to have "holes" in the schedule. It also allows us to have a one to one matching between linear orders and schedules, since given a linear order, there is only one schedule without idle time and respecting the order; whereas there is an infinite number of schedules with idle times respecting the order.

This rule was introduced (but not studied) by Pascual et al. [2018], where the authors observed that, if tasks have unitary lengths, this rule minimizes the Spearman distance, which is defined as $SD(S, S_k) = \sum_{t_i \in \mathcal{J}} |pos_i(S) - pos_i(S_k)|$, where $pos_j(S)$ is the position of item j in ranking S, i.e. the completion time of task j in schedule S if items are unitary tasks.

B) The ΣT rule. This rule, introduced by Pascual et al. [2018], extends the classical Tardiness (T) scheduling criterion [Brucker, 2010]. The tardiness of a task t_i in a schedule S is 0 if task t_i is scheduled in S before its due date, and is equal to its delay with respect to its due date otherwise. As done for ΣD, we consider the completion time of a task t_i in schedule S_k as the due date of voter v_k for task t_i. The sum of the tardiness of the tasks in a schedule S compared to the completion times in a preference profile P is then:

$$T(S, P) = \sum_{S_k \in P} \sum_{t_i \in \mathcal{J}} \max(0, C_i(S) - d_{i,k}) \tag{3.2}$$

The ΣT rule returns a schedule minimizing the sum of tardiness with the preference profile P.

C) The PTA Kemeny rule. We introduce a new rule, the Processing Time Aware Kemeny rule, an extension of the well-known *Kemeny rule* [Kemeny, 1959]. The Kendall tau distance is a famous metric to measure how close two rankings are: it counts the number of pairwise disagreements between two rankings (for each pair of candidates $\{a, b\}$ it counts one if t_a is ranked before t_b in one ranking and not in the other ranking). The Kemeny rule minimizes the sum of the Kendall tau distances to the preference profile, i.e. the voter's preferred rankings.

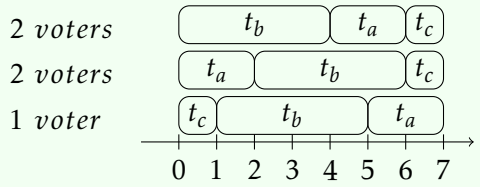
Despite its good axiomatic properties, this rule, which does not take into account the length of the tasks, is not suitable for the collective schedules problem. Consider for example an instance with only two tasks, a short task t_a and a long task t_b. If a majority of voters prefer t_b to be scheduled first, then in the returned schedule it will be the case. However, in EB settings, it may be suitable that t_a is scheduled before t_b since the small task t_a will delay the large one t_b only by a small amount of time, while the contrary is not true.

We therefore propose a weighted extension of the Kemeny rule: the PTA Kemeny rule, which minimizes the sum of weighted Kendall tau distances between a schedule S and the schedules of the preference profile P. The weighted Kendall tau distance between two schedules S and S_k counts the weighted number of pairwise disagreements between two rankings; for each pair of tasks $\{t_a, t_b\}$ such that t_b is scheduled before t_a in S_k and not in S, it counts p_a. This weight measures the delay caused by task t_a on task t_b in S (whereas t_a caused no delay on t_b in S_k). The score measuring the difference between a schedule S and P is:

$$\Delta_{KT}^{PTA}(S, P) = \sum_{S_k \in P} \sum_{\{t_a, t_b\} \in \mathcal{J}^2} p_a \times \mathbb{1}_{t_a \succ_S t_b, t_b \succ_{S_k} t_a} \tag{3.3}$$

Example 3.2.3: Computing the weighted Kendall-Tau score

We consider an instance with three tasks $\{t_a, t_b, t_c\}$ and five voters. We have $p_a = 2, p_b = 4$ and $p_c = 1$. The preference profile is as follows (we indicate in front of each schedule the number of voters for which it is the preferred schedule):



Let us compute the PTA Kendall tau score of schedule $S = (t_b \succ_S t_a \succ_S t_c)$. There is 0 disagreement with the first set of 2 voters. There is 1 disagreement with the second set of 2 voters because the pair $\{t_a, t_b\}$ is inverted. Therefore we count $p_b \times 2 = 8$, since t_b is scheduled before t_a in S. There are 2 disagreements with the last voter, one on the pair $\{t_a, t_c\}$ and one on the pair $\{t_b, t_c\}$. Therefore, we count $p_a = 2$ plus $p_b = 4$. Overall, the score of $S = (t_b \succ_S t_a \succ_S t_c)$ is $8+2+4=14$.

The PTA Kemeny rule returns a schedule minimizing the weighted Kendall Tau distance to the preference profile P.

Observation 3.2.2: Choosing an aggregator

One can note that to define the three rules presented above, we used two steps. The first step consists in defining a way to measure the difference between two schedules: one that we try to evaluate and one expressed by a voter. This is the deviation, the tardiness and the weighted Kendall-tau distance. The second step consists in aggregating these individual differences to obtain a general score over the whole preference profile. For these three rules, we used the sum ($\sum$). However it could be possible to use other aggregators like the minimum (min) or the product ($\prod$). The optimal solution for the (min) would be the schedule minimizing the maximum difference with any voter, in a sense, this ensures that the least satisfied voter is as satisfied as possible.

D) The Earliest Median Date (EMD) rule. The Earliest Median Date (EMD) rule computes the median completion time of each task in the preference profile, and returns a schedule in which the tasks are ordered by increasing median completion time. If two tasks have the same median completion time, it schedules the shortest one first.

Example 3.2.4: EMD rule

In the instance of Example 3.2.3, the vector of the completion times of task t_a is $(6, 6, 2, 2, 7)$, and its median is thus 6. The median of the completion times of task t_b and t_c are 5 and 7, respectively. Therefore, the schedule S returned by the EMD rule is $(t_b \succ_S t_a \succ_S t_c)$.

This rule has two major benefits. Firstly it is easier to understand for a voter than a rule using an optimization criterion, such as the first three rules, and voters are more eager to participate in a process that they understand fully. Secondly, it is fast to compute. Indeed, we only need to compute, for each task, its median completion time, which can be done in $O(nv)$. Indeed, assuming that we get in $O(1)$ the completion time of a task in each preferred schedule, the median of the completion times of a given task is the median of v values, which can be computed in $O(v)$ [Blum et al., 1973]. The tasks are then scheduled by increasing median completion times, which costs $O(n \log n)$. The complexity of the EMD rule is thus $O(nv + n \log n)$, which is in practice $O(nv)$ since we generally have $v \geq \log n$.

Observation 3.2.3: EMD rule - Jackson's rule extension

The EMD rule can be seen as an extension of the Earliest Due Date rule (EDD) when tasks have multiple due dates. The EDD rule is a list algorithm scheduling tasks by non decreasing due dates [Jackson, 1955]. It is known to solve optimally some scheduling problems, like $(1 \| L_{\max})$ in which we want to minimize the maximum difference between a task's completion time and its due date. It is also commonly used as a part of other scheduling algorithm to solve more complex problems [Pinedo, 2012].

3.2.3 Resoluteness.

Note that some of these rules return a schedule minimizing an optimization function, and that it is possible that several optimal schedules exist. In computational social choice, rules may be either resolute or irresolute. A rule is *resolute* if it always returns one single solution, and it is *irresolute* if it returns a set of solutions. Thus, rules optimizing an objective function may either be irresolute, and return all the optimal solutions, or they can be resolute and use a tie-breaking mechanism which allows to determine a unique optimal solution for each instance.

Irresolute rules have the advantage that a decision maker can choose among the optimal solutions, the one that he or she prefers. However, the set of optimal solutions can be large, and sometimes even exponential, making it difficult to compute in practice. Furthermore, in real situations, there is not always a decision maker which makes choices, and an algorithm has to return a unique solution: in this case, the rule must be resolute and needs to use a tie breaking mechanism that allows to decide between the optimal solutions.

In this chapter, unless otherwise stated, we consider that each rule is resolute and returns thus always a unique solution. However, since a good tie breaking mechanism is usually dependent on the context, we will not describe it. Instead, we will study the properties of the set of optimal solutions and see if using a tie breaking mechanism impacts the axiomatic properties of the rule – as we will see, most of the time, this will not be the case.

3.3 Axiomatic properties

In this section, we study from an axiomatic point of view the four rules that we have introduced earlier. We use existing axioms and properties (such as the PTA Condorcet consistency property, defined by Pascual et al. [2018]). We extend standard existing axioms (such as neutrality) to our context, and we introduce new axioms (such as the length reduction monotonicity). Table 3.1 summarizes the axioms fulfilled by each of the four rules. We also outline some incompatibilities between these axioms and properties.

3.3.1 Neutrality and PTA neutrality.

The neutrality axiom is a classical requirement of a social choice rule. A rule is *neutral* if it does not discriminate apriori between different candidates. Note that this axiom can be fulfilled only by irresolute rules, since a resolute rule should return only one solution, even when there are only two equal length tasks t_a and t_b, and two voters: one voter who prefers that t_a is before t_b, while the other voter prefers that t_b is before t_a (the same remark holds for consensus rankings instead of consensus schedules). Therefore, in this subsection we will consider that our the rules ΣD, ΣT, and PTA Kemeny return all the optimal solutions of the function they optimize. Throughout this subsection, we will call $P_{(t_a \leftrightarrow t_b)}$ the preference profile obtained from P by switching the positions of two tasks t_a and t_b in the preferences.

Definition 3.3.1: Neutrality

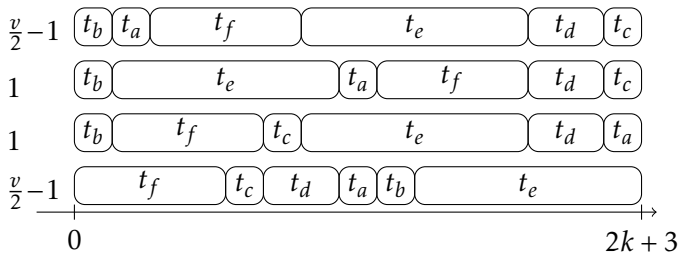
Let r be an irresolute aggregation rule, P a preference profile, and $\mathcal{S}^*$ the set of solutions returned by r when applied on P. Let $\mathcal{S}^*_{(t_a \leftrightarrow t_b)}$ the set of solutions returned by r on $P_{(t_a \leftrightarrow t_b)}$. The rule r is *neutral* if and only if for each solution S in $\mathcal{S}^*$, $S_{(t_a \leftrightarrow t_b)}$ is in $\mathcal{S}^*_{(t_a \leftrightarrow t_b)}$.

Proposition 3.3.1: Neutrality - ΣD

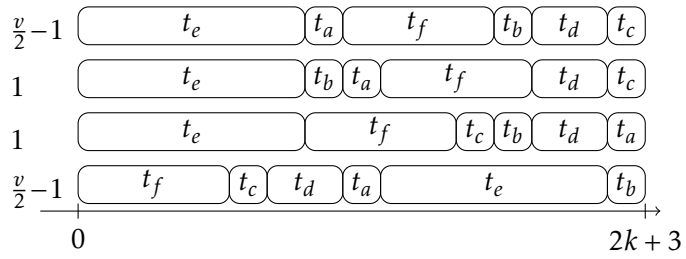
The ΣD rule is not neutral even if it does not apply any tie-breaking mechanism.

Proof. Let us consider an instance with $n = 6$ tasks $\{t_a, t_b, t_c, t_d, t_e, t_f\}$, we have $p_a = p_b = p_c = 1, p_d = 2, p_e = k$ and $p_f = k - 2$, with k a positive integer. The instance has a high

even number of v voters having the following preferences:



For $k = 20$ and $v = 400$, the ΣD rule returns the schedule $S = (t_b \succ_S t_f \succ_S t_a \succ_S t_e \succ_S t_d \succ_S t_c)$ since it is the only one minimizing the absolute deviation with the profile P. If we consider the profile $P_{(t_b \leftrightarrow t_e)}$ in which the positions of t_b and t_e are swapped, we have:



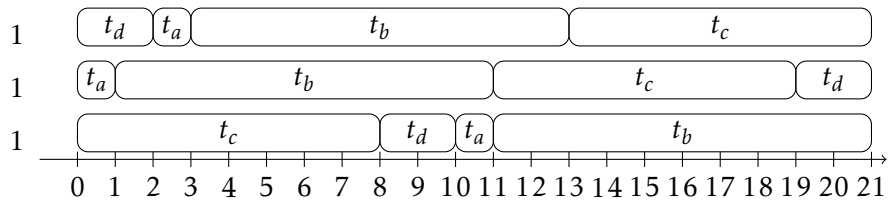
For profile $P_{(t_b \leftrightarrow t_e)}$, the only optimal schedule is $S' = (t_e \succ_{S'} t_a \succ_{S'} t_f \succ_{S'} t_b \succ_{S'} t_d \succ_{S'} t_c)$. If the ΣD rule was neutral, S and S' would be similar but the position of t_b and t_e would be swapped. However, the positions of t_a and t_f are also modified, meaning that the ΣD rule is not neutral. □

As we will see later, the ΣT and the PTA Kemeny rules do not fulfill neutrality (this will be corollaries of Propositions 3.3.7 and 3.3.9).

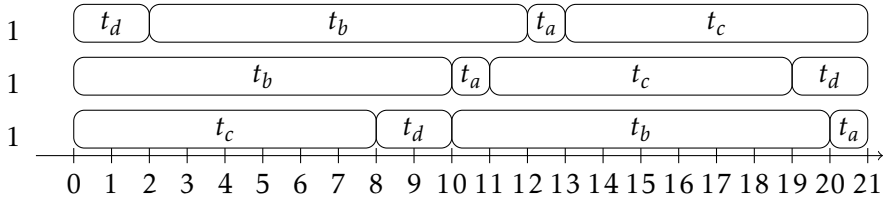
Proposition 3.3.2: Neutrality - EMD

The EMD rule does not fulfill neutrality.

Proof. Let us consider an instance with $n = 4$ tasks $\{t_a, t_b, t_c, t_d\}$ and $v = 3$ voters. We have $p_a = 1$, $p_b = 10$, $p_c = 8$ and $p_d = 2$. The preferences are as follows:



For such a profile the median completion times are as follows: $m_a(P) = 3, m_b(P) = 13, m_c(P) = 19$ and $m_d(P) = 10$ where $m_i(P)$ is the median completion time of task t_i in the preference profile P. The schedule returned by the EMD rule is then $S = (t_a \prec_S t_d \prec_S t_b \prec_S t_c$. Let us now study the preference profile $P_{(t_a \leftrightarrow t_b)}$:



For the profile $P_{(t_a \leftrightarrow t_b)}$, the median completion times are as follows: $m_a(P_{(t_a \leftrightarrow t_b)}) = 13, m_b(P_{(t_a \leftrightarrow t_b)}) = 12, m_c(P_{(t_a \leftrightarrow t_b)}) = 19$ and $m_d(P_{(t_a \leftrightarrow t_b)}) = 10$, therefore, the schedule returned by the EMD rule is $S' = (t_d \succ_{S'} t_b \succ_{S'} t_a \succ_{S'} t_c)$. If the EMD rule was neutral, S and S' would be similar but the positions of t_a and t_b would be swapped. Since here the position of t_d also changes, EMD is not neutral. □

Treating all tasks equally despite the fact that they can have very different length does not seem natural. However, rules should still have some guarantee of equal treatment. We introduce the *PTA neutrality* axiom, which ensures that two tasks of equal length are considered in the same way.

Definition 3.3.2: PTA neutrality

Let r be an irresolute aggregation rule, P a preference profile, and $\mathcal{S}^*$ the set of solutions returned by r when applied on P. Let $\mathcal{S}^*_{(t_a \leftrightarrow t_b)}$ the set of solutions returned by r on $P_{(t_a \leftrightarrow t_b)}$. The rule r is *PTA neutral* if and only if, for any two tasks t_a and t_b such that $p_a = p_b$, for each solution S in $\mathcal{S}^*$, $S_{(t_a \leftrightarrow t_b)}$ is in $\mathcal{S}^*_{(t_a \leftrightarrow t_b)}$.

The PTA neutrality axiom relaxes the concept of neutrality for the cases in which tasks (candidates) have lengths (weights). This axiom ensures that two candidates with the same characteristics are treated equally. When all the tasks have the same length, the PTA neutrality axiom is equal to the neutrality axiom.

Proposition 3.3.3: PTA-Neutrality - All rules

If they do not apply any tie-breaking mechanism, the PTA Kemeny, ΣD, ΣT and EMD rules are PTA neutral.

Proof. Let $P = \{S_1, \ldots, S_v\}$ be a preference profile and t_i and t_j be two tasks such that $p_i = p_j$. For any schedule S_k expressed by a voter v_k, we know that in $S_{k(t_i \leftrightarrow t_j)}$, we have $C_i(S_{k(t_i \leftrightarrow t_j)}) = C_j(S_k)$ and $C_j(S_{k(t_i \leftrightarrow t_j)}) = C_i(S_k)$. The completion times of any other task

than t_i and t_j is the same in S_k and in $S_{k(t_i \leftrightarrow t_j)}$ since no other task has been moved and since $p_i = p_j$.

For each possible consensus schedule S for P, we can note that $S_{(t_i \leftrightarrow t_j)}$ has the same deviation (resp. tardiness) for $P_{(t_i \leftrightarrow t_j)}$ than S for P. This implies that if a schedule S is optimal for ΣD (resp. ΣT) for a profile P, then the schedule $S_{(t_i \leftrightarrow t_j)}$ is optimal for ΣD (resp. ΣT) for he profile $P_{(t_i \leftrightarrow t_j)}$: the PTA neutrality holds for ΣD and ΣT.

The PTA Kemeny rule returns a ranking minimizing the weighted sum of pairwise disagreements with the profile. By swapping the positions of t_i and t_j in both the profile P and a schedule S, we do not change the disagreements on pairs of tasks that do not contain t_i nor t_j. The pair $\{t_i, t_j\}$ is permuted in both $S_{(t_i \leftrightarrow t_j)}$ and $P_{(t_i \leftrightarrow t_j)}$ leading to the same number of disagreements. Indeed, whenever there was a disagreement between S and a preference $S_{k(t_i \leftrightarrow t_j)}$ in P on a pair $\{t_i, t_x\}$ then there will be a disagreement on $\{t_j, t_x\}$ between $S_{(t_i \leftrightarrow t_j)}$ and $S_{k(t_i \leftrightarrow t_j)}$. Similarly if S and a preference S_k of P agree on the pair $\{t_i, t_x\}$, then $S_{(t_i \leftrightarrow t_j)}$ and the preference $S_{k(t_i \leftrightarrow t_j)}$ in $P_{(t_i \leftrightarrow t_j)}$ will agree on $\{t_j, t_x\}$. Since the lengths of t_i and t_j are the same, the weights on a disagreement between S and P will be the same than a disagreement between $S_{(t_i \leftrightarrow t_j)}$ and $P_{(t_i \leftrightarrow t_j)}$, meaning that the overall sum of weighted disagreements between S and P is the same than between $S_{(t_i \leftrightarrow t_j)}$ and $P_{(t_i \leftrightarrow t_j)}$. Since this applies to every schedule S, if S is optimal for the profile P, then $S_{(t_i \leftrightarrow t_j)}$ is optimal for $P_{(t_i \leftrightarrow t_j)}$. Hence the PTA Kemeny rule is PTA neutral.

As seen above, for each voter v_k, we have $C_i(S_{k(t_i \leftrightarrow t_j)}) = C_j(S_k)$ and $C_j(S_{k(t_i \leftrightarrow t_j)}) = C_i(S_k)$, and for any other task t_l we have $C_l(S_k) = C_l(S_{k(t_i \leftrightarrow t_j)})$. Therefore, the set of completion times of any other task is the same in P and in $P_{(t_i \leftrightarrow t_j)}$, and the median completion time of a task is the same in P and in $P_{(t_i \leftrightarrow t_j)}$ except for t_i and t_j which swapped their median and therefore swapped their position in the order of the tasks when sorted by increasing median time. Therefore, the EMD rule will return the same schedules when applied on P and on $P_{(t_i \leftrightarrow t_j)}$ except that t_i and t_j are swapped: it is PTA neutral. □

Note that the neutrality axiom is incompatible with the resoluteness axiom [Brandt et al., 2016], and this is the same for PTA neutrality. That means that any rule which always returns only a single solution cannot be neutral: for our rules, once we use a tie-breaking mechanism, we have to give up PTA-neutrality. However, if we focus on the set of optimal solutions, they all fulfill the PTA neutrality axiom.

3.3.2 Distance.

Some aggregation rules are based on the minimization of a metric. By metric, we mean a mapping between a pair of elements, in our case two schedules, and a value. This is the case of ΣT (resp. ΣD) where the value associated to the pair (S, S_k), where S_k is the preferred schedule of a voter v_k and S is a given schedule, is the sum of the tardiness (resp. deviations) of the tasks in S with respect to their completion time in S_k. This is also the case of the PTA Kemeny rule where the value associated to the pair (S, S'_k) is the weighted number of pairwise disagreement between S and S_k.

Most of these rules sum these values over the whole preference profile to evaluate the difference between a schedule and a preference profile. This is the the case of the ΣT, ΣD, and PTA Kemeny rule. If a rule minimizes the sum of this metric over the voters and if the metric is a distance (i.e. it satisfies non-negativity, identity of indiscernible, triangle inequality and symmetry), we say that the aggregation rule is "based on a distance". The fact that a metric is a distance leads to interesting properties Brandt et al. [2016].

Observation 3.3.1: Distance-based rules

In this chapter, we only focus on the sum but these desirable axioms are also fulfilled by rule using other aggregators. In [Elkind et al., 2010, 2011], the authors study a more general framework in which it is possible to use any norm and not just the sum (which is the l_1 norm). They show that some properties of the distance and the norm imply certain axiomatic properties for the rule. They show for example that any rule using a (pseudo)distance and any norm as an aggregator fulfill reinforcement.

Proposition 3.3.4: Distance - ΣD

The absolute deviation metric is a distance.

Proof. To be a distance, a metric m must fulfill four properties:

(1) Non negativity: $m(S,S') \geq 0, \forall S, S' \in X^{\mathcal{J}}$

(2) Identity of indiscernibles: $m(S,S') = 0$ iff $S = S'$

(3) Symmetry: $m(S,S') = m(S',S), \forall S, S' \in X^{\mathcal{J}}$

(4) Triangle inequality: $m(S,S') \leq m(S,z) + m(z,S'), \forall S, S', z \in X^{\mathcal{J}}$

The non negativity (1) property is direct since we sum absolute values, which are always positive.
We prove the identity of indiscernibles (2) by noting that two schedules S, S' are identical iff all the tasks complete at the exact same time in both schedules. Therefore, if S and S' are identical, then there is no difference between the completion times of a task in the two schedules, and the deviation is thus null. Otherwise, at least one task completes at a different time in the two schedules, leading to a non-null difference, and a positive overall absolute deviation between the two schedules.
The symmetry (3) property is a direct consequence of the evenness of the absolute value function. By definition, $D(S,S') = \sum_{t_i \in \mathcal{J}} |C_i(S) - C_i(S')|$ and $D(S',S) = \sum_{t_i \in \mathcal{J}} |C_i(S') - C_i(S)|$. By noting that: $C_i(S) - C_i(S') = -(C_i(S') - C_i(S))$ and since $|a| = |-a|, \forall a \in \mathbb{R}$, we have $D(S,S') = D(S',S)$.
Finally, we prove the triangle inequality (4) thanks to the subadditivity property of the absolute value function. We consider the absolute deviation between two schedules

S and S': $D(S,S')$. Let z be a third schedule. By definition: $D(S,S') = \sum_{t_i \in \mathcal{J}} |C_i(S) - C_i(S')| = \sum_{t_i \in \mathcal{J}} |C_i(S) - C_i(z) + C_i(z) - C_i(S')|$. By subadditivity of the absolute value, we have:

$$D(S,S') \leq \sum_{t_i \in \mathcal{J}} (|C_i(S) - C_i(z)| + |C_i(z) - C_i(S')|)$$

$$D(S,S') \leq D(S,z) + D(z,S')$$

□

As we will see in the sequel (Propositions 3.3.8 and 3.3.10), the fact that the D metric is a distance implies that the ΣD rule is not PTA Condorcet consistent, and that it can return solutions with a sum of tardiness arbitrarily larger than the optimal sum of tardiness. Before seeing this, let us start by recalling what is the PTA Condorcet consistency property, introduced in [Pascual et al., 2018].

3.3.3 PTA Condorcet consistency.

This axiom, particularly meaningful in EB settings, states that a task t_a should be scheduled before task t_b if a fraction of at least $p_a/(p_a + p_b)$ of the voters schedule t_a before t_b. The idea behind this axiom is that the longer a task is, the more it should be supported in order to be scheduled early.

Definition 3.3.3: PTA Condorcet consistency [Pascual et al., 2018]

A schedule S is PTA Condorcet consistent with a preference profile P if, for any two tasks t_a and t_b, it holds that t_a is scheduled before t_b in S whenever at least $\frac{p_a}{p_a+p_b} \cdot v$ voters put t_a before t_b in their preferred schedule. A scheduling rule satisfies the PTA Condorcet principle if for each preference profile it returns only the PTA Condorcet consistent schedule, whenever such a schedule exists.

Note that if all the tasks have the same length, the PTA Condorcet consistency is equal to the well-known Condorcet consistency [De Condorcet, 2014].

Proposition 3.3.5: PTA Condorcet consistency - PTA Kemeny

The PTA Kemeny rule is PTA Condorcet consistent.

Proof. Let S be a schedule returned by the PTA Kemeny rule. For the sake of contradiction, let us suppose that, in S, there is a pair of tasks t_a and t_b such that t_a is scheduled before t_b whereas more than $\frac{p_b}{p_a+p_b} \times v$ voters scheduled t_b before t_a and that a PTA Condorcet schedule exists.
We study two cases. Firstly, consider the tasks t_a and t_b are scheduled consecutively in S. In that case, we call $S_{(t_a \leftrightarrow t_b)}$ the schedule obtained from S in which we swap the position of t_a and t_b. Since both schedules are identical except for the inversion of the

pair $\{t_a, t_b\}$ their weighted Kendall tau scores vary only by the number of disagreements on this pair.

- We have assumed that the number $v_{b>a}$ of voters scheduling t_b before t_a is larger than $\frac{p_b}{p_a+p_b} \times v$. Since in S, t_a is scheduled before t_b, the weighted disagreement of the voters on pair $\{t_a, t_b\}$ in S is larger than $\frac{p_b}{p_a+p_b} \times v \times p_a$.

- In $S_{(t_a \leftrightarrow t_b)}$, t_b is scheduled before t_a. Since $v_{b>a} > \frac{p_b}{p_a+p_b} \times v$, we know that the number $v_{a>b}$ of voters scheduling t_a before t_b is smaller than $\frac{p_a}{p_a+p_b} \times v$. Therefore, the weighted disagreement on pair $\{t_a, t_b\}$ is smaller than $\frac{p_a}{p_a+p_b} \times v \times p_b$.

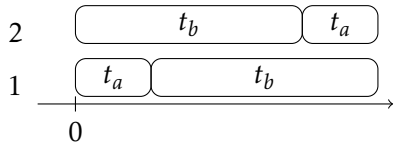
Thus the score of $S_{(t_a \leftrightarrow t_b)}$ is smaller than the one of S: S is not optimal for the PTA Kemeny rule, a contradiction.

Secondly, let us consider that t_a and t_b are not consecutive in S, and let t_c be the task which follows t_a in S. In S, it is not possible to swap two consecutive tasks to reduce the weighted Kendall tau score, otherwise the schedule could not be returned by the PTA Kemeny rule. Thus, by denoting by $S_{(t_a \leftrightarrow t_c)}$ the schedule S in which we exchange the order of tasks t_a and t_c, we get that $\Delta_{KT}^{PTA}(S_{(t_a \leftrightarrow t_c)}, P) - \Delta_{KT}^{PTA}(S, P) \geq 0$. This implies that $v_{t_a>t_c} \times p_c - v_{t_c>t_a} \times p_a \geq 0$ and $v_{t_a>t_c} \times \frac{p_c}{p_a+p_c} - v_{t_c>t_a} \times \frac{p_a}{p_a+p_c} \geq 0$, where $v_{t_c>t_a} = v - v_{t_a>t_c}$ is the number of voters who schedule t_c before t_a in their preferred schedule. Therefore, $v_{t_a>t_c} \geq v \times \frac{p_a}{p_a+p_c}$. Therefore, task t_a is scheduled before t_c in any PTA Condorcet consistent schedule. By using the same argument, we find that task t_c is scheduled before task t_d, which follows t_c in S, and that t_c has to be scheduled before t_d in any PTA Condorcet consistent schedule, and so forth until we meet task t_b. This set of tasks forms a cycle since t_a has to be scheduled before t_c in a PTA Condorcet consistent schedule, t_c has to be scheduled before t_d in a PTA Condorcet consistent schedule, ..., until we meet t_b. Moreover t_b has to be scheduled before t_a in a PTA Condorcet consistent schedule since more than $\frac{p_b}{p_a+p_b} \times v$ voters scheduled t_b before t_a. The existence of this cycle means that no PTA Condorcet consistent schedule exists for the profile, a contradiction. □

Proposition 3.3.6: PTA Condorcet consistency - EMD

The EMD rule is not PTA-Condorcet consistent.

Proof. Let us consider an instance with $n = 2$ tasks $\{t_a, t_b\}$ and $v = 3$ voters. We have $p_a = 1$ and $p_b = 3$. The preferences are as follows:



In such a profile, the median completion time of t_b is smaller than the median completion time of t_a: the EMD rule returns the schedule $S = (t_a \prec_S t_b)$. However t_a is scheduled in 1/3 of the preferences before t_b, and 1/3 is larger than $p_a/(p_a + p_b) = 1/4$: in a PTA-Condorcet consistent schedule t_a is scheduled before t_b, and the EMD rule is thus not PTA-Condorcet consistent. □

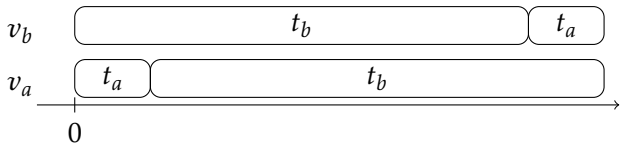
3.3.4 Incompatibilities between axioms and properties.

One can wonder if the PTA Kemeny rule (without breaking-tie rule) is the only rule which is PTA Condorcet consistent, neutral and which fulfills reinforcement, just like the Kemeny rule is the only Condorcet consistent neutral rule fulfilling reinforcement [Young and Levenglick, 1978]. We will show that it is not true, since PTA Kemeny does not fulfill neutrality. We even show a more general statement : no neutral rule can be PTA Condorcet consistent. This answers an open question of Pascual et al. [2018] where the author conjectured "that rules satisfying neutrality and reinforcement fail the PTA Condorcet principle" and said that "it is an interesting open question whether such an impossibility theorem holds".

Proposition 3.3.7: Incompatibility - PTA Condorcet / Neutrality

No neutral rule can be PTA Condorcet consistent.

Proof. Let us consider an instance I with an odd number of voters $v \geq 3$, two tasks t_a and t_b, such that $p_a = 1$ and $p_b = v$, and a preference profile as follows: $v_a = \frac{v-1}{2}$ voters prefer schedule $t_a \succ t_b$ (this schedule will be denoted by A), and $v_b = \frac{v+1}{2}$ voters prefer schedule $t_b \succ t_a$ (schedule denoted by B).



By contradiction, let us suppose that r is a rule which is both neutral and PTA Condorcet consistent. Since r is PTA Condorcet consistent, it will necessarily return the only PTA Condorcet consistent schedule when applied on instance I: A (indeed at least $\frac{p_a}{p_a+p_b} \times v = \frac{v}{v+1} \leq 1$ voter prefer to schedule t_a before t_b).

Let $P_{(t_a \leftrightarrow t_b)}$ be the preference profile obtained from P in which the positions of t_a and t_b are swapped in all the voters' preferences. Since r is neutral, it necessarily returns schedule A in which we have inverted t_a and t_b, i.e. schedule B. However, this schedule is not PTA Condorcet consistent, whereas there exists a PTA Condorcet schedule. Indeed, schedule A is a PTA Condorcet consistent schedule for $P_{(t_a \leftrightarrow t_b)}$ since at least $\frac{p_a}{p_a+p_b} \times v = \frac{v}{v+1} \leq 1 \leq v_a$ voters prefer to schedule t_a before t_b, while $\frac{p_b}{p_a+p_b} \times v = \frac{v^2}{(v+1)}$ is larger than v_b for all values of $v \geq 3$. □

This proposition implies that the PTA Kemeny rule is not neutral, even if no tie-breaking mechanism is used, since it is PTA Condorcet consistent.

Aggregation rules based on distance metrics have several good axiomatic properties [Elkind et al., 2010, 2011; Brandt et al., 2016]. However, they cannot be PTA Condorcet consistent, as shown by the following proposition.

Proposition 3.3.8: Incompatibility - PTA Condorcet / Distance

Any resolute aggregation rule returning a schedule minimizing a distance with the preference profile violates the PTA Condorcet consistency property. This result holds for any tie-breaking mechanism.

Proof. Let us consider an instance I with two tasks t_a and t_b, such that $p_a = 1$ and $p_b = v$, an odd number of voters $v \geq 3$, and a preference profile as follows: $v_a = \lfloor \frac{v-1}{2} \rfloor$ voters prefer schedule $t_a \succ t_b$ (this schedule will be denoted by A), and $v_b = \lceil \frac{v+1}{2} \rceil$ voters prefer schedule $t_b \succ t_a$ (schedule denoted by B). A distance relation t_d fulfills symmetry: for each pair of schedules S and S', $d(S,S') = d(S',S)$. Therefore, for our instance, by symmetry we have: $d(A,B) = d(B,A)$. Since $v_b > v_a$, any aggregation rule r based on minimizing a distance with the profile will return B only. However, the only Condorcet consistent schedule is A. Since rule r returns B, r is not PTA Condorcet consistent. □

Let us now show that neutrality and distance minimization can lead to very inefficient solutions for tardiness minimization.

Proposition 3.3.9: Inapproximability for ΣT of neutral rules

For any $\alpha \geq 1$, there is no neutral aggregation rule returning a set of solutions $\mathcal{S}$ such that all the solutions in $\mathcal{S}$ are α-approximate for ΣT.

Proof. Let us consider an instance I_k with two tasks t_a and t_b, such that $p_a = 1$ and $p_b = k$, an odd number of voters $v \geq 3$, and a preference profile as follows: $v_a = \lfloor \frac{v-1}{2} \rfloor$ voters prefer schedule $t_a \succ t_b$ (this schedule will be denoted by A), and $v_b = \lceil \frac{v+1}{2} \rceil$ voters prefer schedule $t_b \succ t_a$ (schedule denoted by B). We define profile $P_{(t_a \leftrightarrow t_b)}$ as the profile P in which tasks t_a and t_b are swapped in the preferences. Any neutral rule which returns A (resp. B) when applied on P will return B (resp. A) when applied on $P_{(t_a \leftrightarrow t_b)}$. A neutral rule could also return $\{A, B\}$

For profile P, schedule A has a sum of tardiness of $\lfloor \frac{v-1}{2} \rfloor$, since task t_b is delayed by 1 in comparison to schedule B. Schedule B has a sum of tardiness of $\lceil \frac{v+1}{2} \rceil \times k$ since task t_a is delayed by k in comparison to schedule A.

Likewise, in profile $P_{(t_a \leftrightarrow t_b)}$, schedule A has a sum of tardiness of $\lceil \frac{v+1}{2} \rceil$, and schedule B has a sum of tardiness of $\lfloor \frac{v-1}{2} \rfloor \cdot k$.

For both profiles P and $P_{(t_a \leftrightarrow t_b)}$, schedule B has a total sum of tardiness k times higher than the optimal (schedule A), which can be arbitrarily far from the optimal. Since a neutral rule r returns B either for profile P or for profile $P_{(t_a \leftrightarrow t_b)}$, or both, and

since k can be as big as we want, the sum of tardiness of at least one schedule returned by r can be arbitrarily far from the optimal. □

Since the ΣT rule, without tie-breaking mechanism, returns only optimal solutions for the tardiness minimization, this implies that the ΣT rule is not neutral. Let us now show that aggregation rules based on a distance minimization, as ΣD, can return very bad solutions for ΣT.

Proposition 3.3.10: Inapproximability for ΣT of distance-based rules

For any $\alpha \geq 1$, there is no aggregation rule based on a distance minimization and always returning at least one α-approximate solution for ΣT.

Proof. Let us consider an instance I_k with two tasks t_a and t_b, such that $p_a = 1$ and $p_b = k$, an odd number of voters $v \geq 3$, and a preference profile as follows: $v_a = \lfloor \frac{v-1}{2} \rfloor$ voters prefer schedule $t_a \succ t_b$ (this schedule will be denoted by A), and $v_b = \lceil \frac{v+1}{2} \rceil$ voters prefer schedule $t_b \succ t_a$ (schedule denoted by B). Any distance t_d is symmetric, therefore $d(A,B) = d(B,A)$. Any rule returning the schedule minimizing the distance with the profile will return A (resp. B) if A (resp. B) is more present than B (resp. A) in the profile. Since a majority of voter prefer B, any rule based on a distance minimization returns B. For profile P, A has a total sum of tardiness of $\lceil \frac{v+1}{2} \rceil \times 1$ since task t_b is delayed by 1 in comparison to schedule B. On the other hand, B has a total sum of tardiness of $\lfloor \frac{v-1}{2} \rfloor \times k$, since task t_a is delayed by k in comparison to schedule A. Since k can be as high as we want, the sum of tardiness in schedule B can be arbitrarily far from the optimal sum of tardiness. □

3.3.5 Length reduction monotonicity.

Let us now introduce a new axiomatic property, which is close to the discount monotonicity axiom [Talmon and Faliszewski, 2019] for the participatory budgeting problem. A rule r satisfies the *discount monotonicity* axiom if a project cannot be penalised because it is cheaper (i.e. if a project is selected by rule r then it should also be selected by this rule if its price decreases, all else being equal). We propose a new axiom, that we call *length reduction monotonicity*, and which states that the starting time of a task in a schedule cannot be delayed if its length decreases (all else being equal). This axiom is particularly meaningful in EB settings, where all the voters would like all the tasks to be scheduled as soon as possible.

Definition 3.3.4: Length Reduction Monotonicity

Let S be the schedule returned by a resolute rule r on instance I. Assume that we decrease the length of a task t_i in I, all else being equal. Let S' be the schedule returned by r on this new instance. Rule r fulfills *length reduction monotonicity* if task t_i does not start later in S' than in S.

Proposition 3.3.11: Length Reduction Monotonicity - EMD

The EMD rule fulfills length reduction monotonicity for any tie-breaking mechanism.

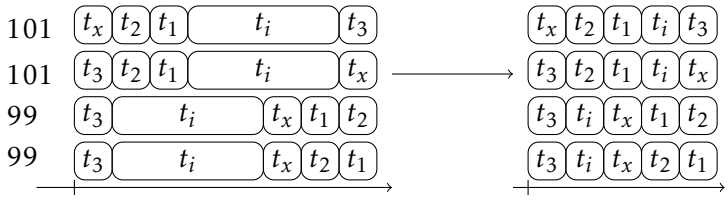
Proof. Let P be a preference profile and S be the schedule returned by the EMD rule on P. Let P' be the preference profile obtained from P in which a task t_i has its length reduced by $x > 0$, everything else being equal. For each voter v_k, the completion time of t_i in P' is equal to its completion time in P minus x. Its median completion time is then reduced by x in P' with respect to its median completion time in P. For any other task t_j – whose length does not change –, its median is reduced by at most x (since in each preference its completion time is reduced by x if t_j is scheduled after t_i, and is not reduced otherwise). The median completion time of task t_i decreases by x whereas the median completion times of the other tasks decreases by at most x, so EMD will not schedule t_i later in P than in P': the EMD rule fulfills the Length Reduction Monotonicity property. □

Since the EMD rule fulfills the length reduction monotonicity property, it seems particularly indicated for EB settings. Unlike the EMD rule, let us now see that the ΣD rule does not fulfill the length reduction monotonicity property.

Proposition 3.3.12: Length Reduction Monotonicity - ΣD

The ΣD rule does not fulfill length reduction monotonicity for any tie-breaking mechanism.

Proof. Let us consider an instance with 5 tasks $\{t_1, t_2, t_3, t_x, t_i\}$ with $p_1 = p_2 = p_3 = p_x = 1$ and $p_i = 10$. The preferences of the 400 voters are as follows:



For the profile on the left, the only schedule S minimizing the absolute deviation is : $S = (t_3 \succ_S t_i \succ_S t_x \succ_S t_2 \succ_S t_1)$. For the profile on the right, the only schedule S' minimizing the absolute deviation is such that: $S' = (t_3 \succ_{S'} t_2 \succ_{S'} t_x \succ_{S'} t_i \succ_{S'} t_1)$. Task t_i has a reduced length but it starts later in S' than in S: ΣD does not fulfill length reduction monotonicity. □

It is not very surprising that the ΣD rule does not fulfill this axiom. Indeed, the LRM axiom is particularly meaningful in EB settings, whereas ΣD aims at returning a schedule that fits as much as possible as the completion times given by the voters, and

is not particularly well adapted for EB settings. Determining whether the rules ΣT and PTA Kemeny fulfill the LRM axiom is an open problem.

3.3.6 Reinforcement.

Definition 3.3.5: Reinforcement

An aggregation rule r fulfills *reinforcement* (also known as *consistency*) [Brandt et al., 2016] if, when a ranking R is returned by r on two distinct subsets of voters A and B, the same ranking R is returned by r on $A \cup B$.

For irresolute rules, in order to fulfill reinforcement, the rule has to return the intersection of the subsets of solutions returned on A and B if it is non empty. Since the PTA Kemeny rule sums the weighted Kendall tau score among the voters, it fulfills reinforcement.

Proposition 3.3.13: Reinforcement - PTA Kemeny

The PTA Kemeny rule fulfills reinforcement.

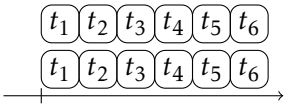
Proof. We consider two subsets of voters V_1 and V_2. Since the score is obtained by summing the weighted disagreements over all the voters, the score over $V_1 \cup V_2$ is the sum of the score on V_1 and the score on V_2. Therefore, if a schedule minimizes the PTA Kendall tau score on both V_1 and V_2, then it will minimize it on the union of the two subsets. □

Note that the PTA Kemeny rule fulfills reinforcement and PTA Condorcet consistency, whereas the already known aggregation rules [Pascual et al., 2018] for the collective schedule problem either fulfill one or the other but not both.

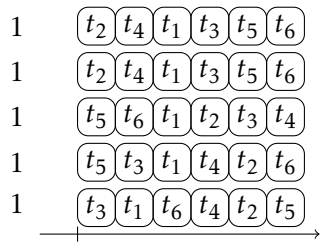
Proposition 3.3.14: Reinforcement - EMD

The EMD rule does not fulfill reinforcement.

Proof. Let us consider an instance with $n = 6$ unitary tasks $\{t_1, t_2, t_3, t_4, t_5, t_6\}$ and $v = 7$ voters divided into two groups V_1 and V_2. Group V_1 contains two voters whose preferences are as follows :



Group V_2 contains five voters :



The median completion times are as follows: $m_1(V_1) = 1, m_2(V_1) = 2, m_3(V_1) = 3, m_4(V_1) = 4, m_5(V_1) = 5, m_6(V_1) = 6$ and $m_1(V_2) = 3, m_2(V_2) = 4, m_3(V_2) = 4, m_4(V_2) = 4, m_5(V_2) = 5, m_6(V_2) = 6$. The EMD rule always returns the schedule $S = (t_1 \prec_S t_2 \prec_S t_3 \prec_S t_4 \prec_S t_5 \prec_S t_6)$ when applied on V_1. When applied on V_2 it returns the same schedule S and some other schedules. If EMD fulfilled reinforcement, then it should return S when applied on $V_1 \cup V_2$. For the profile $V_1 \cup V_2$ the median completion times are as follows: $m_1(V_1 \cup V_2) = 3, m_2(V_1 \cup V_2) = 2, m_3(V_1 \cup V_2) = 3, m_4(V_1 \cup V_2) = 4, m_5(V_1 \cup V_2) = 5, m_6(V_1 \cup V_2) = 6$. Therefore the EMD rule can return two schedules S_1 and S_2, $S_1 = (t_2 \prec_{S_1} t_1 \prec_{S_1} t_3 \prec_{S_1} t_4 \prec_{S_1} t_5 \prec_{S_1} t_6)$ and $S_2 = (t_2 \prec_{S_2} t_3 \prec_{S_2} t_1 \prec_{S_2} t_4 \prec_{S_2} t_5 \prec_{S_2} t_6)$. Both of these schedules are different from S: the EMD rule does not fulfill reinforcement. □

As mentioned in Section 3.3.2, this last result implies that the EMD rule is not based on a distance.

3.3.7 Unanimity.

Let us now focus on the *unanimity* axiom, a well-known axiom in social choice. This axiom states that if all the voters rank candidate t_a higher than candidate t_b then, in the consensus ranking, t_a should be ranked higher than t_b. We take the same definition, replacing "ranked higher" by "scheduled before":

Definition 3.3.6: Unanimity

Let P be a preference profile and r be an aggregation rule. The rule r fulfills *unanimity* iff when task t_a is scheduled before another task t_b in all the preferences in P, then t_a is scheduled before t_b in any solution returned by r.

Note that this axiom is interesting through its link with precedence constraints in scheduling. Indeed, if all the voters schedule a task before another one, it may indicate that there is a dependency between the two tasks (i.e. a task must be scheduled before the other one). A rule which fulfills the unanimity axiom can then infer the precedence constraints from a preference profile.

Proposition 3.3.15: Unanimity - EMD

The EMD rule fulfills unanimity.

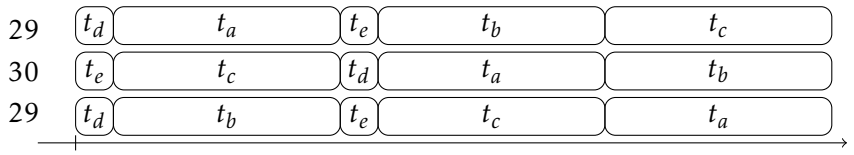
Proof. Let P be a preference profile in which a task t_a is always scheduled before a task t_b. Since t_a is always scheduled before t_b, for each voter, the completion time of t_a is strictly smaller than the completion time of t_b, and thus the median completion time of t_b is strictly larger than the median completion time of t_a. Therefore t_a is scheduled before t_b by EMD: the EMD rule fulfills unanimity. □

In [Pascual et al., 2018], the authors prove that the ΣT rule does not fulfill unanimity (this property is called Pareto efficiency in the paper). Let us now show that the ΣD does not fulfill this property either.

Proposition 3.3.16: Unanimity - ΣD

The ΣD rule does not fulfill unanimity for any tie-breaking mechanism.

Proof. Let us consider an instance with 5 tasks $\{t_a, t_b, t_c, t_d, t_e\}$ such that $p_a = p_b = p_c = 10$, $p_d = p_e = 1$ and $v = 88$ voters. We consider the following preferences.



In this example, a short task t_e is always scheduled before a long task t_c in the preferences. However in the unique optimal solution S for ΣD, which is $t_d \succ_S t_c \succ_S t_e \succ_S t_a \succ_S t_b$, t_e is scheduled after t_c. Therefore, the ΣD rule does not fulfill unanimity.

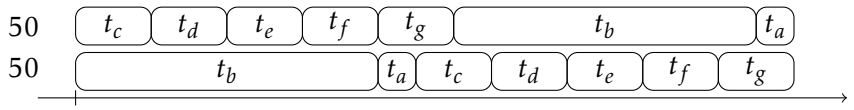
Note that, if we reverse all the schedules in the preference profile, then the long task t_c is always scheduled before the short task t_e but has to be scheduled after the t_e in the optimal solution, which is S but reversed. □

One could expect the PTA Kemeny rule to fulfill unanimity since the Kemeny rule does, and since it minimizes the pairwise disagreements with the voters. We can show that this is in fact not the case, by exhibiting a a counter-example.

Proposition 3.3.17: Unanimity - PTA Kemeny

The PTA Kemeny rule does not fulfill unanimity for any tie-breaking mechanism.

Proof. We consider an instance with $n = 7$ tasks $\{t_a, t_b, t_c, t_d, t_e, t_f, t_g\}$, such that $p_a = 1$, $p_b = 10$ and $p_c = p_d = p_e = p_f = p_g = 2$, and $v = 100$ voters. The preferences are as follows :



In this preference profile, the task t_b is always scheduled before t_a, however in the only optimal solution for PTA Kemeny, t_a is scheduled before t_b, indeed the optimal solution S is $t_a \prec_S t_c \prec_S t_d \prec_S t_e \prec_S t_f \prec_S t_g \prec_S t_b$. □

Note that unanimity is fulfilled if all the tasks are unit tasks. This has indeed already been shown for ΣT [Pascual et al., 2018], and this is true for PTA Kemeny since the Kemeny rule fulfills the unanimity axiom.

In our context, the unanimity axiom is not fulfilled because of the lengths of the tasks. It may indeed be better to disagree with the whole population in order to minimize the average delay or deviation, for ΣT and ΣD, or to disagree with the whole population if this disagreement has a small weight, in order to reduce other disagreements which have larger weights, for PTA Kemeny. Let us now restrict the unanimity axiom to the case where all voters agree to schedule a small task t_a before a large task t_b: we will see that the solutions returned by the PTA Kemeny rule always schedule t_a before t_b, that at least one optimal solution returned by ΣT also schedules t_a before t_b, whereas all the optimal solutions for ΣD may have to schedule t_b before t_a as we can see in the proof of proposition 3.3.16.

Proposition 3.3.18: Unanimity special case - PTA Kemeny

Let t_a and t_b be two tasks such that $p_a \leq p_b$. If task t_a is always scheduled before task t_b in the preferences of the voters, then t_a is scheduled before t_b in any optimal schedule for the PTA Kemeny rule.

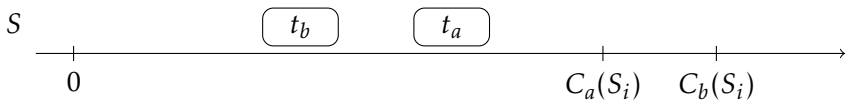
Proof. Let t_a and t_b be two tasks such that $p_a \leq p_b$. Let us assume, by contradiction, that there is a schedule S such that t_b is scheduled before t_a and which is optimal for the PTA Kemeny rule. Let $S_{(t_a \leftrightarrow t_b)}$ be the schedule obtained from S by swapping the position of t_a and t_b. Let t_c be a task different from t_a and t_b. If t_c is scheduled before t_b or after t_a in S, then the swap of t_a and t_b has no impact on the disagreements with t_c. If t_c is scheduled between t_a and t_b, then we have $b \succ_S c$ and $c \succ_S a$ and $a \succ_{S_{(t_a \leftrightarrow t_b)}} c$ and $c \succ_{S_{(t_a \leftrightarrow t_b)}} b$ (the order between t_c and the tasks other than t_a and t_b does not change between $S_{(t_a \leftrightarrow t_b)}$ and S). Task t_a is always scheduled before task t_b in the preferences and $p_a \leq p_b$, therefore the overall cost of scheduling t_a before t_c is smaller than or equal to the cost of scheduling t_b before t_c. Furthermore, since t_a is always scheduled before t_b in the preferences, scheduling t_a before t_b does not create a new disagreement, whereas the cost of scheduling t_b before t_a is equal to $v \cdot p_b$. The overall cost of S is then strictly larger than the cost of $S_{(t_a \leftrightarrow t_b)}$ which means that S is not optimal, a contradiction. □

Proposition 3.3.19: Unanimity special case - ΣT

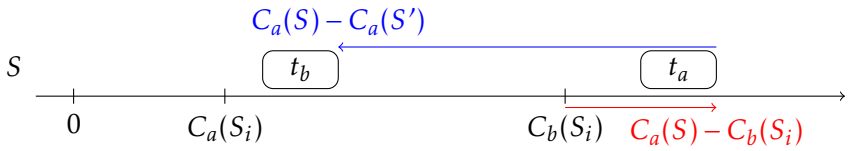
Let t_a and t_b be two tasks such that $p_a \leq p_b$. If task t_a is always scheduled before task t_b in the preferences of the voters, then t_a is scheduled before t_b in at least one optimal schedule for the ΣT rule.

Proof. Suppose that an schedule S is optimal for the ΣT rule and such that t_b is scheduled before t_a in S, $C_a(S) > C_b(S)$. By swapping the positions of t_a and t_b in S, we obtain a new feasible solution S' in which the completion times of all tasks but t_a and t_b are either smaller than or equal to the ones in S. The completion time of t_b in S' is the one of t_a in S and the completion time of t_a in S' is smaller than or equal to the one of t_b in S. The completion time of t_b in each preference is strictly higher than the completion time of t_a. For any voter i, there are two cases:

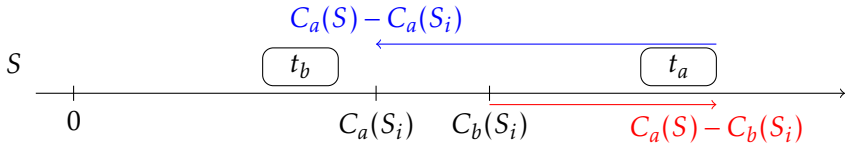
- Task t_a is completed in S before its completion time in S_i. In that case, if task t_b completes in S' at the same time than t_a in S, it will also complete before its completion time in S_i and therefore both tasks t_a and t_b will not be tardy, in S', just like in S.



- Task t_a is completed in S after its completion time in S_i. We will distinguish two subcases:
  - $C_a(S') \geq C_a(S_i)$. In that case the tardiness of t_a for voter i is decreased by $C_a(S) - C_a(S')$. On the other hand, the tardiness of task t_b is increased for voter v_i by at most $C_b(S') - C_b(S)$, since we have $C_a(S) = C_b(S')$ and $C_a(S') \leq C_b(S)$, the overall tardiness does not increase.



  - $C_a(S') < C_a(S_i)$. In that case, the tardiness of task t_a decreases by $C_a(S) - C_a(S_i)$ and the tardiness of t_b increases by at most $C_b(S') - C_b(S_i)$, since $C_b(S_i) > C_a(S_i)$, the tardiness does not increase overall.



□

Thus, if we are looking for a single solution for ΣT, we can restrict the search to solutions fulfilling the unanimity axiom for couples of tasks for which all the voters

agree that the smaller task should be scheduled first. For ΣD, we can guarantee solutions which fulfill this axiom for couples of tasks of the same length.

Proposition 3.3.20: Unanimity special case - ΣD

Let t_a and t_b be two tasks such that $p_a = p_b$. If task t_a is always scheduled before task t_b in the preferences of the voters, then t_a is scheduled before t_b in at least one optimal schedule for the ΣD rule.

Proof. Suppose that an schedule S is optimal for the ΣD rule and such that t_b is scheduled before t_a in S and $p_a = p_b$. By swapping the positions of t_a and t_b in S, we obtain a new feasible solution $S_{(t_a \leftrightarrow t_b)}$ in which the completion times of all tasks but t_a and t_b are equal to the ones in S. The completion time of t_b in $S_{(t_a \leftrightarrow t_b)}$ is the one of t_a in S and the completion time of t_a in $S_{(t_a \leftrightarrow t_b)}$ is lower or equal to the one of t_b in S. The completion time of t_b in each preference is strictly higher than the completion time of t_a. Therefore, in $S_{(t_a \leftrightarrow t_b)}$ the deviation of t_b which is ending in $S_{(t_a \leftrightarrow t_b)}$ at the time t_a ends in S, is lower or equal to the deviation of t_a in S. Similarly the deviation of t_a in $S_{(t_a \leftrightarrow t_b)}$ is lower or equal to the deviation of t_b in S. Overall, the deviation of $S_{(t_a \leftrightarrow t_b)}$ is lower or equal to the deviation of S. Their deviation are equal if both t_a and t_b are scheduled before their minimum completion time or both after their last completion time in the preference profile. □

We have seen that with the ΣT and PTA Kemeny rules, if a task t_a is scheduled before a task t_b by all voters and t_a is not longer than t_b, then there exists an optimal solution which schedules t_a before t_b. This is not the case for ΣD. In EB settings, we would expect well supported short tasks to be scheduled before less supported large tasks. Therefore the ΣT and PTA Kemeny rules seem well adapted for EB settings, while the ΣD rule seems less relevant in these settings.

3.3.8 Summary of the axiomatic properties of the rules.

We summarize the results shown in this section in Table 3.1. A sign "*" means that the property has been showed by Pascual et al. [2018], the other results are shown in this chapter.

3.4 Computational complexity and algorithms.

3.4.1 Complexity.

In this section we study the computational complexities of the ΣD and the PTA Kemeny rules. We will then focus on resolution methods for these rules. The ΣT rule has already been proven to be strongly NP-hard [Pascual et al., 2018]. In the same work, authors use linear programming to solve instances up to 20 tasks and 5000 voters, which is satisfactory since realistic instances are likely to have few tasks and a lot of voters.

Rule	N	PTA N	R	PTA C	LRM	Distance	Unanimity (t_a before t_b)		
							$p_a < p_b$	$p_a = p_b$	$p_a > p_b$
PTA K	✗	✓	✓	✓	?	✗	✓	✓	✗
ΣT	✗	✓	✓*	✗*	?	✗	~	~	✗*
ΣD	✗	✓	✓*	✗	✗	✓	✗	~	✗
EMD	✗	✓	✗	✗	✓	✗	✓	✓	✓

Table 3.1: Fulfilled (✓) and unfulfilled (✗) axioms by the PTA Kemeny, ΣT, ΣD and EMD rules. Symbol ~ means that the property is fulfilled by at least one optimal solution. The acronyms in the columns correspond to: neutrality (N), PTA neutrality (PTA N), reinforcement (R), PTA Condorcet consistency (PTA C), length reduction monotonicity (LRM).

The PTA Kemeny rule is NP-hard to compute since it is an extension of the Kemeny rule, which is NP-hard to compute [Bartholdi et al., 1989]. Most of the algorithms used to compute the ranking returned by the Kemeny rule can be adapted to return the schedule returned by the PTA Kemeny rule, by adding weights on the disagreements in the resolution method. In the following section, to compute schedules returned by the PTA Kemeny rule, we will use a weighted adaptation of an exact linear programming formulation for the Kemeny rule [Conitzer et al., 2006].

Regarding the ΣD rule, when there are exactly two voters, the problem is easy to solve: we return one of the two schedules in the preference profile (since deviation is a distance, any other schedule would have a larger deviation to the profile because of triangle inequalities). In the general case, the problem is NP-hard, as shown below.

Theorem 3.4.1: Strong NP-hardness of ΣD

The problem of returning a schedule minimizing the total absolute deviation is strongly NP-hard.

Proof. In order to prove that computing the schedules returned by ΣD is NP-hard, we start by introducing and proving a preliminary lemma. In the sequel, for reasons of readability, we will denote by ΣD_P the problem which consists in returning a schedule which minimizes the sum of the absolute deviations with the preference profile (i.e. ΣD_P is the problem solved by the ΣD rule).

In the sequel, we consider a polynomial time reduction from the problem $(1|no-idle|\Sigma D)$ which has been proven as strongly NP-hard when coded in unary [Wan and Yuan, 2013]. In this problem we consider an instance I composed of a set $\mathcal{J}$ of n tasks, each task t_i having a processing time $p_i \in \mathbb{N}^*$ and a due date d_i. By $L = \sum_{t_i \in \mathcal{J}} p_i$, we denote the overall load of the tasks. A feasible solution for this problem is a schedule S of all tasks in J on a single machine, with no idle time. We denote by $D(S)$ the sum of the absolute deviations $\sum_{t_i \in \mathcal{J}} |C_i(S) - d_i|$ where $C_i(S)$ is the completion time of task t_i in the schedule S. Given an integer B, the objective is to determine if a schedule S with

a total sum of absolute deviations $D(S)$ smaller than or equal to B exists. Since no task can complete before its processing time or after L (because there is no idle time), we can assume without loss of generality that $L \geq d_i \geq p_i$. If a task i had a deadline smaller than its processing time, then all feasible solutions would at least have a deviation of $p_i - d_i$ for task i, therefore, we can reduce B by that amount and have $d_i = p_i$; an analogous remark can be done for $d_i > L$.

From an instance I of the $(1|no-idle|\Sigma D)$ problem, we define an instance I' of the ΣD_P problem. We have a set $\mathcal{J}'$ of $n' = n + 4L$ tasks. For each task t_i of $\mathcal{J}$, there is a task t'_i in $\mathcal{J}'$ with $p'_i = p_i$. We denote by $J \subset \mathcal{J}'$ the set of tasks t'_1 to t'_n created from the tasks of $\mathcal{J}$. The set $\mathcal{J}'$ also contains $4L$ tasks of length 1. These $4L$ tasks are partitioned into 4 sets L_1, L_2, L_3 and L_4. Therefore the set of all the tasks $\mathcal{J}' = J \cup L_1 \cup L_2 \cup L_3 \cup L_4$. Instance I' has $4n$ voters: for each task $t_i \in \mathcal{J}$, we create 4 voters $v^i_1, v^i_2, v^i_3, v^i_4$ (see Figure 3.2).

- Voter v^i_1, of *type 1*, schedules first the tasks of L_1 , then the tasks of L_2, then the tasks of L_3, then the tasks of J and finally the tasks of L_4.
- Voter v^i_2, of *type 2*, schedules firstly the tasks of L_1, followed by the tasks of J, then the tasks of L_2, then the tasks of L_3 and finally the tasks of L_4.
- Voter v^i_3, of *type 3*, schedules task t'_i in order that this task completes at time $d_i + 2L$. The rest of the schedule is as follows: first, the tasks of L_2, then the tasks of J except t'_i, then the tasks of L_1 are scheduled around task t'_i. The schedule ends with the tasks of L_3 followed by the tasks of L_4.
- Voter v^i_4, of *type 4*, schedules task t'_i in order that this task completes at time $d_i + 2L$. The rest of the schedule is as follows: first, the tasks of L_1, then the tasks of L_2, then the tasks of L_4 are scheduled around task t'_i, then the tasks of J without t'_i. The tasks of L_3 end the schedule.

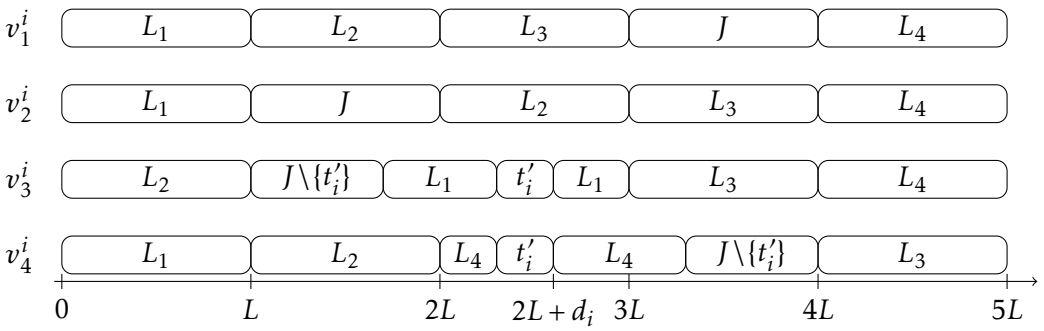


Figure 3.2: Voters associated with a task t'_i

The order on the tasks in each of the subsets L_1, L_2, L_3, L_4 is the same for all voters. For the set J, the order is the same for all voters, but, for each voter of type 3 and 4 one task is scheduled at a given time, regardless of its usual rank in the order.

Let us note that we can create such an instance in polynomial time since the instance for the $(1|no-idle|\sum D)$ problem is coded in unary.

Lemma 3.4.1: Structure of an optimal solution

Given an instance I of the $(1|no-idle|\sum D)$ problem, there exists an optimal solution for the instance I' of ΣD_P, created as described above, in which the tasks are scheduled as follows: L_1 first, L_2 second, $\mathcal{J}$ third, then L_3 and finally L_4.

Proof. Figure 3.3 illustrates the structure of an optimal solution as stated in this lemma. We prove this lemma by proving four facts.

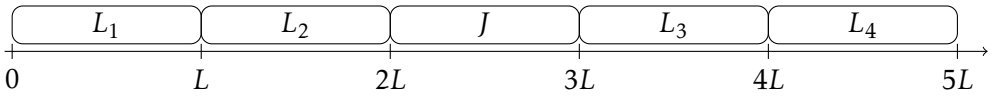


Figure 3.3: Structure of an optimal solution.

Fact 1: There exists an optimal schedule S^* in which the tasks of L_1 are scheduled before the tasks of L_2 and J.
To prove *fact 1*, we consider an optimal schedule S^* in which at least one task of L_1 is scheduled after a task of J or L_2. Thanks to proposition 3.3.20, we can consider that in S, the tasks of L_1 and L_2 are scheduled before the tasks of L_3 and L_4 and in the same order than in the preferences since the tasks of L_1 and L_2 are always scheduled before the tasks of L_3 and L_4 and the tasks of L_1 and L_2 are always scheduled in the same order. Let us call $x_1 \in L_1$ the first task of L_1 scheduled just after a task $x_2 \in L_2 \cup J$. We note $C_1(S^*)$ and $C_2(S^*)$ the completion times of tasks x_1 and x_2 in S^*. Since x_1 is the first task of L_1 to be scheduled just after a task of L_2 or J and since the tasks of L_1 are scheduled in the same order as in the preferences, task x_2 starts after the tasks of L_1 preceding x_1 in the preferences. We study the schedule S in which the tasks x_1 and x_2 are swapped. We distinguish two cases:

1. Task x_2 is in L_2: the swap changes the order on the tasks x_1 and x_2, the order on the tasks of L_1 (resp. L_2) is unchanged. Therefore x_1 (resp. x_2) is still scheduled after (resp. before) the tasks scheduled after it (resp. before it) in the preferences, which means that, in S, the task x_1 (resp. x_2) completes at or after (resp. at or before) its completion time for voters of type 1,2 and 4 (resp. for voters of type 2). Thus, for each voter of type 1 and 4, the absolute deviation is reduced by one, and reduced by two for each voter of type 2. Overall, the absolute deviation is reduced by $4n$. On the other hand, voters of type 1,3 and 4 could increase their deviations of 1 relatively to task x_2 and voters of type 3 could also increase their deviation for the task x_1 of 1. In the worst case, this increase is of $4n$, which equals the reduction, therefore S would also be optimal.

2. Task x_2 is in J: following the same reasoning, we can see that voters of type 1,2 and 4 will decrease their deviations for task x_1 by p_{x_2} with the swap. Voters of type 1 will also decrease their deviation for task x_2 by one since x_1 completes before $3L$ in S^* which implies that x_2 completes before $3L$ in S. Overall the reduction is of $3np_{x_2} + n$. Voters of type 2,3 and 4 could increase their deviation for x_1 by 1 and voters of type 3 could increase their deviation with x_1 by p_{x_2}, overall the increase is at most of $3n + np_{x_2}$, since $p_{x_2} \geq 1$ the increase is smaller than or equal to the decrease, S is also optimal.

In both cases, S is at least as good as S^*, therefore, from any optimal solution respecting proposition 3.3.20, we can iteratively obtain a new optimal solution in which the tasks of L_1 are scheduled before the tasks of L_2 and J.

Fact 2: There exists an optimal schedule S^* in which tasks of L_4 are scheduled after tasks of L_3 and J.
Fact 2 can be proved in an analogous way than *Fact 1*.

Fact 3: There exists an optimal solution S^* in which tasks of L_2 are scheduled before tasks of J.
We show that there is an optimal solution in which the tasks of L_2 are scheduled before the tasks of J. We consider an optimal solution S^*, respecting the previous facts and proposition 3.3.20, in which at least one task of L_2 is scheduled after a task of J. Let us denote by x_2 the first task of L_2 scheduled after a task of J. Let us call x_j the task of J scheduled just before x_2 in S^*. Note that such a task always exists since the task of L_1 are scheduled before the tasks of J and L_2 and the tasks of L_3 and L_4 are scheduled after the ones of L_2. We study the schedule S, similar to S^* except that x_2 and x_j are swapped. Since the tasks of L_2 are in the same order than in the preferences, and since we swap x_2 only with tasks of J, x_2 cannot complete in S before tasks of L_2 scheduled before it in S^*. Therefore, x_2 completes in S at least at the same time than in the preferences of voters of type 1 and 4. By swapping x_2 and x_j, we reduce the absolute deviation on x_2 for voters of type 1,3 and 4 by p_{x_j}, we also reduce absolute deviation on x_j for voters of type 1, by 1. Overall, we reduce the sum of absolute deviation by $3np_{x_j} + n$. We may increase the absolute deviation on x_j for voters of type 2,3 and 4 by one and deviation on x_2 for voters of type 2 by p_{x_j}, increasing the total sum of deviation by at most $3n + np_{x_j}$. Since $p_{x_j} \geq 1$, the increase is smaller than the decrease, therefore S is also optimal.

Fact 4: There exists an optimal solution S^* in which tasks of L_3 are scheduled after tasks of J.
We can prove *fact 4* in the same way than *fact 3*.

From Facts 1 to 4, we get Lemma 3.4.1. □

We can now go back to the proof of theorem 3.4.1.
In a schedule which follows the structure explained in Lemma 3.4.1, it is possible

to calculate the absolute deviations associated with the tasks of subsets L_1, L_2, L_3 and L_4:

- L_1: the tasks of L_1 are scheduled exactly like in the preference of voters of type 1, 2 and 4. The voter of type 3 associated with the task i has a delay of $L+(L-p_i)$ on the d_i first tasks of L_1 and a delay of $2L$ on the others. Overall, the deviation is: $n \times \sum_i d_i \times (2L-p_i) + (L-d_i) \times 2L$.
- L_2: Voters of type 1 and 4 have no deviation on the tasks of L_2. The $2n$ voters of type 2 and 3 have a deviation of L on each of the L tasks of L_2, which amounts to $2nL \times L$.
- L_3: Symmetrically to L_2, the sum of deviation of tasks of L_3 is also: $2nL \times L$.
- L_4: Voters of type 1,2 and 3 have no deviation on the tasks of L_4. For the voters of type 4: the first $d_i - p_i$ tasks of L_4 are delayed of $2L$, the rest of the tasks of L_4 are delayed by $2L-p_i$, which amounts to: $n \times \sum_i (d_i - p_i) \times 2L + (L-(d_i-p_i)) \times (2L-p_i) = n \times \sum_i (d_i - p_i)(p_i) + 2L^2 - Lp_i$.

Overall the sum of deviations M associated with the subsets L_1, L_2, L_3 and L_4 is:

$$M = \left(n \times \sum_i d_i \times (2L-p_i) + (L-d_i) \times 2L \right)$$

$$+ \left(2nL^2\right) + \left(2nL^2\right) + \left(n \times \sum_i (d_i - p_i)(p_i) + 2L^2 - Lp_i \right)$$

$$M = 4nL^2 + \left(n \sum_i -p_i d_i + 2L^2 + p_i d_i - p_i^2 + 2L^2 - Lp_i \right)$$

$$M = 4nL^2 + n \times \sum_i 4L^2 - p_i(L+p_i)$$

We now study the deviation of the tasks of J. The median completion time of task t'_i in J is $d'_i = 2L + d_i$. Let us see that, regardless of the order on the tasks of J in the preference, voters of type 1 and 2 will always have a total deviation on task t'_i of 2L. Since the order is the same for all the voters, the task will complete at a time $L+K$ with K an integer lower than L, in the preference of any voter of type 2 and at $3L+K$ in the preference of any voter of type 1. Therefore, in any schedule S, since the task completes at a time $C_i(S)$ between $2L$ and $3L$, we will have a total deviation of $C_i(S) - (L+K) + (3L+K) - C_i(S) = 2L$. We count then $2L$ for every pair of voter of type 1 and 2, which amounts to $2L \times n$ for each task, so the overall deviation of $2Ln^2$.

For the last two type of voters, for each task t'_i, we distinguish two cases:

1. Voters v_3^i and v_4^i both have scheduled t'_i so it completes at $2L + d_i = d'_i$. The deviation of a schedule S, regarding these two voters is therefore $2|C_i(S) - d'_i|$.

2. All other voters of type 3 and 4 schedule tasks of J in the same order except of one task t'_j, which is scheduled to complete at d'_j. Let us denote by K the integer such that task t'_i completes at $L+K$ in v_3^j, then t'_i completes at $3L+p_j+K$ in v_4^j. Since t'_i completes between $2L$ and $3L$ in the optimal solution S we are considering, we know that the deviation with v_3^j and v_4^j regarding task t'_i will be $C_i(S)-(L+K)+3L+p_j+K-C_i(S)=2L+p_j$. We calculate this value for all tasks, and call it N:

$$N=\sum_{t'_i\in J}\left(\sum_{t'_j\in J\setminus\{t'_i\}}2L+p_j\right)$$

$$N=\sum_{t'_i\in J}2L(n-1)+L-p_i=2Ln^2-Ln-L$$

By summing all these terms, the deviation of a solution S respecting lemma 3.4.1 is:

$$D(S)=M+2Ln^2+N+2\sum_{t'_i\in J}|C_i(S)-d'_i|$$

If a solution S with a cost lower than B exists for instance I of problem $(1|no-idle|\Sigma D)$, then there is a solution S' with a cost lower than $M+2Ln^2+N+2B$ for instance I' of ΣD_P. We can find this solution by reproducing the order on the task of $\mathcal{J}$ on the tasks on J. More precisely, S' respects Lemma 3.4.1 and schedules task from J in the order corresponding to S with the tasks of $\mathcal{J}$. We would have $C_i(S')=C_i(S)+2L$ and $d'_i=d_i+2L$. Therefore, for all i, we have $|C_i(S)-d_i|=|C_i(S')-d'_i|$. Since, $\sum_{t_i\in\mathcal{J}}|C_i(S)-d_i|\leq B$, we have $\sum_{t'_i\in J}|C_i(S')-dd'_i|\leq B$ and consequently, $D(S')\leq M+2Ln^2+N+2B$.
Reciprocally, if there exists a solution S' with a total deviation $D(S')$ smaller than $M+2Ln^2+N+2B$ for an instance I' of ΣD_P, we can create a solution S with a cost lower than B for an instance I of $(1|no-idle|\sum D)$ by recreating the order on the tasks of J on the tasks of $\mathcal{J}$.

We showed that there exists a solution of cost at most B for the $(1|no-idle|\Sigma D)$ problem for instance I iff there is a solution of cost at most $M+2Ln^2+N+2B$ for instance I' of ΣD_P, that we can obtain in polynomial time. Since $(1|no-idle|\Sigma D)$ is strongly NP-hard, ΣD_P is strongly NP-hard. □

Since computing an optimal schedule for ΣD is strongly NP-hard, we propose two resolution methods. First, we use linear programming, as it has been done with ΣT, allowing us to solve exactly instances up to 15 tasks in less than 30 minutes. Second, we propose to use the EMD rule as a heuristic and to use local search in order to improve the solution.

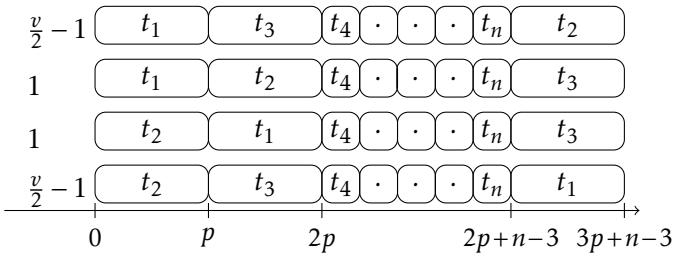
3.4.2 EMD with local search: a heuristic for ΣD and ΣT.

Since both the ΣD and ΣT rules solve NP-hard problem, we propose to use the EMD rule as a heuristic to solve these problems. As we will see in Section 3.5, EMD performs well in practice, even if, in the worst cases, it can lead to really unsatisfactory schedules, which can be shown by exhibiting a worst case instance.

Proposition 3.4.1: No α-approximation of EMD for ΣD and ΣT

For any $\alpha \geq 1$, EMD is not α-approximate for the total absolute deviation minimization, nor for the total tardiness minimization.

Proof. Let us consider an instance with v voters and n tasks. Tasks t_1, t_2 and t_3 are of size p, with p an integer and $n-3$ tasks $t_4, \ldots, t_n$ are of size 1. We consider the following preference profile:



In such an instance, tasks t_1 and t_2 have median completion times $m_1 = m_2 = p$. The task t_3 has a median completion time $m_3 = 2p$. Tasks t_4 to t_n have median completion times from $2p+1$ to $2p+n-3$. Therefore the EMD rule returns a schedule S with t_1 and t_2 first, in any order, then t_3 and finally t_4 to t_n in this order: $S = (t_1 \succ_S t_2 \succ_S t_3 \succ_S t_4 \succ_S \ldots \succ_S t_n)$. This solution has a sum of deviation of $\Sigma D(S) = vpn + vn - 3v - 4p$.

Let us now consider another solution $S' = (t_1 \succ_{S'} t_3 \succ_{S'} t_4 \succ_{S'} \ldots \succ_{S'} t_n \succ_{S'} t_2)$, we calculate its total deviation and find: $\Sigma D(S') = 2pv + vn - 3v + 2p + 2n - 6$.

We calculate the ratio between the two values:

$$\frac{vpn + vn - 3v - 4p}{2pv + vn - 3v + 2p + 2n - 6}$$

When p, n and v tend towards $+\infty$ the ratio tends towards $+\infty$ as well. Therefore the EMD rule can return a schedule with a sum of deviations arbitrarily far from the optimal one.

We can use the same instance to show a similar result regarding ΣT. The tardiness of schedule $S = (t_1 \succ_S t_2 \succ_S t_3 \succ_S t_4 \succ_S \ldots \succ_S t_n)$ is $\Sigma T(S) = vpn + pv/2 + (n-3+p)v - 2$. The tardiness of $S' = (t_1 \succ_{S'} t_3 \succ_{S'} t_4 \succ_{S'} \ldots \succ_{S'} t_n \succ_{S'} t_2)$ is $\Sigma T(S') = (n-3+p) + (n-3+2p)v/2$. Once more, when n,v and p tend towards $+\infty$, the ratio tends towards $+\infty$. □

Local search. In order to improve the solution returned by the EMD rule, we propose a local search algorithm. We define the neighbourhood of a schedule S as the set of

schedules obtained from S in which two consecutive tasks have been swapped. If at least one neighbour has a total deviation (resp. tardiness) smaller than S, we choose the best one, and we restart from it. Otherwise, S is a local optimum and we stop the algorithm. At each step, we study $(n-1)$ neighbours: the complexity is linear with the number of steps. In our experiments, by letting the algorithm reach a local optimum, we saw that the result obtained is usually very close to its local optimum at n steps and, that the local search always ends before $2n$ steps: in practice, we can bound the number of steps to $2n$ without reducing the quality of the solution.

3.5 Experiments.

Instances. Since, up to our knowledge, there is no database of instances for the collective schedule problem, we use synthetic instances. We generate two types of preference profiles: uniform (denoted below by U), in which the preferences are drawn uniformly (each possible permutation of the task is as likely as the others); and correlated (C), in which the preferences are drawn according to the Plackett-Luce model [Plackett, 1975; Luce, 2012]. In this model, each task t_i has an objective utility u_i (the utilities of the tasks are drawn uniformly in the [0,1] interval). We consider that the voters pick the tasks sequentially (i.e. they choose the first task of the schedule, then the second, and so forth). When choosing a task in a subset J, each task t_i of J has a probability of being picked of $u_i / \sum_{t_j \in J} u_j$. The lengths of the tasks are chosen uniformly at random between 1 and 10 (the results do not differ when the lengths are chosen in interval [1,5]). For all the experiments, we use CPLEX, a linear programming solver, to compute an optimal solution for each rule.

Number of optimal solutions. For most of the instances that we have generated, our rules had only one optimal solution. This was the case for more than 99% of the instances for ΣT and ΣD. For PTA Kemeny, this was the case for about 90% (resp. 95%) of the instances of 100 voters (resp. 250 voters), and for 98% of the instances in the case of correlated instances of 250 voters.

Computation times. We run, on a 6-core Intel i5 processor, the two linear programming algorithms corresponding to the ΣD and PTA Kemeny rules. The mean computation times are given in Table 3.2.

		ΣD			PTA Kemeny		
Number of voters	P	$n=4$	$n=8$	$n=12$	$n=4$	$n=8$	$n=12$
50	U	0.01	0.28	10.4	0.004	0.02	0.05
	C	0.005	0.13	0.95	0.002	0.02	0.05
500	U	0.01	25.0	104.1	0.003	2.1	4.6
	C	0.006	13.4	47.6	0.003	1.3	3.8

Table 3.2: Mean computation times (s) for ΣD and PTA Kemeny.

These algorithms allow to solve small but realistic instances. Note that correlated

instances, which are more likely to appear in realistic settings, require less computation time than uniform ones. Note also that computing an optimal schedule for PTA Kemeny is way faster than an optimal schedule for ΣD.

Observation 3.5.1: Linear programming formulation

We now describe the linear programming formulation we used for these experiments. There are $n(n-1)$ binary variables $prec_{i,j}$. Variable $prec_{i,j}$ is equal to 1 if task t_i is scheduled before task t_j in the solution. To compute the cost of a solution we add n integer variables C_i which represent the completion time of task t_i. Finally, we add $n \cdot v$ variables $dev_{i,v}$ containing the deviation of task t_i for voter v_v. The objective is then to minimize the sum of the variables $dev_{i,v}$.

$$
\begin{array}{lrcll}
\text{minimize} & \displaystyle\sum_{v \in V} \sum_{i \in \mathcal{J}} dev_{i,v} & & & \\
\text{s. t.} & prec_{i,j} + prec_{j,i} & = & 1 & \forall i,j \in V^2, i \neq j \\
& prec_{i,j} + prec_{j,k} + prec_{k,i} & \leq & 2 & \forall i,j,k \in V^3, i \neq j, i \neq k, k \neq j \\
& p_j + \displaystyle\sum_{i \in \mathcal{J} \setminus \{j\}} prec_{i,j} \cdot p_i & = & C_j & \forall j \in \mathcal{J} \\
& d_{i,k} - C_i & \leq & dev_{i,k} & \forall i \in \mathcal{J}, \forall k \in V \\
& -(d_{i,k} - C_i) & \leq & dev_{i,k} & \forall i \in \mathcal{J}, \forall k \in V \\
& prec_{i,j} & \in & \{0,1\} & \forall i,j \in \mathcal{J}^2, i \neq j \\
& t_j & \in & \mathbb{N}^+ & \forall j \in \mathcal{J}
\end{array}
$$

Regarding the constraints, the first two lines ensure that the solution is a total order of the tasks. The third line ensures that variable C_j contains the completion time of task t_j. The following two lines make sure that variable $dev_{i,k}$ contains the deviation of task t_i for voter v_k.

Performance of EMD. We now evaluate the performance of the EMD algorithm in comparison to the optimal resolution in terms of computation time and total deviation. We compute the ratio $r = D(EMD,P)/D(S^*,P)$ (resp. $r = T(EMD,P)/T(S^*,P)$) where S^* is a schedule returned by ΣD (resp. ΣT) and EMD is a schedule returned by the EMD algorithm. We compute r before and after the local search. Results can be found in Figures 3.4 and 3.5. In these figures, the orange lines shows the median value of the ratio r, the boxes extend from the first quartile to the third quartile and the dots show the results outside of these quartiles.

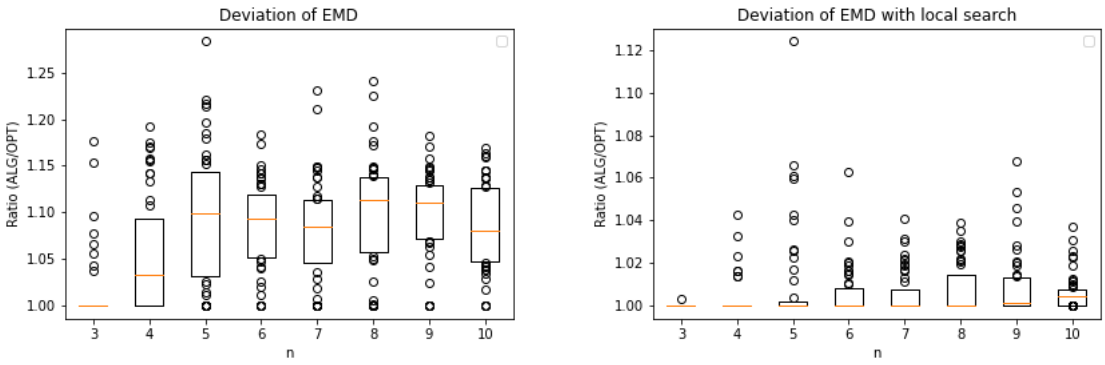


Figure 3.4: Ratio r between the deviation of the schedule returned by EMD without (left) and with local search (right) in comparison to the optimal solution returned by ΣD.

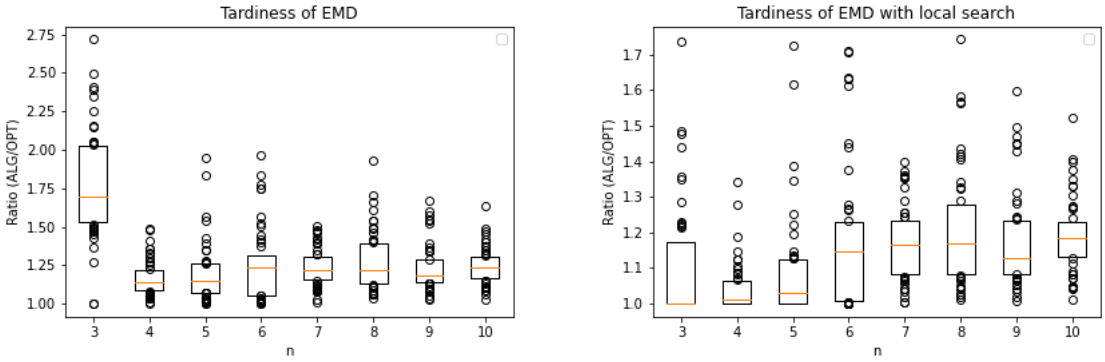


Figure 3.5: Ratio r between the tardiness of the schedule returned by EMD without (left) and with local search (right) in comparison to the optimal solution returned by ΣT.

The EMD algorithm alone returns solutions with a sum of deviations (resp. tardiness) about 10% (resp. 25%) higher than the optimal sum of deviations (resp. tardiness). With local search, the solution improves, especially for ΣD, with on average a sum of deviations (resp. tardiness) less than 1% (resp. 20%) higher than the optimal one. In terms of computation time, for 10 tasks and 100 voters, the heuristic (EMD +local search) takes 0.037 seconds to return its solution before the local search, and 0.63 seconds in total, while the linear program takes 4.5 seconds. This heuristic is thus a very fast and efficient alternative to the ΣD rule for large instances. The EMD rule does not seem to work as well to approximate ΣT than ΣD. This can be explained by the fact that a task scheduled at its median completion time has a minimum overall deviation but can be tardy for many voters. On a more general note, schedules returned by the ΣD and ΣT rules can be quite different, as we will see now, and the EMD rule returns

schedules close to the ones returned by the ΣD rule.

Difference between the ΣD, ΣT and PTA Kemeny rules. We execute the three rules on 300 instances, and we compare the schedules obtained with respect to the total deviation (ΣD), the total tardiness (ΣT) and the weighted Kendall Tau score (KT). We compare each schedule obtained to the optimal schedule for the considered metric. For example, the "1.06" in column ΣT of Table 3.3 means that, on average, for uniform instances with 5 tasks, the schedule returned by the ΣT rule has a sum of deviation 1.06 times larger than the minimal sum of deviation.

		ΣD		ΣT		PTA K	
P	M	$n=5$	$n=10$	$n=5$	$n=10$	$n=5$	$n=10$
U	ΣD	1	1	1.06	1.07	1.07	1.09
	ΣT	1.12	1.16	1	1	1.01	1.02
	KT	1.12	1.16	1.01	1.01	1	1
C	ΣD	1	1	1.05	1.09	1.05	1.07
	ΣT	1.06	1.08	1	1	1.001	1.001
	KT	1.07	1.07	1.002	1.01	1	1

Table 3.3: Performance of each rule relative to the others.

Table 3.3 shows that the schedules returned by ΣT and PTA Kemeny are very close to each other (the values they obtain are very close), while the ΣD rule returns more different schedules, even if the scores obtained by the three rules do not differ from more than 16% for uniform instances and 9% for correlated instances. Note that the number of tasks does not seem to change these results. Overall, the PTA Kemeny and ΣT rules return similar schedules, in which short tasks are favored, whereas the ΣD rule seems to return schedules as close as possible to the preference profile.

Length reduction monotonicity (axiom LRM). We have proved in Proposition 3.3.11 that the EMD rule fulfills length reduction monotonicity and we have shown in Proposition 3.3.12 that the ΣD rule does not fulfill this property. We now study to what extent the length reduction monotonicity axiom is fulfilled in practice for the ΣD, ΣT and PTA Kemeny rules. We run the three rules on 1200 instances with 50 voters and 8 tasks. Then, we reduce the length of a random task in each of the instances, and run the three rules again. If the reduced task starts later in the schedule returned by a rule than it did before the reduction, we count one instance for which the rule violates LRM. On the 1200 instances, PTA Kemeny and ΣT always respected LRM. The ΣD rule violated LRM in 102 instances (8.5%). This percentage goes up to 12.3% on uniform instances and up to 18% on uniform instances with tasks with similar lengths.

3.6 Discussion and conclusion

We studied, from an axiomatic and a computational viewpoint, four natural rules for the collective schedule problem. This problem generalizes the collective ranking prob-

lem, since the collective ranking problem can be viewed as the collective ranking problem with unit length tasks. Note the four studied rules either generalize well know rules for the collective ranking problem (the PTA Kemeny rule generalizes the Kemeny rule, and the ΣD rule generalizes Spearman's footrule), or are rules that do not seem to be used for the collective ranking problem yet (ΣT and EMD) – these rules may be interesting also in the context of collective ranking.

We have also introduced some new axioms, as PTA Neutrality, that may be useful in other context where items, or candidates, have weights. We showed incompatibilities between axioms, showing that neutral or distance based rules are not PTA Condorcet consistent and do not approximate the minimal sum of tardiness of the tasks.

Going back to our four rules, we saw that the PTA Kemeny and the ΣT rules seem to be particularly adapted in EB settings, where it is better for a task to be completed early (the ΣT rule seems well adapted to this setting by definition, and the PTA Kemeny rule because it fulfills in particular the PTA Condorcet property). From an experimental viewpoint, we also saw that the solutions returned by the ΣT and the PTA Kemeny rules are very close. On the contrary, the ΣD rule is, by construction, useful in non EB settings.

Despite it does not fulfill reinforcement, contrary to the three other rules, the EMD rule has several advantages: it can be computed in polynomial time (contrary to the three other rules), and it is the only one to guarantee that a task t_a will be scheduled before a task t_b if all the voters have scheduled t_a before t_b. This last remarks guarantees that the schedule returned by EMD fulfills precedence constraints, if there are precedence constraints between the tasks and that these constraints are fulfilled in the schedules given by the voters. The EMD rule also fulfills the Length Reduction Monotonicity axiom. This last point makes it a good candidate for EB settings. However, as seen in experiments, EMD approximates very well the rule ΣD, (and better than it approximates the ΣT and the PTA Kemeny rules): and is thus also, and above all, useful in non EB settings.

Chapter 4

Collective schedules: unit time and constraints

In this chapter, we will consider the collective schedules problem where all tasks have the same length. We study several algorithms taking preferences as parameters and returning a collective solution. These algorithms are based on two main criteria, extending the criteria presented in Chapter 3: a distance criterion, which generalizes the tardiness and the deviation criteria, and a binary criterion which generalizes the unit time penalty criterion [Brucker, 2010]. These algorithms return a solution minimizing either the binary or the distance criterion. This chapter focuses on classic scheduling constraints, namely the release dates, the deadlines and precedence constraints. We will consider two settings, one in which preferences fulfill the constraints and another one in which they do not necessarily fulfill them. In both cases the goal is to study the complexity and the mathematical properties of the algorithms. We study a fast heuristic algorithm for a special case of our problem with regards to its approximation ratio for the distance criterion and its behaviour regarding the constraints.

4.1 Introduction

The collective schedules problem [Pascual et al., 2018] consists in scheduling a set of n tasks shared by v individuals, also called voters. The tasks may represent talks of a conference that will be done in a same room, works to be done sequentially by coworkers, or events that will occur in one of the weekly meetings of an association. Each voter has his or her own preferences regarding the order in which theses tasks will be executed. We consider two models. In the first one, introduced by Pascual et al. [2018] and that we will call *Order Preferences*, each voter gives his or her preferred order – a permutation of the tasks. In the second model, that we introduce in this chapter, and that we call *Interval Preferences*, each voter gives for each task the interval in which he or she would like the task to be done. In this chapter, we focus on situations in which all the tasks have the same duration (a time slot per task). Our aim is, given the preferences of each individual, to compute a good compromise schedule of the n tasks.

Note here that, since the tasks are unit tasks, a schedule of the n tasks can be seen as a ranking of n tasks (or candidates). In this chapter, we will use several concepts – such as precedence constraints – from the scheduling field: we will therefore use the term schedule and not ranking for a permutation of the n tasks. Note however that several results of this paper are interesting not only in the context of scheduling but also in the context of ranking candidates.

The dissatisfaction of a voter if the returned schedule is schedule S is measured thanks to two families of criteria, coming from the scheduling theory field. One is a binary criterion, which says that a voter is satisfied if a task is not scheduled too late (or not too early) in S with respect to the preference of the voter (expressed as a order preferences or interval preferences). The other family of criteria is a distance criterion, which says that the closer the returned schedule is to the voter's preferences, the more satisfied a voter is.

We measure the quality of a compromise schedule S for all the voters by summing up the sum of the dissatisfaction of the voters for schedule S. This sum, divided by v, represents the average dissatisfaction of a voter with solution S. We focus on an utilitarian criterion: our aim is to compute a schedule with the smallest sum of dissatisfaction.

An assignment problem. Without additional constraints, this problem can be solved polynomially, both for Order and Interval Preferences, as it is an assignment problem. Indeed, the returned schedule being a permutation of the n tasks, we know that there will be n time slots, between 0 and n, one for each task. We create a complete bipartite graph with the tasks on the left and the time slots on the right. For each pair (task t, time slot s), the cost of the edge (t, s) is the sum of the dissatisfaction caused by task t to all the voters if t is scheduled at time slot s. Therefore, a schedule that minimizes the total dissatisfaction corresponds to a minimum cost matching in such a graph. The graph can be built in $O(vn^2)$, and a minimum cost matching can be found with Hungarian algorithm in $O(n^3)$ [Tomizawa, 1971; Edmonds and Karp, 1972], leading to a $O(vn^2 + n^3)$ algorithm.

Additional constraints. Our aim is to study this problem by adding the main constraints in scheduling: time constraints and precedence constraints. Time constraints mean that to each task is associated a release date and a due date (or deadline), and that in the returned schedule each task should be scheduled between its release date and its deadline. Precedence constraints mean that there is a precedence graph of the n tasks: if there is an edge from task t_i to task t_j in this graph, this means that in the returned schedule task t_i should be scheduled before task t_j. We will study both the case where these constraints are imposed, and the case where they are inferred from the preferences of the voters.

Overview of our results.

- We first start by introducing notations in Section 3.2, as well as formal definition of the binary and distance criteria studied in this chapter. As we will see, these criteria generalize the other criteria studied before (total tardiness, and total deviation), and also allow us to model famous scheduling criteria, as the minimization

of the total earliness of the task, or also the minimization of the number of late tasks. Rules that return optimal solutions of these criteria will be studied in the sequel.

- In Section 4.3, we focus on the algorithm which, in the Order Preference setting, computes the median start time of each task, and then schedules the tasks by increasing median start times (rule EMD – for Earliest Median Date). We show that, interestingly, this rule returns a schedule which is a 2-approximation of the total tardiness criterion.

- We then focus in Section 4.4 on time constraints: we show that it is still possible to get an optimal solution in polynomial time with time constraints on the tasks. We focus on the rules optimizing the binary and distance criteria (without time constraints), as well as the EMD rule, and we present an axiomatic study of these rules when time constraints are induced by the preferences of the voters (e.g. if all the voters schedule, in their preferred schedules, a task t at time X, is this task t necessarily started exactly at time X in the returned schedule ? If task t is always started after time X in the preferred schedules, is it always scheduled after time X in an optimal solution ?).

- In Section 4.5, we focus on precedence constrains between the tasks. We show that the previously studied rules, which could be run in polynomial time without precedence constraints, can still be used (with an additional polynomial time step) when the precedence constraints are inferred by the preference of the voters. On the contrary, we show that we have to solve NP-hard problem when the precedence constraints are not fulfilled by the preferred schedules of the voters. This is true both for the distance and the binary criterion, and in particular in the cases where we wish to minimize the total deviation, the total tardiness, or the number of late tasks.

- We conclude this paper in Section 4.6 by an overview of our results and a few research directions.

4.2 Preliminaries

4.2.1 Definitions and notations

Order Preferences and Interval Preferences.

We consider a set $\mathcal{J} = \{t_1, \ldots, t_n\}$ of n tasks of interest for a set $V = \{v_1, \ldots, v_v\}$ of v voters. Each task has a processing time of 1. The preferences of voter v_i are denoted by S_i, and depend of the setting used.

In the *Order Preferences* setting, each voter indicates its preferred schedule, as a permutation of the n tasks (we do not consider idle times between the tasks). Therefore, S_i is the preferred schedule of voter v_i. We denote by $C_j(S_i)$ the completion time of task

t_j in the preferred schedule of voter v_i. More generally, given a schedule S of tasks of $\mathcal{J}$, we denote by $C_j(S)$ the completion time of task t_j in S.

In the *Interval Preferences* setting, each voter indicates for each task the interval in which he or she wishes to see the task scheduled. More precisely, for each task $t_j \in \mathcal{J}$, voter $v_i \in V$ indicate a release date – that will be denoted by $r_{j,i}$ – , and which means that voter v_i would like task t_j to be started at the soonest at time $r_{j,i}$. Likewise, voter $v_i \in V$ indicates a due date (also called deadline) – that will be denoted by $d_{j,i}$ –, and which means that voter v_i would like task t_j to be completed at the latest at time $d_{j,i}$. Therefore, S_i is the set of the n pairs (release date, due date) that voter v_i sets for the n tasks. Note that this setting generalizes the Order Preferences settings, since it is possible for a voter to set for each task a release date (resp. a due date) equal to its start (resp. its completion time) in its preferred schedule. The only constraint we impose is that there exists a feasible schedule that fulfills the time constraint given by a voter (i.e. in which each task t_j is scheduled in the interval $[r_{j,i}, d_{j,i}]$).

In the sequel, we will penalize schedules in which tasks are scheduled out of the intervals given by the voters. The Interval Preferences setting allows voters to express pretty precise preferences. Indeed, if a voter wants a task to be done before a given date t, and has no preference on the starting date of a task then she can indicate a release date of 0 and a due date of t. If her only wish is that a task starts after a given time t', then she can indicate a release date of t' and a due date of n. Finally, if a voter wants a task to start exactly at time t'', then she can give a release date of t'' and a due date of $t'' + 1$. This flexibility in the preferences allow voters to express situations in which they have different expectations regarding the task, from having no interest in a task to wanting it to be completed exactly at a given time.

Let us now present the two general objective functions that we will consider in this chapter: the binary criterion, and the distance criterion.

Binary criterion.

The first criterion, called *Binary Criterion*, measures whether a task is executed in the time interval indicated by a voter or not (the penalty is 0 if the tasks is scheduled in the desired interval, and is 1 if the task is not scheduled in the desired interval). Given a schedule S, the dissatisfaction of voter $v_i \in V$ concerning task $t_j \in \mathcal{J}$ is thus:

$$b_j(S, S_i) = \begin{cases} 1 & \text{if } C_j(S) > d_{j,i} \text{ or } C_j(S) \leq r_{j,i} \\ 0 & \text{otherwise} \end{cases}$$

The dissatisfaction of a voter v_i concerning a schedule S with the binary criterion is then:

$$B(S, S_i) = \sum_{t_j \in \mathcal{J}} b_j(S, S_i)$$

Distance criterion.

The second criterion, called *Distance Criterion*, also does not count any penalty when a task is scheduled in its time interval, but otherwise it counts a penalty which

expresses how far from its interval the task is. The dissatisfaction of voter v_i concerning task t_j for schedule S is:

$$\text{dis}_j(S, S_i) = \begin{cases} C_j(S) - d_{j,i} & \text{if } C_j(S) > d_{j,i} \\ r_{j,i} - (C_j(S) - 1) & \text{if } C_j(S) \leq r_{j,i} \\ 0 & \text{otherwise} \end{cases}$$

The dissatisfaction of a voter v_i concerning a schedule S with the distance criterion is then:

$$\text{Dis}(S, S_i) = \sum_{t_j \in \mathcal{J}} \text{dis}_j(S, S_i)$$

Aggregation function.

As said in the introduction, we will study the utilitarian utility function. Our aim will be to minimize $\Sigma_{v_i \in V} B(S, S_i)$ with the binary criterion, or $\Sigma_{v_i \in V} \text{Dis}(S, S_i)$ with the distance criterion. The *Binary Criterion rule* is an algorithm that returns a schedule minimizing binary criterion, while the *Distance Criterion rule* is an algorithm that returns a schedule minimizing distance criterion.

4.2.2 Generalization of classical scheduling criteria.

The two above defined criteria generalize the main criteria already studied in the Order Preferences setting (see Chapter 3 and [Pascual et al., 2018]). Assume indeed that voters have expressed their preferences using the Order Preferences setting (i.e. each voter indicates his or her preferred schedule). Let P be the preference profile.

- *Total deviation*. The total deviation of a schedule S is $D(S, P) = \Sigma_{v_i \in V} D(S, S_i)$, where $D(S, S_i) = \sum_{t_j \in \mathcal{J}} |C_j(S) - C_j(S_i)|$. If our aim is to compute a schedule of minimal total deviation, as does rule ΣD, then we should use the Distance Criterion by setting the release date of task t_j for voter v_i at $C_j(S_i) - 1$, and the due date of task t_j for voter v_i at $C_j(S_i)$.

- *Total tardiness*. The total tardiness of a schedule S is $T(S, P) = \Sigma_{v_i \in V} T(S, S_i)$, where $T(S, S_i) = \sum_{t_j \in \mathcal{J}} \max(0, C_j(S) - C_j(S_i))$. If our aim is to compute a schedule of minimal total tardiness, as does rule ΣT, then we should use the Distance Criterion by setting the release date of task t_j for voter v_i at 0, and the due date of task t_j for voter v_i at $C_j(S_i)$.

- *Total earliness*. The total earliness of a schedule S is $E(S, P) = \Sigma_{v_i \in V} E(S, S_i)$, where $E(S, S_i) = \sum_{t_j \in \mathcal{J}} \max(0, C_j(S_i) - C_j(S))$. If our aim is to minimize the total earliness, a classic criterion in scheduling [Brucker, 2010], then we should use the Distance Criterion by setting the release date of task t_j for voter v_i at $C_j(S_i) - 1$, and the due date of task t_j for voter v_i at n.

- *Total number of late tasks.* The total number of late tasks of a schedule S is $U(S,P) = \Sigma_{v_i \in V} U(S,S_i)$, where $U(S,S_i)$ is the number of tasks of $\mathcal{J}$ such that $C_j(S) > C_j(S_i)$ (such tasks are called *late* tasks). This a classic criterion, denoted by ΣU (for "Unit Penalties"), in scheduling [Brucker, 2010]. We can solve this optimization problem by using the Binary Criterion by setting the release date of task t_j for voter v_i at 0, and the due date of task t_j for voter v_i at $C_j(S_i)$.
- *Total number of tasks not well positioned.* If our aim is to maximize the number of tasks scheduled at the exact position given by the voters, then we should use the Binary Criterion by setting the release date of task t_j for voter v_i at $C_j(S_i)-1$, and the due date of task t_j for voter v_i at $C_j(S_i)$.

In the next section, we introduce the EMD rule and show that it is a 2 approximation of the total tardiness (ΣT) and total earliness (ΣE) criteria.

4.3 An analysis of the EMD rule

The EMD rule, introduced in Chapter 3 in the Order Preferences setting, schedules the tasks by increasing median completion times, where the median time of a task t_j is the median of the set $\{C_j(S_1),\dots,C_j(S_v)\}$. If several tasks have the same median completion time, any tie breaker mechanism can be used.

It was shown previously [Pascual et al., 2018] that, for unit size tasks, and for any preference profile P and any schedule S, we have $D(S,P) = 2T(S,P)$, and thus that $T(S,P) = E(S,P)$ since $D(S,P) = E(S,P) + T(S,P)$. Therefore, a α-approximate algorithm for the total tardiness criterion will also be an α-approximate algorithm for the earliness criteria, and an α-approximate algorithm for the total deviation criterion.

We consider that we are in the Order Preferences setting. Before showing that EMD is 2-approximate for the total tardiness criterion (and thus also for the deviation and earliness criterion), we introduce a way to see the instance that will facilitate the analysis.

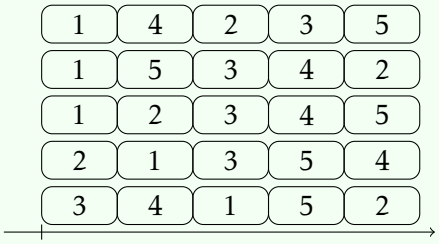
Breaking down the preference profile. We "break down" the preference profile not by looking at voters individually, but by looking at time slots. Note that this does not change the preference profile: it is just another way of looking at it. For each time slot between 1 and n, each voter v_i selected a task that she has scheduled in this time slot in her preferred schedule. We call *choice* a triplet (S_x, t_j, s) indicating that voter v_x schedules task t_j in time slot s, i.e. between time $s-1$ and s, in her preference S_x. We can thus express a preference profile as a set of choices, such that there are v choices for each time slot and there are n choices for each voter, each task and each slot being chosen exactly once by each voter. We denote by $\mathscr{C}$ the set of all the choices and, for each $y \in \{1\dots n\}$, we denote by $\mathscr{C}_y$ the set of all choices (S_x, t_j, s) such that $s \leq y$.

Iterative breakdown of the tardiness criterion. As we have seen, the total tardiness of a schedule S given a preference profile is the sum, over all voters, of the tardiness of each task in S in comparison to its completion time in the preference of the voter. By

breaking down the set of preferences into choices, it is possible to express the tardiness in another way, that will facilitate the analysis of the algorithm. If a task t_j has been scheduled by a voter v_x at time slot s, then, if it is not scheduled in a solution S by time s, we count a penalty; if it is not scheduled by time $s+1$, we count another penalty; and so forth. We can then split the tardiness by looking at the tasks scheduled by S at time slots: for each slot between $C_j(S_i)$ to n, if task t_j has not been scheduled yet, then we count 1 tardiness penalty (for voter i). We sum this over all the voters. By this way, we compute for each slot s the number of penalties caused by the decision taken in s – there will be 1 penalty of each pair (voter v_x, task t_j) if task t_j has not been completed at time s whereas $C_j(S_x) \leq s$.

Example 4.3.1: Tardiness caused by the choice for the first slot

Let us consider an instance with 5 voters and 5 tasks as follows. Each line represents the preferred schedule of a voter – e.g. the preferred schedule of the first voter is made of task 1, then task 4, followed by task 2, then task 3 and finally task 5 (such a schedule can be written as: $1 \prec 4 \prec 2 \prec 3 \prec 5$):

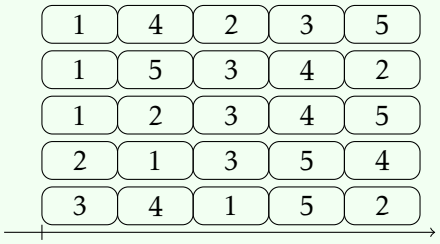


Looking at time slot 1 (between dates 0 and 1), task 1 has been scheduled three times, task 2 once and task 3 once. In a solution S, scheduling task 1 at slot 1 causes a (total) tardiness of 2 since task 2 and 3 which were chosen by two voters will not be scheduled on time. Scheduling task 2 or task 3 causes a tardiness of 4, and scheduling task 4 or task 5 creates a tardiness of 5.

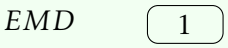
In the proof of the following proposition, to compute the sum of the tardiness (also called the total tardiness) of a schedule S, we will look a time slots, starting from the first one, between dates 0 and 1, to the last one, between dates $n-1$ and n. When looking at time slot y, for each choice $(S_x, t_j, s) \in \mathcal{C}_y$, we will count a penalty if task t_j has not been scheduled at time slot y or before. We denote by k_y the number of late tasks at time slot y: $k_y(S,P) = \sum_{(S_x,t_j,s)\in\mathcal{C}_y} \mathbb{1}_{C_j(S)>y}$. The total tardiness can be expressed as follows: $T(S,P) = \sum_{y=1}^{n} k_y(S,P)$.

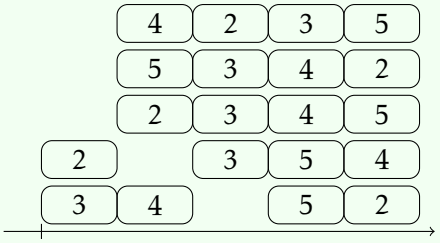
Example 4.3.2: Computing the total tardiness with choices

Let us consider an instance with 5 tasks and 5 voters, thepreference profile P is as follows:



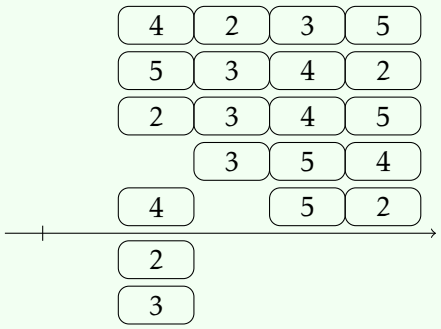
The EMD rule computes the median completion time of all tasks and returns a schedule in which tasks are ordered by non decreasing median completion time. In this example, we have: $(1 \prec 2/3 \prec 4/5)$, let us call S the schedule returned by EMD. The first task to be scheduled is 1. By scheduling 1 in the first slot, this means that all the choices (S_x, t_1, s) are not counted in $k_s(S,P)$ for any $s \geq 1$, i.e. any s. Intuitively, the task scheduled in the first slot is never late for any voter.





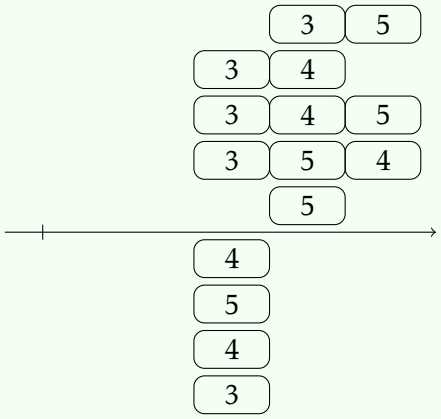
On the other hand, the other choices $(S_x, t_j, 1)$ for $t_j \neq 1$ are counted by $k_1(S,P)$, in this example we have two choices, since one voter scheduled task 2 in slot 1 and one scheduled task 3 in slot 1, which means $k_1(S,P) = 2$.

EMD 1 $Tardiness = 2$

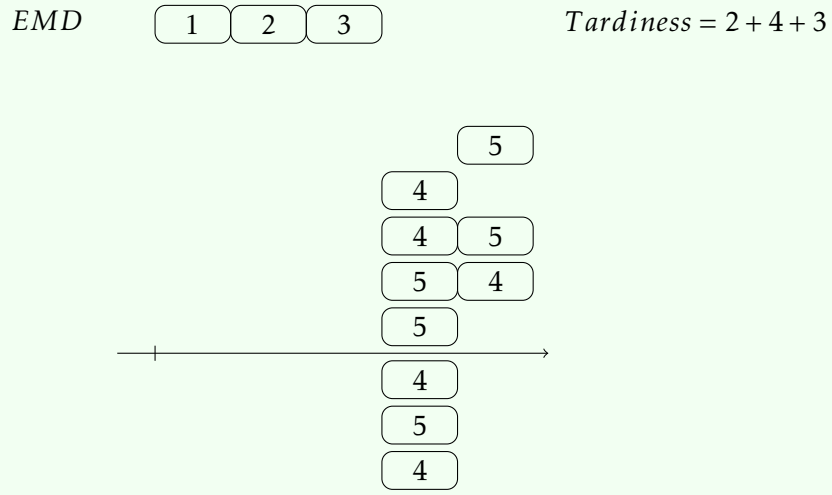


We keep going with slot 2. The EMD rule schedules either task 2 or task 3. Let us suppose that it selects task 2. The value of $k_2(S,P)$ is then the number of choices in $\mathscr{C}_2$ and for which the task is not 1 nor 2, i.e. 3 choices for the slot 2 plus one choice remaining from the first slot.

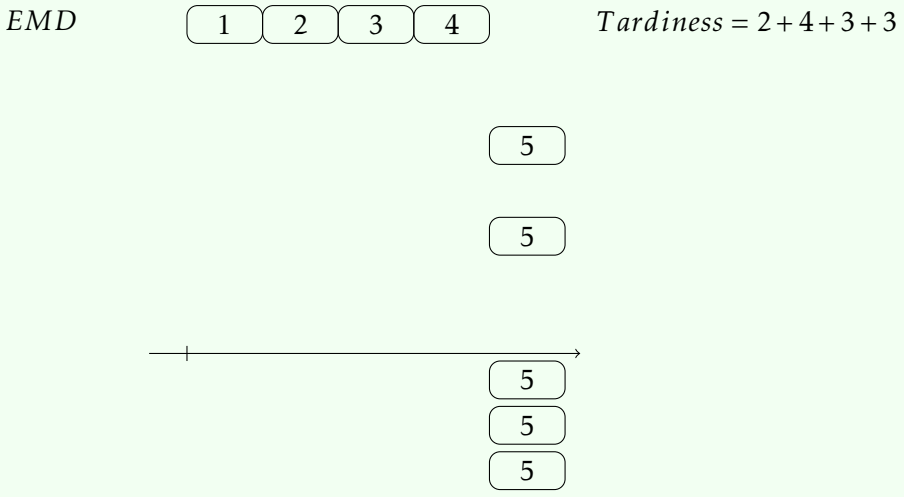
EMD 1 2 $Tardiness = 2 + 4$



We continue with the third slot. The EMD rule schedules task 3 and $k_3(S,P) = 3$.



For slot 4 the EMD rule schedules either task 4 or task 5, let us assume that it selects task 4. There are 3 choices in $\mathscr{C}_4$ for which the task is not in $\{1,2,3,4\}$, therefore $k_4(S,P) = 3$.



For the last slot, there is only one task left and there are no remaining choice so $k_5(S,P) = 0$ (more generally, $k_n(S,P) = 0$. We obtain the total tardiness by summing $k_1 + k_2 + k_3 + k_4 + k_5 = 2 + 4 + 3 + 3 + 0 = 12$.

Proposition 4.3.1: EMD- 2-approximation for ΣT

The EMD rule is 2-approximate for the total tardiness criterion.

Proof. Let us consider a preference profile P. Let S be the schedule returned by the

EMD rule and let S^* be a schedule minimizing the total tardiness with respect to preference profile P. We prove this result by showing that for all $i \geq 0$, $k_i(S,P) \leq 2k_i(S^*,P)$.

For $i = 0$, we have $k_i(S,P) = k_i(S^*,P) = 0$, since no task is scheduled before the first time slot. For any time slot from 1 to n, we express $k_i(S,P)$ as the difference between $i \times v$, the total number of choices from time slot 1 to time slot i, and the number of choices (S_x, t_j, s) such that $s \leq i$ and t_j has been scheduled at the latest at time slot i in S. We denote by q_i the number of tasks with median completion time smaller than or equal to i. There are two cases:

1. $q_i \leq i$: in this case, the EMD rule schedules the q_i tasks with median completion time smaller than or equal to i in the i first time slots. Let q_i^* be the number of tasks with median completion time smaller than or equal to i that are scheduled in S^* at the latest at date i. These q_i^* tasks are necessarily scheduled before date i by the EMD rule as well. Let $\mathcal{Q}_i^*$ be the set of the q_i^* tasks of median completion time smaller than or equal to i and that are scheduled in S^* before or at time i.

   Finally, we denote by Q_i^* the number of choices (S_x, t_j, y) of $\mathcal{C}_i$ such that $t_j \in \mathcal{Q}_i^*$ and $y \leq i$. These choices are removed from the $i \times v$ choices for both the solutions S and S^*. There are $q_i - q_i^*$ tasks of median completion time smaller than or equal to i that are scheduled in S before or at time i and that are scheduled after i in S^*. For each of these tasks, there are at least $v/2$ choices among the $i \times v$ which are removed by scheduling the task before date i. There are also $(i - q_i)$ tasks with median completion time strictly larger than i that are scheduled at the latest at date i in S, but we have no guarantee that scheduling these tasks remove any choice. We therefore have:

   $$k_i(S,P) \leq i \times v - Q_i^* - (q_i - q_i^*)v/2$$

   In S^*, there are $(i - q_i^*)$ tasks of median strictly larger than i, at most, scheduling these tasks before or at time i removes at most $v/2$ choices. We then have:

   $$k_i(S^*,P) \geq i \times v - Q_i^* - (i - q_i^*)v/2$$

   We then compute:

   $$2k_i(S^*,P) - k_i(S,P) \geq 2i \times v - 2Q_i^* - (i - q_i^*)v - i \times v + Q_i^* + (q_i - q_i^*)v/2$$

   $$2k_i(S^*,P) - k_i(S,P) \geq q_i^* v - Q_i^* + (q_i - q_i^*)v/2$$

   We know that $Q_i^* \leq q_i^* v$ since each task in $\mathcal{Q}_i^*$ is scheduled at most v times in the preference profile, once per voter. We also know that $q_i \geq q_i^*$. We then have:

   $$2k_i(S^*,P) - k_i(S,P) \geq 0$$

2. $q_i > i$: in this case, the EMD rule schedules, from dates 0 to i, exactly i tasks of median completion time smaller than or equal to i. There remains $(q_i - i)$ tasks of median completion time smaller than or equal to i that are not scheduled by date

i in S, the schedule returned by the EMD rule. Each of these tasks can appear in at most v choices in $\mathscr{C}_i$.

Let $r_i \geq 0$ be the number of tasks with median completion time strictly larger than i that are not scheduled by date i in S. Let $\mathscr{R}_i$ the set of these r_i tasks, and let R_i the set of choices (S_x, t_j, s) in $\mathscr{C}_i$ such that $j \in \mathscr{R}_i$. We have:

$$k_i(S,P) \leq (q_i - i)v + |R_i|$$

In S^*, the i tasks scheduled by time slot i are split between the q_i tasks of median completion time smaller than or equal to i and the r_i tasks of median completion time strictly larger than i. We denote by $\mathscr{R}_i^*$ the tasks of $\mathscr{R}_i$ scheduled by S^* before or at time i. We call $r_i^* = |\mathscr{R}_i^*|$, and R_i^* the set of choices (S_x, t_j, s) of $\mathscr{C}_i$ such that $j \in \mathscr{R}_i^*$. There are $(q_i - (i - r_i^*))$ tasks of median completion time smaller than or equal to i that are not scheduled by S^* by time i. Each of these tasks is at least in $v/2$ choices in $\mathscr{C}_i$. We can then write:

$$k_i(S^*,P) \geq (q_i - i + r_i^*)v/2 + |R_i| - |R_i^*|$$

We then have:

$$2k_i(S^*,P) - k_i(S,P) \geq (q_i - i + r_i^*)v + 2|R_i| - 2|R_i|^* - (q_i - i)v - |R_i|$$

$$2k_i(S^*,P)^* - k_i(S,P) \geq r_i^* \times v + |R_i| - 2|R_i^*|$$

Since $|R_i| \geq |R_i^*|$ and $r_i^* \times v > |R_i^*|$, we have: $2k_i(S^*,P) - k_i(S,P) \geq 0$.

In both cases, we have $k_i(S,P) \leq 2k_i(S^*,P)$ for all $i \geq 1$. Therefore, we have: $\sum_{i=1}^{n} k_i(S,P) \leq 2\sum_{i=1}^{n} k_i(S^*,P)$ and then $\Sigma T(S,P) \leq 2\Sigma T(S^*,P)$. □

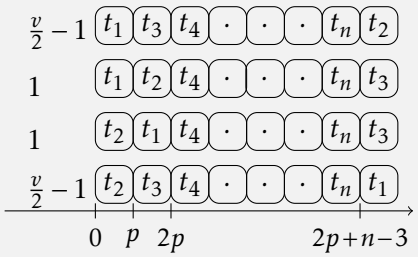
As seen above, since, with unitary tasks, $T(S,P) = E(S,P)$ and $D(S,P) = 2T(S,P)$, for any schedule S and preference profile P, we get the following corollary.

Corollary 4.3.1: EMD rule - 2-approximation for ΣD and ΣE

The EMD rule is 2-approximate for the ΣD criterion, and for the ΣE criterion.

Observation 4.3.1: Tight bound for EMD

We note here that this 2-approximation is tight. We consider the instance used in the proof of Proposition 3.4.1 with $p = 1$, and that we represent below:



The ratio between the deviation of the solution returned by EMD and the optimal deviation was:

$$\frac{vpn+vn-3v-4p}{2pv+vn-3v+2p+2n-6}$$

By replacing p by 1, we have:

$$\frac{2vn-3v-4}{vn-v+2n-4}$$

When n and v tend towards $+\infty$, the ratio tends towards 2.

Additional results in voting theory As noted earlier, the minimization of the deviation criterion when tasks are of unit length is equivalent to the minimization of the Spearman correlation coefficient. This means that the EMD rule returns a ranking that is 2-approximate for the minimization of the Spearman coefficient. Although this minimization problem is polynomially solvable, it is still an interesting property to have for the EMD rule.

Corollary 4.3.2: EMD- 2-approximation of the Spearman rule

The EMD rule is 2-approximate for the minimization of the total Spearman correlation coefficient to the preference profile.

We can show that EMD is 4-approximate for the Kemeny rule.

Proposition 4.3.2: EMD- 4-approximation of the Kemeny rule

The EMD rule is 4-approximate for the minimization of the Kendall-Tau distance to the preference profile.

Proof. Diaconis and Graham [1977] showed that for any ranking R and any preference profile P, the Spearman correlation coefficient ρ (see Example 1.2.3) fulfills the following property: $\Delta_{KT}(R,P) \leq \rho(R,P) \leq 2\Delta_{KT}(R,P)$. We call S the solution returned by EMD, S^*_{KT} a solution minimizing the Kendall-Tau distance to the preference profile and S^* a solution minimizing the total Spearman correlation coefficient with the preference

profile.

For the sake of contradiction, let us assume that:

$$\Delta_{KT}(S,P) > 4\Delta_{KT}(S^*_{KT},P)$$

We then have

$$\rho(S,P) \geq \Delta_{KT}(S,P) > 4\Delta_{KT}(S^*_{KT},P) \geq 2\rho(S^*_{KT},P)$$

And since S^* is optimal for the Spearman, rule we have:

$$\rho(S,P) \geq \Delta_{KT}(S,P) > 4\Delta_{KT}(S^*_{KT},P) \geq 2\rho(S^*_{KT},P) \geq 2\rho(S^*,P)$$

A contradiction, given the result from Proposition 4.3.1. We therefore have:

$$\Delta_{KT}(S,P) \leq 4\Delta_{KT}(S^*_{KT},P)$$

□

In the next section, we focus on release time and due dates constraints.

4.4 Scheduling tasks with time constraints

We first show that it is still possible to compute in polynomial time an optimal solution of the total dissatisfaction of the voters with both the Binary criterion and the Distance criterion presented in Section 4.2.

4.4.1 Getting optimal solutions with time constraints

Let us consider that each task $t_j \in \mathcal{J}$ has a release date r_j and a due date d_j. These dates can be imposed, for example when the tasks represent events that cannot occur before a date r_j or after a date d_j. They can also be inferred from the preferences of the voters (by setting $r_j = \min_{v_i \in V}\{r_{j,i}\}$ and $d_j = \max_{v_i \in V}\{d_{j,i}\}$). In this case, we want no task to be scheduled earlier than in the preferred interval of any voter, or later than in the preferred interval of any voter. This case is particularly interesting if voters are aware of real time constraints on the events that are represented by the tasks, and if the scheduler does not necessarily know these constraints.

Returning an optimal schedule for both the Binary criterion and the Distance criterion is, as without any time constraints, an assignment problem. In the bipartite graph with the tasks on the left and the time slots on the right, for each pair (task t_j, time slot s), we just put an edge between t_j and s if and only if $r_j \leq s \leq d_j - 1$. The costs of the edges are equal to the sum of the dissatisfaction of the v voters if task t_j is scheduled between $s-1$ and s. An optimal solution which fulfills time constraints – if there is one feasible solution – is a minimum cost matching. Such a matching, if it exists, can be found with Hungarian algorithm in $O(n^3)$ [Tomizawa, 1971; Edmonds and Karp, 1972].

In the next section, we study to which extent the rules presented earlier propagate constraints fulfilled by the preferences of the voters. For example, if all the voters schedule a task after a given time, it may be because this task is not available before. This is particularly interesting in contexts in which the preferences given are not necessarily votes but feasible solutions for a problem (potentially optimizing different aspects like cost, employee satisfaction, inventory management ...). In this case, the question becomes: given several feasible solutions satisfying a set of constraints, does the aggregation rule ensure that the returned solution fulfills the same constraints?

4.4.2 Axiomatic study of rules with inferred time constraints

Release dates and deadlines consistencies.

The idea of the two following axioms is the following one: if a task t_j starts after (resp. ends before) a given date s in the preferences of all the voters, we can interpret it as s being a firm release date (resp. deadline) for task t_j. In this case, we would like the rule to return a solution in which t_j starts after (resp. ends before) s.

Definition 4.4.1: Release Date Consistency

Let V be a set of voters and t_j a task such that for each preference S_i expressed by voter $v_i \in V$, we have $C_j(S_i) \geq s$, with s a constant. An aggregation rule fulfills *release date consistency* if it always returns a schedule S in which $C_j(S) \geq s$.

Definition 4.4.2: Deadline Consistency

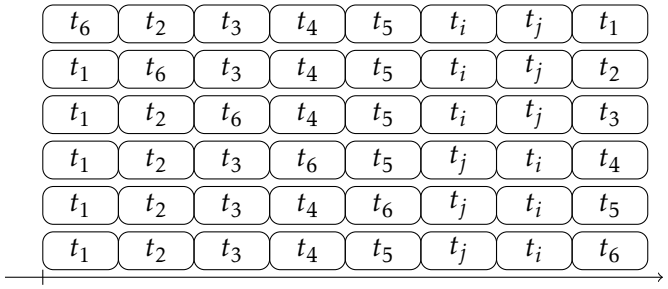
Let V be a set of voters and t_j a task such that for each preference S_i expressed by voter $v_i \in V$, we have $C_j(S_i) \leq s$, with s a constant. An aggregation rule fulfills *deadline consistency* if it always returns a schedule S in which $C_j(S) \leq s$.

We show that the Distance Criterion rule, the Binary Criterion rule, and the EMD rule do not fulfill these axioms.

Proposition 4.4.1: Distance - Deadline and Release Date Consistency

The Distance Criterion rule does not fulfill the deadline consistency nor the release date consistency, even when preferences are expressed as schedules.

Proof. Let us consider an instance with 8 tasks and 6 voters and the following preferences:



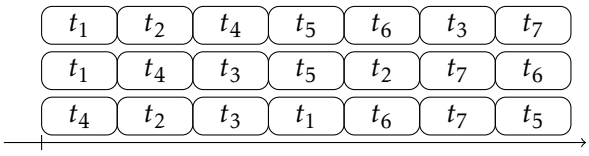
The schedules ($t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_6 \prec t_i \prec t_j$) and ($t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_6 \prec t_j \prec t_i$) are the only two optimal schedules, with a total distance of 54. They do not fulfill release date consistency since all the voters have completed tasks t_i and t_j at time 7 in their preferred schedules, whereas in the returned solution, one of these two tasks is completed at time 8. The best solutions fulfilling release date consistency are ($t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_j \prec t_i \prec t_6$), and ($t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_i \prec t_j \prec t_6$), with a total distance of 56. Therefore, the Distance Criterion rule does not fulfill deadline consistency.

By reversing the preferences (e.g. , when a preferred schedule ($t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_j \prec t_i \prec t_6$) becomes ($t_6 \prec t_i \prec t_j \prec t_5 \prec t_4 \prec t_3 \prec t_2 \prec t_1$)), we obtain an instance in which tasks t_i and t_j always start after or at t_1, but in which the optimal solutions are ($t_i \prec t_j \prec t_6 \prec t_5 \prec t_4 \prec t_3 \prec t_2 \prec t_1$) and ($t_j \prec t_i \prec t_6 \prec t_5 \prec t_4 \prec t_3 \prec t_2 \prec t_1$) (the previous optimal solutions but reversed). Either task t_i or t_j starts at time 0 in these solutions, whereas no voter schedule theses tasks before time 1. Therefore, the Distance Criterion rule does not fulfill release date consistency. □

Proposition 4.4.2: Binary - Deadline and Release Date Consistency

The Binary Criterion rule does not fulfill the deadline consistency nor the release date consistency.

Proof. Let us consider the following preferences of 3 voters over 7 tasks:



We first consider that the deadlines are the one given in the above schedules, and that the release dates are 0. The binary criterion this corresponds to the ΣU criterion (which minimizes the number of late tasks).

The only optimal solution for ΣU is ($t_1 \prec t_2 \prec t_3 \prec t_5 \prec t_6 \prec t_7 \prec t_4$). The number of late tasks in this solution is 3, task t_4 being considered late by the three voters. In a solution fulfilling the deadline consistency property, task t_4 has to be completed at most at time 3. This implies that either task t_1, task t_2 or task t_3 has to end after time

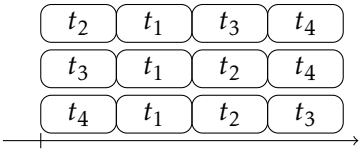
3 and will therefore be late for two voters. Additionally, if task t_1 is delayed, because of deadline consistency, it has to end at time 4, meaning that task t_5 has to be delayed and will therefore be considered late for 2 voters, which amounts to 4 late tasks, more than the optimum. The same line of reasoning can be applied for tasks t_2 and t_3 if they are delayed after time 3, causing delay to task t_5, t_6 or t_7 if they are scheduled at time 4, 5 or 6. Any solution respecting the deadline consistency property has therefore a number of late tasks of at least 4: the ΣU rule does not fulfill deadline consistency.

We can show similarly that the Binary Criterion rule does not fulfill the release date consistency. To this end, we consider that the the deadlines are n, and that the release dates are the one given in the above schedules, once they have been reversed. The binary criterion in this case minimizes the number of early tasks). □

Proposition 4.4.3: EMD- Deadline and Release Date Consistency

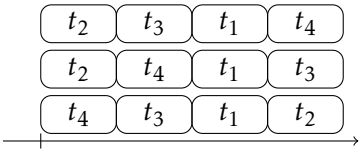
The EMD rule does not fulfill deadline consistency nor release date consistency.

Proof. Let us consider the following preferences of 3 voters over 4 tasks:



With such preferences, the median completion times are as follows: $m_1(P) = 2$, $m_2(P) = m_3(P) = 3$ and $m_4(P) = 4$. The EMD rule returns a schedule in which task t_1 is scheduled first and therefore completes at time 1, which is before its completion time in all the preferences of the voters. Therefore, the EMD rule does not fulfill release date consistency.

Let us now consider the following preferences of 3 voters over 4 tasks:



With such preferences, the median completion times are as follows: $m_1(P) = 3$, $m_2(P) = 1$, $m_3(P) = 2$ and $m_4(P) = 2$. The EMD rule returns a schedule in which task t_1 is scheduled last and therefore completes at time 4, which is after its completion time in all the preferences of the voters: the EMD rule does not fulfill deadline consistency. □

Since our three rules do no fulfill release date nor deadline consistency, we propose a weaker, yet meaningful, property called *temporal unanimity*.

Temporal unanimity

An aggregation rule satisfies temporal unanimity if, when all voters agree on the time interval during which a task t_i is scheduled, then t_i is scheduled during this time interval in the solution returned by the rule. When preferences are given as schedules, this property means that if all voters schedule task i at the same time slot in their preferred schedules, then i should be scheduled at the same time slot in the returned solution. When preferences are expressed as time intervals, it means that if all voters agree on the same release date and deadline for i, then i should be scheduled in this given interval in the returned solution.

Definition 4.4.3: Temporal Unanimity

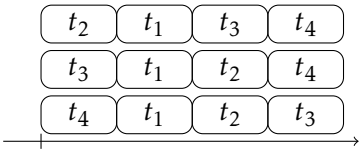
Let V be a set of voters, and let t_j be a task such that for each voter $v_i \in V$, we have $d_{j,i} = d$, with d a constant, and $r_{j,i} = r$, with r a constant strictly smaller than d. An aggregation rule fulfills *temporal unanimity* if it returns a schedule in which task t_j is executed between r and d.

We show that EMD does not fulfill this property, whereas the Binary and the Distance Criterion rules do fulfill this axiom.

Proposition 4.4.4: EMD- Temporal Unanimity

The EMD rule does not fulfill the temporal unanimity property.

Proof. Let us consider the following instance:



The median completion times are as follows: $m_1(P) = 2$, $m_2(P) = m_3(P) = 3$ and $m_4(P) = 4$. The EMD rule returns a schedule in which task t_1 is scheduled first and thus completes at time 1 even though it completed at time 2 in all the schedules expressed by the voters. □

Proposition 4.4.5: Binary - Temporal Unanimity

The Binary Criterion rule fulfills temporal unanimity.

Proof. Let us consider a task $s(1)$ such that, for all voter v_i, $d_{s(1),i} = d$ and $r_{s(1),i} = r$ with r and d two constants such that $0 \leq r < d \leq n$. Let us now consider an optimal schedule S^* for the Binary Criterion minimization in which task $s(1)$ is not scheduled between r and d. Since each of the preferences has to be compatible with a feasible schedule, in

each preferences there are at most $d-r$ tasks with release dates and deadlines included in the $[r,d]$ interval. Since task $s(1)$ is always included in this interval in the preferences of the voters, there is in the $[r,d]$ interval of S^* at least one task $s(2)$ is scheduled in the preferences of the voters at least once before r or after d. We distinguish two sub-cases.

- If this task $s(2)$ does not have a unique release date r' given by the voters and a unique due date d' given by the voters, then we can simply perform the swap between the positions of $s(1)$ and $s(2)$ to obtain a new solution S' in which there is one less task out of its unique time interval and which is at least as good as S^* since we decrease the binary criterion cost for $s(1)$ by the number of voters v and we increase it for $s(2)$ by at most v.

- If this task $s(2)$ has a unique release date r' and due date d' then we consider two subcases:

  - If task $s(2)$ is scheduled in S^* before r' or after d', and therefore not in its unique time interval, or if its time interval covers the position of $s(1)$ in S^*, we can perform the swap between the positions of $s(1)$ and $s(2)$ as in the above mentioned case.
  - Otherwise, we consider the other tasks, if any, scheduled in S^* between r and d and which are not always scheduled between r and d in the preferences. If none of these tasks fulfill any of the two previous conditions then we consider the set $\mathcal{T}_{s(1)}$ of all these tasks scheduled between r and d in S^* and which have a unique time interval in which they are scheduled. For each of these tasks, its time interval r', d' is either as $r' < r$ or $d' > d$, or both. We now consider the interval from the smallest unique release date of a task in $\mathcal{T}_{s(1)}$ to the maximum unique deadline of a task in $\mathcal{T}_{s(1)}$. We then repeat the same reasoning as above:
    * if there is a task $s(3)$ which is in the time interval of a task $s(2)$ from $\mathcal{T}_{s(1)}$, and which does not have a unique time interval or which has a time interval containing the position of $s(1)$ in S^*, then we perform the following circular exchange: task $s(1)$ takes the time slot of $s(2)$, which takes the time slot of $s(3)$, which takes the time slot of $s(1)$. Such a circular exchange does not increase the binary criterion, since the cost relative to $s(1)$ is decreased by v, the cost relative to $s(2)$ does not increase, since $s(2)$ stays in its interval, and the cost of $s(3)$ increases by at most v.
    * If no such task $s(3)$ exists, then there is at least one task which has a unique release date $r'' < r'$, or a unique deadline $d'' > d'$, or both. We then consider the set $\mathcal{T}_{s(2)}$ of such tasks and expand the considered interval. Since at each of these steps we extend the considered interval by at least one unit of time, the interval will necessarily include the position of $s(1)$ at some point and we will be able to perform a swap.

□

Proposition 4.4.6: Distance - Temporal Unanimity with ordered preference

The Distance Criterion rule fulfills temporal unanimity when preferences are schedules.

Proof. Let us consider that a task t_l is always scheduled between time k and time $k+1$ in the preferences of the voters. Let S^* be an optimal solution for the Distance Criterion minimization, and let us assume, by contradiction, that task t_l is not scheduled between k and $k+1$ in S^*. Let S be a schedule obtained from S^* by swapping the positions of task t_l and the task t_j scheduled between k and $k+1$ in S^*. Note that the distance of any task other than t_l or t_j is the same in S and S^*. The distance of task t_l is decreased by the absolute value of the difference between its position in S and its position in S^*, times the number of voters (since all voters scheduled it between k and $k+1$), while the distance of task t_j is increased by at most this value. Therefore S is an optimal schedule as well.

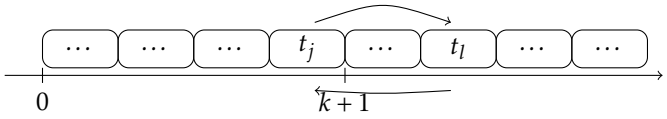


Figure 4.1: Schedule S^* and the swap performed to obtain S.

Let us now examine the case in which the distance of task t_j has increased by $v|C_l(S^*) - C_j(S^*)|$ – we will actually show that this cannot happen. Note that if task t_j was scheduled before task t_l in S^* (case 1) then the distance of t_j increased by $vC_l(S^*) - C_j(S^*)$: it means that t_j has been scheduled before its completion time in S^* by all voters. Likewise, if task t_j was scheduled after task t_l in S^* (case 2) then the distance of t_j increased by $C_j(S^*) - C_l(S^*)$: it means that t_j has been scheduled after its completion time in S^* by all the voters.

In case 1, let b be the maximum completion time of task t_j in the preference profile, and let t_k be the task which is completed at time t_b in S^*. We build schedule S' from S by swapping the position of task t_j and task t_k. The distance of t_j is decreased by the difference between the position of t_j and t_k for all voters. If the distance of t_k increases by the same value it means that task t_k always completes before b in the preferences of the voters. By repeating such swaps, the date b is decreased each time and we will necessarily reach a point where we either find a task for which the distance increase is smaller than the distance decrease when doing the swap or find a b of 1.

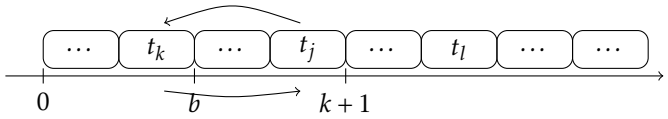


Figure 4.2: Schedule S^* and a preliminary swap (case 1) ensuring that the final swap of task t_l will strictly decrease the total distance.

The same thing can be done in case 2, by defining b as the minimum completion time of t_j in the preference profile, and t_k as the task which is completed at time b in S^*. The distance of t_j is decreased by the difference between the position of t_j and t_k for all voters. If the distance of t_k increases by the same value than the distance of t_j is decreased, it means that task t_k always completes after b in the preferences of the voters. By repeating such swaps, the date b is increased each time and we will necessarily reach a point where we either find a task for which the distance increase is smaller than the distance decrease when doing the swap or find a b of n.

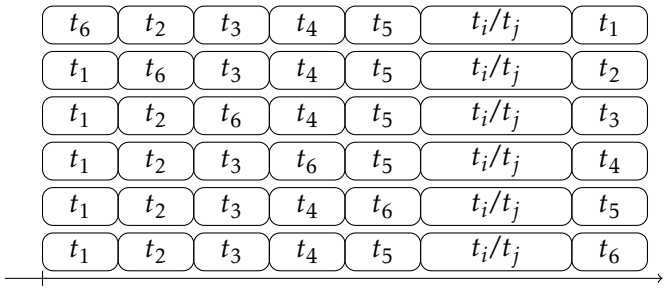
If we did not find $b = 1$, or $b = n$, it means that we have found a schedule of cost (sum of the distances) better than S^*, a contradiction. If we ended with $b = 1$ or $b = n$, then a task that would always complete before (resp. after) or at time $b = 1$ (resp. $b = n$) would always be scheduled first (resp. last), and doing the swap will always be strictly better. The solution obtained after doing the swaps is strictly better than the solution S^* supposed to be optimal, a contradiction.

□

Proposition 4.4.7: Distance - Temporal Unanimity with interval preferences

The Distance Criterion rule does not fulfill temporal unanimity when preferences are expressed as release dates and deadlines.

Proof. Let us consider an instance with 8 tasks and 6 voters and the following preferences:



In this profile each voter gives a time interval of 1 for all tasks except for t_i and t_j which have a time interval of 2. For example the first voter indicates that task t_6 has a release date of 0 and a due date of 1, while both tasks t_i and t_j have a release date of 5 and a due date of 7. The two optimal solutions for the Distance Criterion minimization

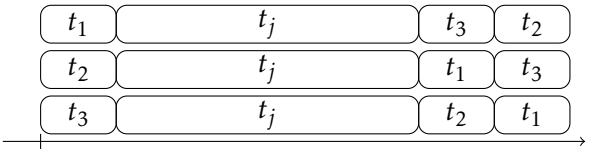
are $(t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_6 \prec t_i \prec t_j)$ and $(t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_6 \prec t_j \prec t_i)$, with a total distance of 48. These optimal schedule do not fulfill temporal unanimity since either t_i or t_j is scheduled outside of the time interval agreed on by all the voters. The best solutions fulfilling temporal unanimity are $(t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_j \prec t_i \prec t_6)$, and $(t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_i \prec t_j \prec t_6)$, with a total distance of 50. □

When all tasks have the same length and the preferences are expressed as schedules, the ΣD rule fulfills temporal unanimity. We show that this is however not the case when tasks do not have the same length.

Proposition 4.4.8: ΣD- Temporal axioms with tasks of different length

When tasks are not of the same length, the ΣD rule does not fulfill temporal unanimity nor release date consistency nor deadline consistency.

Proof. We consider the following instance with 4 tasks, 3 voters and such that $p_1 = p_2 = p_3 = 1$ and $p_j = 4$:



Any optimal solution for ΣD schedule t_j first and then t_1, t_2 and t_3 in any order, for a total deviation of 18. A solution fulfilling temporal unanimity would schedule t_1, t_2 or t_3 first, then t_j and finally the remaining two tasks, in any order. Such a solution has a deviation of 24. We also note that the optimal solution does not fulfill the deadline consistency, and by reversing the instance, we can show that the ΣD rule does not fulfill release date consistency either.

□

4.5 Precedence constraints

In this section, we focus on precedence constraints between the tasks. We will consider two settings.

Firstly, we consider a setting in which the precedence constraints are known by the voters. In this setting, that we call *inferred precedences*, if a task t_a has to be scheduled before a task t_b, then, in the preference S_i of any voter v_i, we have $C_a(S_i) < C_b(S_i)$. Our aim is to determine whether or not a given aggregation rule guarantees that task t_a will be scheduled before task t_b in the schedule returned by the rule. Note that in voting theory this property is called *unanimity*.

The second setting corresponds to the case in which the precedence constraints are not known by the voters, and therefore preferences do not necessarily fulfill these

precedence constraints: the precedence constraints only exist for the schedule that has to be returned. This setting is called *precedence graph*.

We define a family of optimization problems of form $\alpha - Prec$ where α is an optimization criterion and $Prec$ is a setting for the precedence constraints: it is Inferred when precedence constraints are fulfilled by the preferences, or Graph when the preference constraints only apply to the returned solution. For example, the problem ΣD-graph has the following input: a set $\mathcal{J}$ of n tasks ; an acyclic directed graph G which represents the precedence constraints between the tasks in $\mathcal{J}$; a set V of v preferred schedules (permutation of tasks) – these schedules do not necessarily fulfill the precedence constraints. The aim is to output a schedule which fulfills the precedence constraints and, among these feasible schedules, which minimizes the sum of the deviations with respect to the preferences of the voters: $\sum_{v_i \in V} \sum_{t_j \in \mathcal{J}} D_j(S, S_i)$.

Our aim is to study the complexity of problems mentioned before (total deviation, total tardiness, number of late tasks), when there are inferred or given precedence constraints. In Section 4.5.1, we study the case in which precedence constraints are inferred by the preferences of the voters, and we show that problems ΣD-inferred and ΣT-inferred can be solved in polynomial time. In Section 4.5.2, we study the case in which precedence constraints are given and are not necessarily fulfilled by the preferred schedules of the voters. We will show that problems ΣD-graph, ΣT-graph and ΣU-graph are NP-hard.

4.5.1 Inferred precedence constraints

Proposition 4.5.1: ΣT-inferred and ΣD-inferred- Polynomially solvable

Problems ΣD-inferred and ΣT-inferred can be solved in $O(vn^2 + n^3)$.

Proof. We showed earlier (Propositions 3.3.19 and 3.3.20) that when two tasks t_a and t_b are of same length, if task t_a is scheduled before t_b in all the preferences then there exists an optimal solution for the ΣD rule and the ΣT rule such that t_a is scheduled before t_b. Additionally, for any optimal solution in which t_b would be scheduled before t_a, it is possible to swap the position of t_a and t_b without increasing the deviation (or the tardiness). Therefore by doing successive permutations from an optimal solution, we can find another optimal solution in which precedence constraints are fulfilled. We now show that the number of permutations needed is bounded by n^2. Problems ΣD-inferred and ΣT-inferred can thus be solved in polynomial time by

1. computing an optimal solution of ΣD or ΣT (without the precedence constraints) through an assignment problem, as seen in the introduction. This can been done in $O(vn^2 + n^3)$;

2. swapping pair of tasks (t_a, t_b) that do not fulfill precedence constraints in the returned schedule of Step 1. As we will show now, there will be at most n^2 swaps.

We create a precedence directed graph with n vertices, one for each task, and in which there is an edge from vertex t_a to vertex t_b if the task corresponding to vertex t_a is always scheduled before the task corresponding to vertex t_b in the preferences of the voters. There are at most n^2 edges so it is possible to create this graph in $O(n^2)$ operations. Note that this precedence relation is transitive: if t_a is always scheduled before t_b and t_b is always scheduled before t_c then t_a is always scheduled before t_c. This implies that this graph has no cycle. This also implies that there exists at least one vertex with no successor.

We choose a vertex x among the vertices with no successor in the above mentioned precedence graph. For readability, we will in the sequel denote the task corresponding to vertex x as task x. We look whether among the predecessors of x there exist vertices corresponding to a task scheduled after x in the optimal schedule returned by ΣD or ΣT. If such vertices exist, we swap the position of the task x with the task corresponding to its predecessor scheduled after it and as close as possible to x in the returned schedule. We repeat this step until all the tasks corresponding to predecessors of x are scheduled before x. By swapping x with its closest predecessor scheduled after it, we make sure that we do not create any violation of the precedence constraints. Studying all the vertices takes n operations, consisting in at most n swaps: the total number of swaps is then bounded by n^2. □

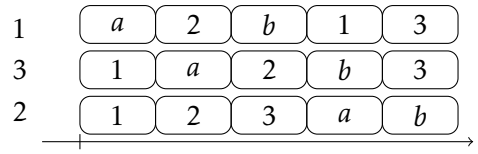
Note that the previous proof is a constructive proof: we can compute an optimal solution for ΣD-inferred (or ΣT-inferred) by solving an assignment problem for ΣD (or ΣT), and then swapping tasks which do not fulfill the precedence constraints as explained in the proof of Proposition 4.5.1.

We cannot take the same approach for ΣU-inferred. Indeed, there are instances in which no optimal solution for the minimization of the total number of late tasks criterion fulfills the inferred precedence constraints, as shown by the following proposition.

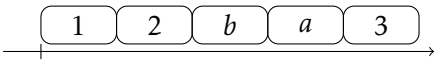
Proposition 4.5.2: ΣU- Precedence not fulfilled by optimal solution

There exist instances for which no optimal solution for the ΣU criterion fulfills the inferred precedence constraints.

Proof. Let us consider the following instance of 5 tasks and 6 voters. The number at the left of each schedule indicates the number of voters whose schedule is the preferred schedule (e.g. the favorite schedule of three voters is the second schedule).



The only optimal solution for the ΣU criterion is the following one:



In this solution, b is scheduled before a, whereas all the voters have scheduled a before b in their favorite schedules: this violates the inferred precedence constraints. □

This last proposition means that we cannot proceed like in Proposition 4.5.1, by computing an optimal solution for ΣU and then swapping tasks which would not be in the right order. Whether problem ΣU-INFERRED is NP-hard or not is an open question.

4.5.2 Imposed precedence graph

We start by proving that this problem is strongly NP-hard for the total tardiness criterion.

Proposition 4.5.3: ΣT-GRAPH- NP-hardness

The ΣT-GRAPH problem is strongly NP-hard, even when the precedence graph consists in chains.

We prove this proposition by doing a polynomial time reduction from the scheduling problem denoted by $(1|chains, p_j = 1|\Sigma T_j)$ using the Graham's notation, a classical way to denote problems in scheduling theory [Brucker, 2010]. An instance of this problem is:

- a set J of n unit tasks, each task j having a due date d_j. Without loss of generality, we assume that $d_j \leq n$ for all j.
- a precedence graph, modeling precedence constraints between the tasks. We assume that this graph is made of chains (i.e. each task has at most one successor and one predecessor, and there is no cycle).

The optimization version of this problem consists in minimizing the sum of the tardiness of the tasks. The decision version of this problem consists in answering the following question: given an integer K, is there a schedule S of the tasks in J on a single machine, such that the precedence constraints are fulfilled, and such that the total tardiness of the tasks, $\sum_{j\in J} \max(0, C_j(S) - d_j)$, is smaller than or equal to K ? This problem is known to be NP-hard [Leung and Young, 1990].

We create an instance of ΣD-GRAPH from the instance from $(1|chains, p_j = 1|\Sigma T_j)$ as follows.

- For each task j in J we create a task t_j and a task dum_j. These tasks are split into two sets $J_t = \{j_1, \ldots, j_n\}$ and $J_{dum} = \{dum_1, \ldots, dum_n\}$. The set J' of the tasks of the instance of ΣT-GRAPH is the union of J_t and J_{dum}.

- For each precedence relation in the $(1|chains, p_j = 1|\Sigma T_j)$ problem between tasks i and j, we create a precedence constraint between t_i and t_j in the precedence graph of problem ΣT-GRAPH.

- For each task j in J we also create three voters. Their preferred schedules, that we will describe now, are represented on Figure 4.3. The first two voters, of type T, schedule t_j first, followed by t_{j+1} and so forth until t_n, then t_1 to t_{j-1} by increasing index. They then schedule the dum tasks following the same pattern: dum_j first, then dum_{j+1} to dum_n, followed by dum_1 to dum_{j-1} by increasing index (see top schedule in Figure 4.3). The last voter, of type D, schedules task t_j between time $d_j - 1$ and d_j. Before that, she schedules $(d_j - 1)$ dum tasks from dum_j by increasing index (using again a circular order of the tasks, where task dum_1 follows task dum_n). The remaining dum tasks are scheduled after t_j by increasing indexes. The schedule is completed with tasks $t_{j+1}, t_{j+2}, \ldots$, until task t_{j-1} if $j \neq 1$, or task t_n if $j = 1$ (see bottom schedule in Figure 4.3).

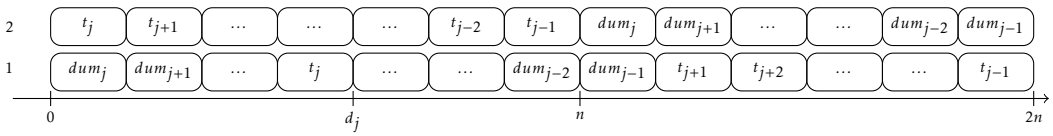


Figure 4.3: Preferred schedules of the 3 voters generated for task j.

In order to prove Proposition 4.5.3, we start by proving the following lemma.

Lemma 4.5.1: Structure of an optimal solution in the reduction

For the instance of the ΣT-GRAPH problem described above, there is an optimal solution in which all the t tasks are scheduled before all the dum tasks.

Proof. Let us assume by contradiction that there is no optimal solution in which all t tasks are scheduled before all dum tasks. Let S be such an optimal solution: there is at least one dum task completing just before a task t. Let us call dum_i the first dum task scheduled before a t task, and t_j be the t task scheduled just after dum_i. Let k be the completion time of dum_i in S: we have $C_{dum_i}(S) = k$ and $C_{t_j}(S) = k + 1$ (note that $1 \leq k < 2n$).

We call S' the schedule obtained from S by swapping the position of dum_i and t_j. The total tardiness of S' is similar to S except for the tardiness of dum_i and t_j. We then have $C_{dum_i}(S') = k + 1$ and $C_{t_j}(S') = k$. Note that if the precedence constraints over the t tasks are satisfied by S, they are also satisfied by S' since the order on the t tasks has not changed. Therefore, since S is a feasible solution, S' is also a feasible solution. We distinguish two sub-cases:

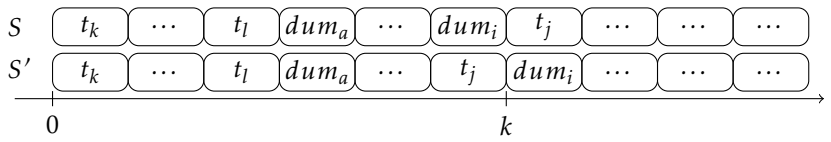


Figure 4.4: Schedules S and S'. The first dum task to be scheduled just before a t task in S is dum_i.

- $k \leq n$. In this case, the tardiness relative to task t_j is reduced in S' in comparison to S by at least $2k$. Indeed, there are $2k$ voters scheduling t_j at time k or before. Therefore, moving t_j from $k+1$ to k reduces the tardiness of t_j by one for each of these voters, giving a total of $2k$. The tardiness of task dum_i is increased in S' in comparison to S. There are at most k voters of type D scheduling dum_i at time k or before: for each of these voters, the tardiness is increased by one. The sum of the tardiness of S' is decreased by at least $2k$ (due to t_j), and increased by at most k (due to dum_i), in comparison to the sum of the tardiness of S: the total tardiness of S' is thus smaller than the tardiness of S. Since S minimizes the sum of the tardiness, there is a contradiction.

- $k > n$. In this case, the tardiness relative to task t_j is reduced in S' in comparison to S by $2n+1+(k-n)$. Indeed, there are $2n$ voters of type T scheduling t_j before k, and one voter of type D scheduling t_j so that this task is completed at date d_j with $d_j \leq n$. This makes a total of $2n+1$. Additionally, there are $k-n$ voters of type D scheduling t_j at time $n+1$, $n+2$ up to k. For each of these voters, the tardiness of t_j is reduced by one.

  On the other hand, the tardiness of dum_i is increased in S' in comparison to S by $n+2(k-n)$. The n voters of type D scheduled dum_i so it that it is completed at most at time $n+1$, meaning that delaying dum_i from k to $k+1$ increases the tardiness by one for each of these n voters. Additionally, there are $2(k-n)$ voters of type T scheduling dum_i so that it completes at dates $n+1$, $n+2$ to k: the tardiness is increased by one for each of these voters. If we compare the increase in tardiness for task dum_i, $n+2(k-n)$, to the decrease of the tardiness for task t_j, $2n+1+(k-n)$, we see that the sum of the tardiness in S' is decreased by $2n+1-k$. Since $k < 2n$, this value is always strictly positive. This means that the total tardiness of S' is strictly smaller than the tardiness of S, an optimal solution: a contradiction.

□

We can now start the proof of Proposition 4.5.3.

Proof. From Lemma 4.5.1, we know that there exists an optimal schedule S in which t tasks are scheduled before dum tasks. We analyze the sum of the tardiness in such a schedule. We first show that the sum of the tardiness of dum tasks is the same in any schedule fulfilling the property of Lemma 4.5.1 (first item below), and we then analyze the sum of the tardiness due to t tasks (second item below).

- We show that in any schedule in which dum tasks are scheduled after t tasks, the tardiness due to dum tasks is always the same.

  Voters of type T schedule each dum task twice between n and $n+1$, twice between $n+1$ and $n+2$ and so on until $2n-1$ and $2n$. In schedule S, the dum task scheduled between n and $n+1$ in not late for any voter of type T, the task scheduled between $n+1$ and $n+2$ is late of one unit of time for 2 voters of type T, and so on. Overall, the total tardiness of dum tasks for T voters is then $2\sum_{i=1}^{n}\sum_{j=1}^{i}(j-1)$, a constant number.

  Let us now show that the sum of the tardiness of dum tasks for D voters will be the same in any schedule S in which t tasks are scheduled before dum tasks. Indeed, for each D voter j, and for each task dum_i, the completion time of dum_i in the preferred schedule of j is at most $n+1$, whereas the completion time of dum_i in S is at least $n+1$. Therefore, the sum of the tardiness due to dum tasks for D voters is equal to the sum of the distances between completion times of dum tasks in the preferred schedules of voters D to date $n+1$ – which is a constant, since preferred schedules are fixed –, plus the sum of the distances of dum tasks between date $n+1$ and the completion time of dum tasks in S – this is also a constant since the completion times of dum tasks in S are the set of times $\{n+1,\ldots,2n\}$. Therefore, the the sum of the tardiness of dum tasks for D voters is a constant.

  We have seen that the sum of the tardiness of dum tasks is value is the same for any schedule S which fulfills Lemma 4.5.1. Let T_{dum} denote this value, which is constant.

- Regarding tasks t, voters of type T schedule them such that each task t_i is completed twice at time 1, twice at time 2 and so on. So, regardless of the order of tasks t in S, the first task of S is not late for any voter, the second task of S is late by 1 unit of time for 2 voters, the third task is late by 1 unit of time for 2 voters, by 2 units of time for two voters and so on. Therefore, the sum of the tardiness of t tasks for voters of type T is also the same for each schedule S in which t tasks precede dum tasks. Let T_t denote this sum of tardiness.

  Voters D schedule all tasks t after $n+1$ except one task t_j (for the j-th voter of type D), and this task is completed at time d_j. Therefore in S, each task t_j is always early for all voters D except one, the j-th voter of type D, and its tardiness for this voter is equal to $\max(0, C_{t_j}(S) - d_j)$.

The sum of the tardiness $T(S)$ in schedule S is thus equal to:

$$T(S) = T_{dum} + T_t + \sum_{t_j \in J_t} \max(0, C_{t_j}(S) - d_j)$$

Since T_{dum} and T_t do not depend on the order of the tasks in S as long as all tasks t are scheduled first and all tasks dum are scheduled afterwards, the tardiness of schedule S only depends on the position of tasks t relatively to the due dates of the $(1|chains, p_j = 1|\Sigma T_j)$ problem.

We will now prove that there exists a solution S for the instance of the ΣT-GRAPH problem described above such that $T(S) \leq T_{dum} + T_t + K$, if and only if there exists a schedule S' for $(1|chains, p_j = 1|\Sigma T_j)$ problem such that the tardiness is smaller than or equal to K. In other words, the answer to the question of ΣT-GRAPH problem is then "yes" if and only if the answer to the question of the corresponding instance of $(1|chains, p_j = 1|\Sigma T_j)$ is "yes".

Let us assume first that there is a solution S of ΣT-GRAPH problem such that $T(S) \leq T_{dum} + T_t + K$. It means that $\sum_{t_j \in J} \max(0, C_{t_j}(S) - d_j) \leq K$. Let S' be a schedule of tasks of $(1|chains, p_j = 1|\Sigma T_j)$ such that the completion time of task j in S' is equal to the completion time of t_j in S. We have $\sum_{j \in J} \max(0, C_j(S') - d_j) \leq K$, and this solution is feasible since the precedence constraints between the tasks of the $(1|chains, p_j = 1|\Sigma T_j)$ problem are the same than between the t tasks. The answer to the question of the $(1|chains, p_j = 1|\Sigma T_j)$ is then "yes".

Let us now assume that there is a feasible solution (schedule) S' of $(1|chains, p_j = 1|\Sigma T_j)$ such that the total tardiness is smaller than or equal to K. If we create solution S such that the completion time of task t_j in S is equal to the completion time of j in S', we then have $\sum_{t_j \in J_t} \max(0, C_{t_j}(S) - d_j) \leq K$. The dum tasks are then scheduled in any order. Such a solution has then a total tardiness of $T_t + T_{dum} + K$. This solution is feasible since the precedence constraints between tasks of the $(1|chains, p_j = 1|\Sigma T_j)$ problem are the same than between the t tasks. This implies that the answer to the ΣT-GRAPH problem is thus "yes".

There is a polynomial time reduction from decision problem $(1|chains, p_j = 1|\Sigma T_j)$, which is strongly NP-complete, to the decision version of our problem ΣT-GRAPH. Problem ΣT-GRAPH is thus strongly NP-hard. □

Since, as we have seen before, with unit tasks graphs, and for any profile P and any schedule S, the sum of the deviations in S with respect to profile P is equal to twice the sum of the tardiness in S, a schedule minimizing the sum of the deviations among schedules which fulfill the precedence constraints will also minimize the sum of the tardiness. Given Proposition 4.5.3, we deduce the following corollary.

Corollary 4.5.1: ΣD-GRAPH- NP-hardness

The ΣD-GRAPH problem is strongly NP-hard, even when the precedence graph consists in chains.

We now show that problem ΣU-GRAPH, which aims at minimizing the number of late tasks in the returned schedule, with respect to the preferred schedules of the voters, is also a strongly NP-hard problem.

Proposition 4.5.4: ΣU- NP-hardness

The ΣU-GRAPH problem is strongly NP-hard, even when the precedence graph only consists in chains.

We prove this results by doing a polynomial time reduction from the $(1|chains, p_j = 1|\Sigma U_j)$ problem. The decision version of this problem is the following one. An instance of this problem is:

- A set $J' = \{1, \ldots, n\}$ of n unit tasks. Each task i has a deadline d_i.
- A a acyclic precedence graph of n vertices $\{1, \ldots, n\}$: there is one edge from vertex i to vertex j if task i has to be scheduled before task j. This graph can be only a set of chains between some tasks.
- An integer K'

The aim of optimization problem is to compute a schedule which fulfills the precedence constraints and which minimizes the number of late tasks (i.e. tasks which are completed after their deadlines). The question of the corresponding decision problem is the following one: is there a schedule S which fulfills the precedence constraints and in which at most K' tasks are late ?

Garey and Johnson [1976] have shown that this problem is strongly NP-hard with general precedence constraints, even with unit time tasks. Lenstra and Rinnooy Kan [1980] have sharpened this result by showing that this problem remains strongly NP-hard, even if the set of precedence constraints is a set of chains.

Without loss of generality we assume that $d_i \leq n$ (tasks with deadlines larger than n will never be late in a schedule of n unit tasks without idle time). We create an instance of ΣU-GRAPH as follows.

For each task i of J', we create a task t_i and a task dum_i. For each task i we also create $(n+1)$ voters as shown in Figure 4.5. There are n voters "of type T" scheduling task t_i first, then t_{i+1} and so forth until t_n and then scheduling tasks t_1 to t_{i-1} by increasing index. They then schedule tasks dum_1, dum_2, $\ldots dum_n$. The last voter, "of type D" schedules task t_i so that it is completed at time d_i, and, if $d_i \neq 1$, she schedules task dum_2 to dum_{d_i-2} by increasing index from time 0 to time $d_i - 2$. From time d_i, she schedules tasks dum_{d_i-1} to dum_n until time n, by increasing index, and she schedules dum_1 so that this tasks is completed at time $n+1$. She completes the schedule with tasks $t_{i+1}, \ldots, t_n$ by increasing index, followed by tasks t_1 to t_{i-1} by increasing index. For any precedence relation between tasks i and j in $(1|chains, p_j = 1|\Sigma U_j)$, we create the same preference relation between tasks t_i and t_j of our ΣU-GRAPH instance.

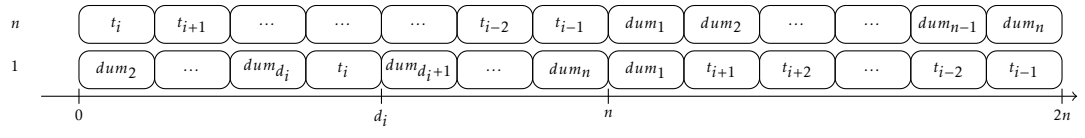


Figure 4.5: Preferred schedules of the $n+1$ voters generated for task i.

In order to prove Proposition 4.5.4, we introduce several lemmas which describe an optimal schedule for the above described instance. As in the proof of Proposi-

tion 4.5.3, we will see that computing such an optimal solution allow us the associated NP-complete scheduling problem (problem $(1|chains, p_j = 1|\Sigma U_j)$ in our case).

Lemma 4.5.2: Task dum_1 always starts in n

There exists an optimal solution for ΣU-GRAPH in which task dum_1 completes at time $n+1$.

Proof. All voters schedule dum_1 so that it is completed at time $n+1$. Let S be a schedule in which dum_i does not complete at time $n+1$. We distinguish two sub-cases:

1. Task dum_1 completes before time $n+1$: we create schedule S' from S by scheduling dum_i so that it is completed at time $n+1$. We decrease from 1 unit of time any task scheduled in S between dum_1 and time $n+1$. Task dum_1 is not late in S' for any voter, just like in S and the task that have been scheduled before cannot become late in S' if they were not in S. Therefore the number of late tasks cannot increase from S to S'.

2. Task dum_1 completes after $n+1$. We distinguish two sub-cases:

   - If the task j completing at time $n+1$ in S is a dum task, we create S' from S by swapping the position of dum_1 with the task j. The unit time penalty for all tasks but j and dum_1 are identical between S and S'. Task dum_1 is in S' on time for the $n(n+1) = n^2 + n$ voters, whereas it was late in S. On the other hand the unit time cost for task j is increased, but at most by n^2 voters, since the n voters of type D already considered it late since they scheduled it before time $n+1$. Overall the unit time penalty is reduced in S' in comparison to S.

   - If the task j completing at time $n+1$ in S is a t task, we create a new schedule S' by scheduling dum_1 so that it completes at time $n+1$. We then perform consecutive swaps such that the order on the t tasks is the same in S, which is a feasible solution, and S'. If there is at least one t task scheduled between $n+1$ and, $C_{dum_1}(S)$, the completion time of dum_1 in S, we schedule task j at the time slot occupied by the first t task scheduled after $n+1$ in S. Let t_i be such a task. This task t_i is then scheduled at the time slot of the following t task which is completed before $C_{dum_1}(S)$, and so on until there is no t task left before $C_{dum_1}(S)$. The final t task moved that way goes on the time slot occupied by dum_1 in S (i.e. is completed at time $C_{dum_1}(S)$).

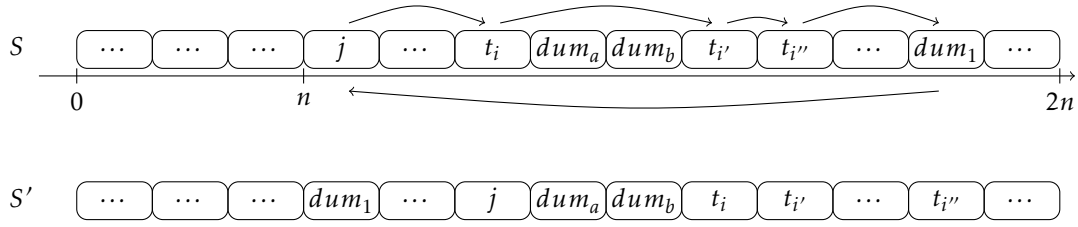


Figure 4.6: Schedule S and the swaps performed to obtain S'.

Note that if the precedence constraints between the t tasks are fulfilled by S, they are also fulfilled by S' since the order on the t tasks do not change, just their positions.

The t tasks which have been moved in S' were considered late in S by all T voters: delaying them do not increase unit time penalty for T voters. Since D voters schedule t task in a cyclic fashion, each t task completes once at time $n+2$, once at time $n+3$ and so on. Therefore delaying a t task by one unit of time between $n+1$ and $2n$ increases its unit time penalty by 1 (since one additional voter will consider it late). Therefore, when delaying these tasks, the cumulative delay is at most n. On the other hand, scheduling dum_1 at time $n+1$ decreases the number of late tasks by n^2+n since it is late for all voters in S and on time for all voters in S'. This means that the total unit time penalty is smaller in S' than in S.

In all the cases, we managed to generate a solution S' in which dum_1 is scheduled between time n and time $n+1$ with the total number of late tasks of S' smaller than or equal to the number of late tasks in S. Therefore there always exist an optimal solution in which dum_1 is scheduled between n and $n+1$.

□

Lemma 4.5.3: *dum* tasks starting from n are ordered by increasing indices

There exists an optimal solution of ΣU-GRAPH that fulfills Lemma 4.5.2 and such that there is in this solution a set of successive dum tasks scheduled by increasing index from time n, and none of these tasks are considered late by any voter of type T.

Proof. Let S be an optimal solution fulfilling the property of Lemma 4.5.2: task dum_1 is completed at time $n+1$.

Let us assume that in S some tasks of the dum set starting at time n are not scheduled by increasing index. Let dum_a and dum_b be the two tasks scheduled the earliest in this set and such that dum_a is scheduled before dum_b with $a > b$. Since they are the first two tasks fulfilling this condition any task of this set scheduled before dum_a in S has a smaller index than a and is also scheduled before dum_a in the preferences of voters T.

Let us consider the solution S' obtained from S by swapping the positions of dum_a and dum_b. Since $b < a$, dum_b is scheduled before dum_a in the preferences of T voters and since all tasks of the set scheduled before dum_a in S are also scheduled before dum_a in preferences of voters T, dum_a cannot be late for voters of type T. This means that task dum_a does not become late for T voters in S'. This tasks is late for D voters in both S and S' since it is scheduled after $n+1$. Since the completion time of dum_b is reduced in S' in comparison to S, it cannot be late in S' whereas it was not late in S. Therefore, the number of late tasks in S' is not larger than the number of mate tasks in S.

Repeating these swaps until all the dum tasks of the set are scheduled by increasing index, we obtain a new solution in which all of these tasks are on time for T voters and in scheduled by increasing index and such that number of late tasks is not increased in comparison to S. □

Lemma 4.5.4: Tasks t are scheduled before tasks dum on $[0,n]$

There exists an optimal solution for ΣU-GRAPH which fulfills Lemma 4.5.3, and such that all t tasks scheduled between time 0 and time n are scheduled before any dum tasks scheduled between time 0 and time n.

Proof. Let us consider a solution S fulfilling the properties of Lemmas 4.5.2 and 4.5.3 and such that there is a task dum_i scheduled between time 0 and time n and such that there is a task t_j scheduled just after dum_i in S. Since the task scheduled between time n and $n+1$ is dum_1 in S, task t_j completes at most at time n.

We create a schedule S' from S by swapping the positions of dum_i and t_j. For each date k between 1 and n, there are n voters of type T scheduling t_j so that it is completed at time k. Therefore, advancing t_j by one unit of time between 1 and n, decreases the number of late tasks by n. On the other hand, task dum_i is delayed by one unit of time. This does not impact the T voters since they schedule dum_i after time $n+1$. Voters of type D might have an increased unit time penalty for task dum_i. Since there are n voters of type D, this increases the number of late tasks by at most n. Therefore, the number of late tasks in S' is smaller than or equal to the the number of late tasks in S. □

Lemma 4.5.5: Structure of an optimal solution

There exists an optimal solution for ΣU-GRAPH which fulfills Lemma 4.5.4, and in which all the t tasks are scheduled before all dum tasks. Moreover, in this solution, the dum tasks are scheduled in order of increasing indexes.

Proof. Let S be an optimal solution satisfying the properties of Lemma 4.5.4 and such that all tasks t are not scheduled before all dum tasks. Let dum_i be the first dum task to be scheduled in S. This implies that $C_{dum_i} \leq n$. Because of Lemma 4.5.4, task dum_i has to be scheduled after a series of t tasks, and all tasks scheduled after dum_i and before dum_1 are dum tasks as well. Let t_j be the first t task scheduled after dum_i. As we have

seen, dum_i is scheduled after $n+1$ and after a set of dum tasks scheduled by increasing index.

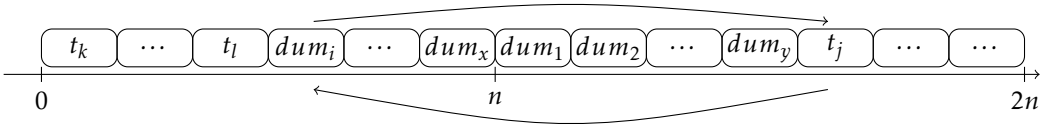


Figure 4.7: Schedule S with the swap performed to obtain S_{tmp}.

Let S_{tmp} be the schedule obtained by swapping from S the position of dum_i and t_j, and let S' be the schedule obtained by swapping from S the position of dum_i and t_j and in which dum_i is re-positioned in the dum set so that the tasks in the set are scheduled by increasing indexes (therefore that S' can also be obtained from S_{tmp} by repositioning dum_i at the right place in the set of dum tasks that follow it in S).

Note that since S fulfills the precedence constraints on the t tasks, then they are fulfilled by S_{tmp} and S' as well since the order on the t tasks does not change. In its new position in S_{tmp} and S', task dum_i is not late for voters of type T (who schedule dum_i after time n). It may be late for some voters of type D whereas it was not necessarily late for these voters in schedule S. Therefore, since there are n voters of type D, the number of late tasks due to dum_i is increased by at most n in S_{tmp} and in S'. Let us now focus on the total number of late tasks tardiness in S_{tmp}. The only tasks whose time slot has changed (compared to is time slot in S) is t_j, which was late for all voters of type T in S but now completes at most at time n. It thus in S_{tmp} on time for at least n voters of type T (the ones scheduling it between time $n-1$ and n). Overall the number of late tasks does not increase in S_{tmp} in comparison to S. Since all the voters schedule, in their preferred schedules, the dum tasks by increasing order, the number of late tasks in S' is not larger than the number of late tasks in S_{tmp}. Therefore, the number of late tasks does not increase in S' compared to in S.

By repeating, if needed, this type of swaps, we obtain an optimal solution in which all t tasks are scheduled before all dum tasks, and in which dum tasks are scheduled by increasing indices. □

Starting from any optimal solution S and applying the successive swaps described in Lemmas 4.5.2 to 4.5.5, we obtain an optimal solution in which tasks t are scheduled first and are followed by dum tasks which are scheduled between time n and $2n$ by increasing indices. Let us now prove Proposition 4.5.4.

Proof. We show that there exists a solution with a total number of late task smaller than or equal to K' for $(1|chains, p_j = 1|\Sigma U_j)$ if and only if there exists a solution with a total number of late tasks for ΣU-GRAPH smaller than or equal to $K = K' + n(n+1) + \sum_{i=1}^{n}(i-1)n$.

Let us first assume that there exists a solution with at most K late tasks for ΣU-GRAPH. Thanks to Lemmas 4.5.5, we know that there exists an optimal solution in

which tasks of type t are scheduled before dum tasks, which are scheduled by increasing indices. In such a solution S, the number of late tasks can be split into two parts, one independent from the order of the t tasks, and one depending on this order.

Regardless of the order of the t tasks, the dum tasks are all on time for the voters of type T, and all (except dum_1) late for the voters of type D. There are therefore $n-1$ dum tasks late for each of the n voters of type D, which amounts to $n(n-1)$ late tasks. Furthermore, t task completes n times at time 1, n times at time 2, and so on until time n. The t task completing at time 1 will be on time for all voters of type T, the t task completing at time 2 will be late for n voters of type T, the third task will be late for $2n$ voters and so on. This amounts to $\sum_{i=1}^{n}(i-1)n$. For each $i \in \{1,\dots,n\}$, task t_i is on time for D voters, except for the i-th D voter, who scheduled task t_i so that it is completed at time d_i.

This means that the total number of late task in S is $U_t(S) + n(n-1) + \sum_{i=1}^{n}(i-1)n$, where $U_t(S)$ denotes the number of late t tasks in S for voters of type D. Since S is an optimal solution and since the answer to the ΣU-GRAPH problem is 'yes', this means that $U_t(S) \leq K'$.

We label the t tasks according to their position in schedule S, which can be described as follows: $(t_{S(1)}, t_{S(2)}, \dots, t_{S(n)}, dum_1, dum_2, \dots, dum_n)$, where $S(i)$ denotes the index of the task scheduled in position i in S. We consider the schedule S' of tasks of $(1|chains, p_j = 1|\Sigma U_j)$: $S(1), S(2), \dots, S(n)$. Note that since S is a feasible solution of ΣU-GRAPH and since the precedence constraints on t tasks are the same than on the tasks of the $(1|chains, p_j = 1|\Sigma U_j)$ instance, S' is a feasible solution of $(1|chains, p_j = 1|\Sigma U_j)$. In S', task $S(i)$ is completed at the same time than task $t_{S(i)}$ in S, therefore task $S(i)$ is late if and only if $S(i)$ is late for the voter scheduling $S(i)$ at time $d_{S(i)}$. Therefore if $U_t(S) \leq K'$, the total number of late tasks in S' is also smaller than or equal to K', which means that the answer to the $(1|chains, p_j = 1|\Sigma U_j)$ problem is also 'yes'.

Reciprocally, if the answer to the $(1|chains, p_j = 1|\Sigma U_j)$ problem is 'yes', then there exists a schedule S' for $(1|chains, p_j = 1|\Sigma U_j)$ such that the total number of late tasks in S' is smaller than or equal to K'. We consider S the schedule $(t_{S'(1)}, \dots, t_{S'(n)}, dum_1, dum_2, \dots, dum_n)$ for the ΣU-GRAPH problem. Schedule S fulfills the precedence constraints of the ΣU-GRAPH instance since these precedence constraints are the same than the precedence constraints on the t tasks of the corresponding instance of ΣU-GRAPH.

Since S fulfills the property of Lemma 4.5.5, there is a constant number of late task $n(n-1) + \sum_{i=1}^{n}$ for voters of type T. The number of late t tasks for voters of type D depends on whether task t_i is scheduled before or after time d_i since only one D voter schedules task t_i before time n (she schedules t_i between times $d_i - 1$ and d_i). Task t_i is completed in S at the same time than task i in S'. Therefore task t_i completes after d_i in S if and only if task i is late in S'. Therefore the number of late t tasks in S for voters of type D is equal to the number of late tasks in S'. Since the number of late tasks in S' is smaller than or equal to K', the total number of late tasks in S is smaller than or equal to $K' + n(n+1) + \sum_{i=1}^{n}(i-1)n$ and the answer to ΣU-GRAPH is then 'yes'.

The answer to the ΣU-GRAPH problem is 'yes' if and only if the answer to the $(1|chains, p_j = 1|\Sigma U_j)$ problem is 'yes'. Since the decision version of problem $(1|chains, p_j =$

$1|\Sigma U_j$) is strongly NP-complete [Lenstra and Rinnooy Kan, 1980], we conclude that the decision version or problem ΣU-GRAPH is also strongly NP-complete. □

Proposition 4.5.3 shows that problem ΣT-GRAPH is strongly NP-hard, while Proposition 4.5.4 shows that problem ΣU-GRAPH is strongly NP-hard, even if the precedence graphs are only made of chains of tasks. Since, as we have seen in Section 4.2.2, problem ΣT is a special case of the Distance Criterion, and problem ΣU is a special case of the Binary Criterion, we get the following corollary.

Corollary 4.5.2: Distance and Binary - NP-hardness

Returning an optimal solution for the Distance Criterion or the Binary Criterion are strongly NP-hard problems when there are imposed precedence constraints. This is true even with precedence graphs only made of chains.

4.6 Conclusion

In this chapter, we studied the collective scheduling problem with unit size tasks, which can also be seen as a collective ranking problem since tasks of length 1 can be considered as items and preferred schedules as preferred rankings.

We introduced two general objective functions, one based on a distance, and the other one on a binary criterion. The distance based function minimizes the average distance between the returned schedule (or ranking) and the preferences of the voters (expressed as preferred schedules or preferred intervals for each task). It generalizes already known rules that minimize of the average deviation (ΣD), or the average tardiness (ΣT). The binary function generalizes the rule ΣU that minimizes the average number of late tasks. These rules can be applied in polynomial time even if we add release dates and deadlines constraints on the tasks.

We studied these two general rules from an axiomatic point of view when we infer release dates and deadlines from the preferences of the voters, showing that they do not fulfills release date or deadline consistency, but that they fulfill temporal unanimity, three axioms that we have introduced in this chapter.

We have also shown that the rule EMD which schedules the tasks by increasing median completion time (or by increasing median place in a ranking if we consider rankings instead of schedules), is a 2-approximation for the sum of the deviation (or the sum of the tardiness) minimization.

Last but not least, we studied the case where there are precedence constraints between the tasks. Note that precedence constraints also make sense in the context of rankings if items, a constraint between two items a and b saying that item a has to be ranked higher than item b. We showed that if the precedence constraints are fulfilled by the preferred schedules (or rankings) of the voters, then it is easy to get an optimal schedule (ranking) which fulfills the precedence constraints while minimizing the average deviation (or the average tardiness). When the preferred schedules do

not necessarily fulfill the constraints, we showed that on the contrary, it is NP-hard to find a schedule that fulfills the precedence constraints while minimizing the average deviation (or the average tardiness, or the average number of late tasks).

Chapter 5

A Non-Utilitarian Discrete Choice Model for Preference Aggregation

In this chapter, we take a probabilistic approach to preference aggregation. A set of v voters express preferences over a set of n candidates. In this chapter, we will follow a probabilistic approach as described in Section 1.2. We make the hypothesis that there exists a ground truth ranking, i.e. an objective way of ranking the candidates. The voters have a noisy perception of this ground truth and express their perception via their votes. In this chapter, we call "model" a probabilistic model which represents the noise. Such a model associates a conditional probability to each preference: it allows us to state that "if the ground truth ranking is R^*, then the probability for a voter to express her preference R is $p(R|R^*)$" and to compute the value of this probability. We study in this chapter a non-utilitarian discrete choice model for preference aggregation. Unlike the Plackett-Luce model, one of the most studied probabilistic model, that we will introduce in the chapter, this model is not based on the assignment of utility values to alternatives, but on probabilities p_i to choose the best alternative (according to a ground truth ranking R^*) in a subset of i alternatives. We consider $k-1$ parameters p_i (for $i=2$ to k) in the model, where k is bounded by the number n of alternatives (or candidates). We study the application of this model to voting, where we assume that the input is a set of choice functions provided by voters. If $k=2$, our model amounts to the model used by Young [1988] in his statistical analysis of Condorcet's voting method, and a maximum likelihood ranking is a consensus ranking for the Kemeny rule [Kemeny, 1959]. If $k>2$, we show that, under some restrictive assumptions about probabilities p_i, the maximum likelihood ranking is a consensus ranking for the k-wise Kemeny rule [Gilbert et al., 2020]. When we relax these assumptions, we provide a characterization result for the maximum likelihood ranking R and probabilities p_i. We propose an exact algorithm as well as a heuristic to compute both ranking R and probabilities p_i. Numerical tests are presented to assess the efficiency of these algorithms, and measure the model fitness on synthetic and real data.

The results presented in this chapter have been published in [Durand et al., 2022].

5.1 Introduction

5.1.1 Discrete choice models for preference aggregation

Preference aggregation is ubiquitous in multiple fields, among which are social choice [Arrow, 1951; Sen, 1999], information retrieval [Cormack et al., 2009], collaborative filtering [Pennock et al., 2000], or peer grading [Raman and Joachims, 2014]. The aggregation problem is formulated as follows: given v agents (or voters) and n alternatives (or candidates), each agent's preferences are specified by a ranking (permutation) of the alternatives, and the aim is to determine a single *consensus* ranking. Alternatively, preferences can also be expressed as choice functions instead of rankings [Aleskerov, 1999], i.e., each agent chooses her preferred candidate among various subsets of candidates. A choice function allows more possibilities for the voters (cyclic preferences are even possible), and may be easier to elicit if only a few subsets of candidates are considered. However, if all subsets of candidates are considered, their number becomes quickly very large (2^n). The procedure producing a consensus ranking from the v agents' preferences (expressed as rankings or choice functions) is called a *voting rule*.

A stream of research aims to rationalize voting rules by using statistical models for rank data, whose characteristics depend on the application domain (see e.g. [Xia, 2019]). This assumption of a statistical model behind the agents' preferences dates back to Condorcet. As emphasized by Young [1988], "Condorcet argued that if the object of voting is to determine the 'best' decision for society but voters sometimes make mistakes in their judgments, then the majority alternative (if it exists) is statistically most likely to be the best choice."

Young's examination of Condorcet's work through the lens of modern statistics leads him to put forward the Kemeny rule [Kemeny, 1959] (see Definition 1.2.2 in Section 1.2). This well-known rule consists of producing a consensus ranking R that minimizes the number of disagreements between R and the pairwise preferences of the agents on the candidates. Young shows that a consensus ranking for the Kemeny rule is a Maximum Likelihood Estimate (MLE) of a "ground truth" ranking R^* of the alternatives if one assumes that the pairwise preferences of the voters follow a statistical model parameterized by R^* under specific assumptions. The assumptions (already made by Condorcet) are:

1) In every pairwise comparison, each voter chooses the best alternative in R^* with some fixed probability p, with $p > \frac{1}{2}$.

2) Each voter's judgment on every pair of alternatives is independent of her judgment on every other pair[1].

3) Each voter's judgment is independent of the other voters' judgments.

When voters' preferences are expressed as rankings, it is also known that a consensus ranking for the Kemeny rule is an MLE of a ground truth ranking R^* for a

[1] Note that this assumption allows the preferences to be cyclic.

distance-based statistical model for ranking data [Conitzer et al., 2009]. Consider indeed the conditional probability distribution Pr on rankings R' of candidates defined by $Pr(R'|R^*) \propto 2^{-\Delta_{KT}(R^*,R')}$, where $\Delta_{KT}(R^*,R')$ is the Kendall tau distance between R^* and R' (number of pairwise disagreements between R^* and R'). Assuming that each voter's judgment is independent of the other voters' judgments, it is easy to show that the Kemeny rule returns a ranking R maximizing $Pr(R_1,\ldots,R_n|R) = \prod_{j=1}^{v} Pr(R_j|R) = 2^{-\sum_j \Delta_{KT}(R,R_j)}$, i.e., an MLE of R^*.

Other works about the use of MLE for preference aggregation explore the estimation of the parameters of *discrete choice models* from voting data. A discrete choice model consists of predicting the probabilities, called *choice probabilities*, of choosing $c \in S$ when presented with a subset S of alternatives, for each possible subset S [Luce, 2012]. A set of agents' rankings can be seen as choice data by considering that each ranking rationalizes a *choice function*. A choice function f picks a favorite alternative in any subset S of alternatives. For instance, the ranking $1 \prec 2 \prec 3$ (where "$\prec$" stands for "is preferred to") rationalizes the choice function $f(\{1,2\}) = 1$, $f(\{1,3\}) = 1$, $f(\{2,3\}) = 2$, and $f(\{1,2,3\}) = 1$. The most famous discrete choice model is known as the Plackett-Luce model. It consists in assigning a utility u_c to each alternative c, and setting the probability $Pr(f(S) = c)$ to choose c in S equal to $u_c / \sum_{d \in S} u_d$. The corresponding voting rule returns the ranking of alternatives by decreasing order of maximum likelihood utilities. Unlike most discrete choice model, the model we propose hereafter does not rely on the assignment of utility values (or utility distributions) to alternatives.

The use of discrete choice models based on utilities for preference aggregation deviates from Young's point of view. Indeed, Young uses distinct parameters to model, *on the one hand*, the respective "objective" skills of the candidates, namely the parameter R^* (ground truth ranking), and *on the other hand*, the "reliability" of the judgments of the voters, namely the parameter p (the closer the probability p is to 1, the more consistent the preferences are with the ground truth ranking). In discrete choice models based on utilities, the utilities are used *both* for modeling the objective skills of the candidates and the reliability of the judgments (the greater the differences in utilities, the more reliable the voters' judgments). Besides, unlike Young's model, that is related to the Kemeny rule, the consensus rankings obtained by sorting the candidates by decreasing order of maximum likelihood utilities are not related to well-identified voting rules.

5.1.2 Overview of our results

We propose a discrete choice model inspired by Young's model for the Kemeny rule. Given a ground truth ranking R^* of the alternatives, the choice of an agent in a subset of i alternatives is consistent with R^* with a probability p_i (p_i is $\alpha_i > 1$ times greater than the probability to choose any other given candidate in a subset of size i). Unlike many discrete choice models used for social choice, the model is thus *non-utilitarian*, i.e., not based on the assignment of utility scores to alternatives. While the introduction of utility scores is appealing because the cardinal data are richer than the ordinal ones,

the interpretation of such utility scores is not always obvious, e.g., when comparing artworks. We show the following results regarding the model we propose:

- Proposition 5.5.1 states that, if the value of α_i does not depend on i, then a maximum likelihood ranking is a consensus ranking for the k-wise Kemeny rule, a recently introduced voting rule that returns a ranking minimizing the number of disagreements with the choice functions of the voters [Gilbert et al., 2020].
- If values α_i depend on i, we provide a characterization result (Proposition 5.5.2) for a maximum likelihood estimation of the ground truth ranking R^* and α_i's. The characterization involves a weighted variant of the k-wise Kemeny rule.
- Based on Proposition 5.5.2, we provide an exact algorithm and a heuristic algorithm for determining a maximum likelihood couple in the general case.
- Finally, using synthetic and real data, we present numerical tests to assess the efficiency of these algorithms, as well as the model fitness to data.

5.2 Related work

The related work concerns either the maximum likelihood approach to voting, or set extensions of the Kemeny rule.

The maximum likelihood approach to voting. In this approach, we make the assumption that a true "objective" ranking of the candidates exists, and that the preferences expressed by the voters are noisy observations of this true ranking. If the preferences are rankings drawn i.i.d. from a distribution, the probability of observing a set $P = \{R_1, \ldots, R_n\}$ is then $Pr(P|R) = \prod_{j=1}^{n} Pr(R_j|R)$. Each probability model for $Pr(R_j|R)$ induces a voting rule where a ranking maximizing $Pr(P|R)$ (the likelihood) is a consensus ranking. Drissi-Bakhkhat and Truchon [2004] investigate a setting in which the probability of comparing two alternatives consistently with a ground truth ranking R^* is increasing with the distance between them in R^*. This leads to a new voting rule that the authors examined from an axiomatic point of view. While every noise model on the votes[2] induces a voting rule, Conitzer and Sandholm [2005] study the opposite direction, using it as a way to rationalize voting rules. They identify noise models for which an MLE ranking is a consensus ranking of well-known voting rules (scoring rules and single transferable vote), and on the contrary, for other rules (Bucklin, Copeland, maximin), they show that no such noise model can be constructed. Conitzer et al. [2009] pursue this line of work, providing an exact characterization of the class of voting rules for which a noise model can be constructed. More recently, Caragiannis et al. [2016] study how many votes are needed by a voting rule to reconstruct the true

[2]When the votes are viewed as noisy perceptions of a ground truth ranking R^*, a noise model is the mathematical description of the probabilities of the votes based on R^*.

ranking. Another line of research focuses on the use of discrete choice models in social choice. Soufiani et al. [2012] study an extension of the Plackett-Luce model. This model can be viewed as a random utility model in which the utilities of alternatives are drawn i.i.d. from a Gumbel distribution. They propose a random utility model based on distributions in the exponential family (to which Gumbel distributions belong), as well as inference methods for the parameters. Among other results, they showed that their model fits better than the Plackett-Luce model to the well-known sushi dataset [Kamishima, 2003].

Set extensions of the Kemeny rule. Gilbert et al. [2020] introduce the k-wise Kemeny rule, show that the computation of a consensus ranking according to this rule is NP-hard, and provide a dynamic programming procedure for this purpose. We will give a detailed presentation of this rule in the ext section. At least two other set extensions of the Kemeny rule have been proposed. Both extensions consider a setting in which, although the voters have preferences over a set S, the election will in fact occur on a subset $S \subseteq S$ drawn according to a probability distribution on 2^S [Baldiga and Green, 2013; Lu and Boutilier, 2010]. A consensus ranking R is then one that minimizes, in expectation, the number of voters' disagreements with the chosen candidate in S (a voter disagrees with R on S if $t_R(S)$ is not her most preferred candidate in S). Baldiga and Green study a setting in which the probability $Pr(S)$ only depends on the cardinality of S. Lu and Boutilier study a special case of the previous setting, where each candidate is unavailable in S with a probability p, independently of the others, i.e., $Pr(S) = p^{|C\setminus S|}(1-p)^{|S|}$. Proposition 5.5.2 later in the paper uses a weighted sum of disagreements $\Delta_{KT}^{k,\alpha}$ on subsets of size at most k that is formally equivalent to the rule used by Baldiga and Green for $k = n$: the weights $\log \alpha_i$ assigned to disagreements on subsets S of size $i = |S|$ play the role of $Pr(S)$. However, the viewpoint we take here is completely different, as the values α_i are not given, but inferred from the choice data. In addition, to determine a maximum likelihood ranking for our model, we do not minimize $\Delta_{KT}^{k,\alpha}$ only, but the sum of $\Delta_{KT}^{k,\alpha}$ and another term.

5.3 Preliminaries

In the following, we will consider that the preferences of the agents are expressed as choice functions. A first possibility to elicit these choice functions is by asking each agent to give her most preferred alternative for each subset of size at most k – this may be a good solution if there are few candidates and k is not too large, or when the agents are not able to give their preferences as rankings. Another possibility is to ask for rankings, and infer choice functions from them (the choice in a subset S of candidates is the highest ranked candidate among S) – a ranking can be seen as a compact representation of a choice function.

Example 5.3.1: Inferring choice functions

Let us consider 3 candidates $\{c_1, c_2, c_3\}$ and 10 voters with preferences, expressed as rankings, as follows:

- 3 voters of type I have preferences $c_1 \prec c_2 \prec c_3$.
- 3 voters of type II have preferences $c_3 \prec c_1 \prec c_2$.
- 2 voters of type III have preferences $c_2 \prec c_1 \prec c_3$.
- 2 voters of type IV have preferences $c_3 \prec c_2 \prec c_1$.

This preference profile yields the choice function profile given in Table 5.1, where each cell gives the favorite alternative $f_j(S)$ in S for voter j of the type corresponding to the row. For example, considering the rightmost column, one sees that c_1 (resp. c_3) is the preferred candidate in $\{c_1, c_2, c_3\}$ for voters of type I (resp. II and IV).

	$S=\{c_1,c_2\}$	$S=\{c_1,c_3\}$	$S=\{c_2,c_3\}$	$S=\{c_1,c_2,c_3\}$
$f_j(S)$ (j of type I)	c_1	c_1	c_2	c_1
$f_j(S)$ (j of type II)	c_1	c_3	c_3	c_3
$f_j(S)$ (j of type III)	c_2	c_1	c_2	c_2
$f_j(S)$ (j of type IV)	c_2	c_3	c_3	c_3

Table 5.1: The choice function profile in Example 5.3.1.

Let $V = \{v_1, \ldots, v_v\}$ be a set of v agents (or voters) and S a set of n alternatives (or candidates). We denote by X^S the set of the $n!$ possible rankings of S. For $k \in \{2, \ldots, n\}$, we denote by Φ_k the set of all subsets S of S such that $2 \leq |S| \leq k$. Given a value $k \in \{2, \ldots, n\}$, each agent $v_j \in V$ has a choice function $f_j : \Phi_k \to S$ which gives, for each subset S of alternatives of size at most k, her preferred alternative in S (assuming that each agent has only one favorite alternative per subset). We denote by $\mathscr{F}_k$ the set of all possible choice functions on sets of size at most k. A *choice function profile* $P = (f_1, \ldots, f_v) \in \mathscr{F}_k^v$ is a tuple of v choice functions f_j, one per agent. In this setting, the purpose of preference aggregation is to determine a *consensus ranking* from the choice functions in P. A voting rule $r : \mathscr{F}_k^v \to (2^{X^S} \setminus \{\emptyset\})$ in which ballots are choice functions, maps each choice function profile to a non-empty set of consensus rankings.

The statistical model for choice functions studied in this paper will reveal closely related to a recently proposed voting rule, namely the k-wise Kemeny rule [Gilbert et al., 2020]. To compute a consensus rankings for the k-wise Kemeny rule, one needs only the information from the *choice matrix* derived from P, denoted by M_P. The choice matrix gives, for each subset S of candidates and each candidate c, the number of voters for which c is the preferred candidate in S. If only subsets of size at most k matter, the choice matrix can be restricted to these subsets.

Example 5.3.2: Example 5.3.1 continued - Choice matrix

The choice matrix synthesizing the results of all setwise contests for the choice functions of Table 5.1 is given in Table 5.2. The matrix reads as follows: for instance, considering the rightmost column, one sees that c_1 is the most preferred candidate in $\{c_1, c_2, c_3\}$ for 3 voters, c_2 is the most preferred candidate for 2 voters, and c_3 is the most preferred candidate for 5 voters.

S	$\{c_1, c_2\}$	$\{c_1, c_3\}$	$\{c_2, c_3\}$	$\{c_1, c_2, c_3\}$
c_1	6	5	–	3
c_2	4	–	5	2
c_3	–	5	5	5

Table 5.2: The choice matrix associated to the instance of Example 5.3.1.

We now formally describe the k-wise Kemeny rule. Given a ranking R and a subset $S \in \Phi_k$ of candidates, let $t_R(S) \in S$ denote the most preferred candidate in S for R (i.e., for each candidate $c \neq t_R(S) \in S$, $t_R(S)$ is ranked at a higher position in R than c – is preferred to c). The k-wise distance $\Delta^k_{KT}(R, f)$ between a ranking R and a choice function $f \in \mathscr{F}_k$ is the number of disagreements between R and f on sets of candidates of size between 2 and k:

$$\Delta^k_{KT}(R, f) = \sum_{S \in \Phi_k} \mathbb{1}_{t_R(S) \neq f(S)}$$

where $\mathbb{1}_{t_R(S) \neq f(S)} = 1$ if $t_R(S) \neq f(S)$, 0 otherwise. Note that when $k = 2$, $\Delta^2_{KT}(R, f)$ is the well-known Kendall tau distance between R and $f \in \mathscr{F}_2$ (which associates a winner to each pair of candidates). We may also express Δ^k_{KT} by splitting Φ_k into sets of subsets of the same cardinality. Let us denote by S_i the set of subsets of S of cardinality equal to i. We have thus $\bigcup_{i=2}^{k} S_i = \Phi_k$ and Δ^k_{KT} can be written:

$$\Delta^k_{KT}(R, f) = \sum_{i=2}^{k} \sum_{S \in S_i} \mathbb{1}_{t_R(S) \neq f(S)}$$

Given a profile P, the cost of a ranking R is the sum of the k-wise distances between R and each choice function f_j ($j \in \{1, \ldots, v\}$) in the choice function profile. It is thus the total number of disagreements between R and the voters on all the possible subsets of candidates of size at most k:

$$\Delta^k_{KT}(R, P) = \sum_{j=1}^{n} \Delta^k_{KT}(R, f_j) = \sum_{j=1}^{n} \sum_{i=2}^{k} \sum_{S \in S_i} \mathbb{1}_{t_R(S) \neq f_j(S)}$$

The k-wise Kemeny rule determines a ranking minimizing $\Delta^k_{KT}(R, P)$ among all the rankings $R \in X^S$. To compute such a consensus ranking, one needs only the information

from the choice matrix M_P. It indeed minimizes $\Delta^k_{KT}(R,P)$:

$$\Delta^k_{KT}(R,P) = \sum_{i=2}^{k} \sum_{S \in S_i} d(M_P, S, R) \tag{5.1}$$

where $d(M_P,S,R)$ is the number of voters whose most preferred candidate in S is *not* $t_R(S)$ (number of disagreements between R and M_P for S). That is, a consensus ranking for the k-wise Kemeny rule minimizes the number of disagreements with the agents' choice functions on subsets of cardinality at most k (a disagreement occurs between a ranking R and choice function f on a subset S of candidates if $f(S) \neq t_R(S)$). The k-wise Kemeny rule generalizes the Kemeny rule, as it amounts to the usual Kemeny rule if $k=2$. Increasing k allows to overcome a well-known drawback of the Kemeny rule, namely that very different consensus rankings may coexist. The example below illustrates this.

Example 5.3.3: Example 5.3.1 continued - k-wise Kemeny rule

In Example 5.3.1, there are two consensus rankings for the Kemeny rule, namely $c_1 \prec c_2 \prec c_3$ and $c_3 \prec c_1 \prec c_2$ since both of them induce 14 pairwise disagreements with the preference profile. Candidate c_3 is thus ranked last in the former, and first in the latter. In contrast, for the 3-wise Kemeny rule, the only consensus ranking is $c_3 \prec c_1 \prec c_2$, with $14+5=19$ disagreements (14 on pairs, and 5 on $\{c_1,c_2,c_3\}$), while there are $14+7=21$ disagreements for $c_1 \prec c_2 \prec c_3$.

5.4 A Non-Utilitarian Discrete Choice Model

We now present the statistical model on choice functions that we will study in the remainder of the paper. The sample space (i.e., the possible observed outcomes from which the parameter of the statistical model are inferred) is the set of choice matrices. In this framework, the assumptions made by Condorcet and Young (see the introduction) need to be adapted, as we consider not only choices on pairs of candidates but also on subsets. Given a true ranking R^* of S, the following assumptions are made on random variables $f_j(S)$ for all voters v_j:

1. for every $i \in \{2,\dots,k\}$, $S \in S_i$, $c \in S \setminus \{t_{R^*}(S)\}$, the probability that $f_j(S) = t_{R^*}(S)$ is $\alpha_i > 1$ times larger than the probability that $f_j(S)=c$: $Pr(f_j(S)=t_{R^*}(S)) = \alpha_i \cdot Pr(f_j(S)=c)$; that is, it is $\alpha_{|S|}$ more likely to choose the highest ranked candidate of S in R^* than any other given candidate of S.

2. for every pair $\{S,S'\}$ of subsets in Φ_k, $f_j(S)$ and $f_j(S')$ are independent.

For any pair $\{v_j, v_{j'}\}$ of voters, we also assume that each voter's preferred choice on each subset of candidates is independent of the other voters' preferences, i.e.:

3. for every $\{v_j, v_{j'}\} \subseteq V$ and $(S,S') \in (\Phi_k)^2$, $f_j(S)$ and $f_{j'}(S')$ are independent.

For $k=2$, these assumptions amount to those made by Young on pairwise judgments in his analysis of Condorcet's theory of voting. If $k>2$, the additional parameters $\alpha_{|S|}$ (for $|S|>2$) give more flexibility to fit the observed choice data, at the cost of a greater computational load. Note that Assumption 1 means that the probability that voter v_j agree with ranking R^* on the preferred candidate in S depends only on the size of S, and not on the members of S.

Assumptions 1 and 2 yield the following statistical model for choice functions f, that we call *k-wise Young's model*, parameterized by a ranking R and choice probabilities $p_i=\alpha_i/(\alpha_i+i-1)$ (conversely, $\alpha_i=(i-1)p_i/(1-p_i)$), where p_i represents $Pr(f(S)=t_r(S))$ for $|S|=i$:

Definition 5.4.1: *k*-wise Young's Model

Given a set S of n alternatives, the k-wise Young's model is defined as follows:

- the parameter space is $X^S\times\Theta$, where X^S is the set of rankings on S and $\Theta=(\frac{1}{2},1]\times\ldots\times(\frac{1}{k},1]$ is the set of choice probabilities $\overrightarrow{p}=(p_2,\ldots,p_k)$,
- for any $(R,\overrightarrow{p})\in X^S\times\Theta$, the probability $Pr(f|R,\overrightarrow{p})$ is

$$\prod_{i=2}^{k}\prod_{S\in S_i} p_i^{1-\mathbb{1}_{t_R(S)\neq f(S)}}\left(\frac{1-p_i}{i-1}\right)^{\mathbb{1}_{t_R(S)\neq f(S)}}$$

where $\mathbb{1}_{t_R(S)\neq f(S)}=1$ if $t_R(S)\neq f(S)$, 0 otherwise.

If $R=R^*$, we have indeed $Pr(f(S)=c)=(1-p_i)/(i-1)$ for $c\neq t_R(S)$ by Assumption 1, and the products in the formula for $Pr(f|R,\overrightarrow{p})$ follow from Assumption 2.

As the preferences revealed by the choices may be cyclic, sampling a choice function according to this model can be decomposed into independent draws for each subset $S\subseteq S$. Given a choice function profile P with v voters, if one assumes the functions in P are *independently* sampled (in line with Assumption 3) from a k-wise Young's model of parameters R and $\overrightarrow{p}$, the probability $Pr(M_P|R,\overrightarrow{p})$ follows a multinomial distribution:

$$\prod_{i=2}^{k}\prod_{S\in S_i}\frac{v!}{\prod_{c\in S}v_c!}p_i^{v-d(M_P,S,R)}\left(\frac{1-p_i}{i-1}\right)^{d(M_P,S,R)} \tag{5.2}$$

where v_c denotes the number of voters that choose candidate c in subset S.

From Equation 5.2, it is clear that the likelihood of $(R,\overrightarrow{p})$ given M_P, denoted by $\mathcal{L}(R,\overrightarrow{p}|M_P)$, is proportional to

$$\prod_{i=2}^{k}\prod_{S\in S_i} p_i^{v-d(M_P,S,R)}\left(\frac{1-p_i}{i-1}\right)^{d(M_P,S,R)} \tag{5.3}$$

because the coefficients $v!/(\prod_{c\in S}v_c!)$ depend neither on R nor on $\overrightarrow{p}$. Let us now study different voting rules arising from Equation 5.3. Depending on whether or not restrictive assumptions are made about probabilities p_i, we show that a maximum likelihood

ranking R is a consensus ranking for the k-wise Kemeny rule, or for a weighted variant whose parameters vary with M_P and R.

5.5 MLE of the Parameters of the k-Wise Young's Model

A consensus ranking for the k-wise Kemeny rule is an MLE of a ground truth ranking R^* if one assumes that the choice function profile is sampled according to the k-wise Young's model when $\alpha_2 = \ldots = \alpha_k = \alpha > 1$, i.e., in a subset S, candidate $t_{R^*}(S)$ is the most likely to be chosen, with a probability α times greater than any other given member of S, whatever the size of S. More formally:

Proposition 5.5.1: k-wise Young model and k-wise Kemeny rule

If there exists $\alpha > 1$ such that $\alpha_2 = \ldots = \alpha_k = \alpha$, then, given a choice matrix M_P, a ranking R has maximum likelihood for the k-wise Young's model iff it minimizes $\Delta^k_{KT}(R,P)$, i.e., ranking R is a consensus ranking for the k-wise Kemeny rule.

Proof. Maximizing Equation 5.3 amounts to maximizing:

$$\log\left(\prod_{i=2}^{k}\prod_{S\in S_i} p_i^{v-d(M_P,S,R)}\cdot\left(\frac{1-p_i}{i-1}\right)^{d(M_P,S,R)}\right)$$

$$=\sum_{i=2}^{k}\sum_{S\in S_i}\left((v-d(M_P,S,R))\cdot\log p_i + d(M_P,S,R)\cdot\log\left(\frac{1-p_i}{i-1}\right)\right)$$

$$=\sum_{i=2}^{k}\sum_{S\in S_i} v\log p_i - \sum_{i=2}^{k}\sum_{S\in S_i} d(M_P,S,R)\cdot\log\left(\frac{p_i}{\frac{1-p_i}{i-1}}\right)$$

For a given set of values p_i, determining a ranking R that maximizes the above formula is equivalent to minimizing:

$$\sum_{i=2}^{k}\sum_{S\in S_i} d(M_P,S,r)\cdot\log\left(\frac{p_i}{\frac{1-p_i}{i-1}}\right)$$

As p_i is the probability to choose $t_R(S)$ and $(1-p_i)/(i-1)$ the probability to choose any other member of S, we have $p_i/(1-p_i/(i-1)) = \alpha_i$. Furthermore, by assumption, $\alpha_i = \alpha\ \forall i \in \{2,\ldots,k\}$. Consequently, the expression simplifies to:

$$(\log\alpha)\cdot\sum_{i=2}^{k}\sum_{S\in S_i} d(M_P,S,R)$$

The coefficient $\log\alpha$ is strictly positive because $\alpha > 1$ by assumption, and it can therefore be omitted when minimizing according to R. From Equation 5.1, we have:

$$\sum_{i=2}^{k}\sum_{S\in S_i} d(M_P,S,R) = \Delta^k_{KT}(R,P)$$

Therefore, whatever the vector $\overrightarrow{p}$ of choice probabilities, a ranking R that maximizes $\mathcal{L}(R, \overrightarrow{p}|M_P)$ minimizes $\Delta^k_{KT}(R,P)$, which concludes the proof. □

For $k=2$, this proposition amounts to the result of Young regarding the interpretation of the Kemeny rule as an MLE of a ground truth ranking.

If we do not assume that the α_i are equal, then the maximum likelihood ranking may depend on $\overrightarrow{\alpha}=(\alpha_2,\dots,\alpha_k)$, and we need to determine[3] a *couple* $(R,\overrightarrow{\alpha})$ of maximum likelihood $\mathcal{L}(R,\overrightarrow{\alpha}|M_P)$, even if we are only interested in R. Determining such a couple $(R,\overrightarrow{\alpha})$ defines a new voting rule in itself, which returns R as a consensus ranking. The following result shows that it can be formulated as a discrete optimization problem on the space of rankings, because, for each ranking R, there exists a closed-form expression to determine the corresponding maximum likelihood values α_i.

Proposition 5.5.2: Characterization of a maximum likelihood ranking

Given a choice matrix M_P, a couple $(R,\overrightarrow{\alpha})$ has maximum likelihood for the k-wise Young's model if and only if ranking R minimizes

$$\Delta^{k,\alpha}_{KT}(R,P) - \sum_{i=2}^{k}\sum_{S\in S_i} v\log\frac{\alpha_i}{\alpha_i+i-1}, \tag{5.4}$$

$$where\ \ \Delta^{k,\alpha}_{KT}(R,P) = \sum_{i=2}^{k}(\log\alpha_i)\sum_{S\in S_i} d(M_P,S,R) \tag{5.5}$$

$$\text{and}\quad \alpha_i = ((i-1)\cdot\sum_{S\in S_i}(v-d(M_P,S,R)))/(\sum_{S\in S_i} d(P,S,R)). \tag{5.6}$$

Proof. From the proof of Proposition 5.5.1, we know that a couple $(R,\overrightarrow{p})$ has maximum likelihood iff, for a given choice matrix M_P and ranking R, it maximizes:

$$f(\overrightarrow{p}) = \sum_{i=2}^{k}\sum_{S\in S_i} v\log p_i - \sum_{i=2}^{k}\sum_{S\in S_i} d(M_P,S,R)\cdot\log\left(\frac{p_i}{\frac{1-p_i}{i-1}}\right) \tag{5.7}$$

To determine an optimum of function f, each component p_i can be optimized independently from the others, because each one appears in a different term of the sum from $i=2$ to k. Noting that $\sum_{S\in S_i} v = \binom{n}{i}\cdot v$ as there are $\binom{n}{i}$ different subsets $S\in S_i$, the partial derivative of order 1 is written as:

$$\frac{\partial f}{\partial p_i}(\overrightarrow{p}) = \frac{\binom{n}{i}\cdot v - \sum\limits_{S\in S_i} d(M_P,S,R)}{p_i} - \frac{\sum\limits_{S\in S_i} d(P,S,R)}{(1-p_i)}.$$

[3] From now on, we use indifferently $\overrightarrow{p}$ or $\overrightarrow{\alpha}$, because one vector can be inferred from the other.

For $p_i \in [0,1]$, the derivative vanishes for:

$$p_i = \left(\binom{n}{i} \cdot v - \sum_{S \in S_i} d(M_P, S, R) \right) / \left(\binom{n}{i} \cdot v \right).$$

It is easy to prove that $\frac{\partial^2 f}{\partial p_i^2}(\overrightarrow{p}) < 0$ for $p_i \in [0,1]$, thus the corresponding stationary point of f is a maximum. From the values p_i of maximum likelihood we derive the values α_i of maximum likelihood:

$$\alpha_i = \frac{p_i}{\frac{1-p_i}{(i-1)}} = (i-1) \cdot \frac{\sum_{S \in S_i} (n - d(M_P, S, R))}{\sum_{S \in S_i} d(M_P, S, R)}$$

The result is obtained by expressing Equation 5.7 in function of α_i instead of p_i, and turning the maximization into a minimization of the opposite expression. □

Note that, according to Proposition 5.5.2, the maximum likelihood value of each p_i given R corresponds to the observed proportion of agreements between R and P on subsets of size i, which is consistent with intuition. The formula of the likelihood of a couple $(R, \overrightarrow{\alpha})$ is written as the sum of two terms:

- the term $\Delta_{KT}^{k,\alpha}(R,P)$ is a weighted sum of disagreements between R and P, where the disagreements on subsets of size i are weighted by $\log \alpha_i$;
- the term $-\sum_{i=2}^{k} \sum_{S \in S_i} v \log(\alpha_i/(\alpha_i+i-1)) = -\log \prod_{j=1}^{v} \prod_{i=2}^{k} \prod_{S \in S_i} p_i$; as $\prod_{j=1}^{v} \prod_{i=2}^{k} \prod_{S \in S_i} p_i \leq 1$, the opposite of its logarithm is positive, and the term is all the greater as the empirical probability that the v choice functions in P coincide with R is low.

Let us now present algorithms (an exact one and a heuristic one) to compute a maximum likelihood couple $(R, \overrightarrow{\alpha})$ given a choice matrix M_P.

5.6 Algorithms for Determining an MLE

A brute force method for determining a couple $(R, \overrightarrow{\alpha})$ of maximum likelihood given P consists of computing a vector $\overrightarrow{\alpha}$ of maximum likelihood for each ranking R (thanks to Prop. 5.5.2), and, turning $\overrightarrow{\alpha}$ into $\overrightarrow{p}$, retaining the couple $(R, \overrightarrow{p})$ that maximizes Equation 5.3.

A Faster Exact Algorithm. It is possible to improve this procedure by considering only a *subset* of rankings R on the candidates. We know indeed from Proposition 5.5.2 that, for any given $\overrightarrow{\alpha}$, the corresponding maximum likelihood ranking R minimizes $\Delta_{KT}^{k,\alpha}(R,P)$ (see Equation 5.5). Minimizing $\Delta_{KT}^{k,\alpha}(R,P)$ can be seen as a multi-objective optimization problem, by associating to each R the vector:

$$\overrightarrow{d}_P(R) = \left(\sum_{S \in S_2} d(M_P, S, R), \ldots, \sum_{S \in S_k} d(M_P, S, R) \right)$$

In multi-objective optimization problems, the goal is often to enumerate all the Pareto optimal solutions, i.e., in our setting, the rankings R such that there does not exist another ranking R' for which $\overrightarrow{d}_P(R') \leq \overrightarrow{d}_P(R)$, where $\overrightarrow{x} \leq \overrightarrow{y}$ if for all i in $\{2,\dots,k\}$ $x_i \leq y_i$, and there exists i in $\{2,\dots,k\}$ $x_i < y_i$. A ranking R minimizing $\Delta_{KT}^{k,\alpha}(R,P)$ is obviously Pareto optimal. Such a ranking actually belongs to a more restricted set: the set of *supported* solutions, i.e., those that optimizes a weighted sum of the objectives [Ehrgott, 2005]. The weight assigned to each objective i is here $\log \alpha_i$. An even more restricted set can be considered: the set of *extreme* rankings. A Pareto optimal ranking R is *extreme* if $\overrightarrow{d}_P(R)$ is a vertex of the convex hull of $\{\overrightarrow{d}_P(R) : R \in X^S\}$ in the $(k-1)$-dimensional objective space, where X^S is the set of all rankings. Indeed, it is well-known in multi-objective optimization that, for each supported ranking R', there exists an extreme ranking R such that $\Delta_{KT}^{k,\alpha}(R,P) = \Delta_{KT}^{k,\alpha}(R',P)$.

A recent work presents a method for enumerating the extreme solutions in multi-objective optimization problems [Przybylski et al., 2019]. Based on such a method, we design an exact procedure for determining a maximum likelihood pair $(R, \overrightarrow{\alpha})$ by Prop. 5.5.2:

1. Determine all the extreme rankings by using the method by Przybylski et al. [2019];
2. For each extreme ranking R, compute by Equation 5.6 the vector $\overrightarrow{\alpha}_R$ such that $(R, \overrightarrow{\alpha})$ has maximum likelihood;
3. Return a couple $(R, \overrightarrow{\alpha}_R)$ that minimizes Equation 5.4.

Although this procedure allows us to reduce the number of rankings we need to consider, there are still many of them, especially when the value of k increases. For this reason, we now propose a faster heuristic giving a very good approximation of an optimal couple $(R, \overrightarrow{\alpha})$.

A Heuristic Algorithm. Instead of considering all the extreme rankings, we propose an Iterative Optimization (IO) heuristic, which alternates two steps:

- α-step: compute an $\overrightarrow{\alpha}$ of maximum likelihood given R by Equation 5.6;
- R-step: compute an R of maximum likelihood given $\overrightarrow{\alpha}$ by minimizing $\Delta_{KT}^{k,\alpha}(R,P)$ (see Equation 5.5).

The minimization of $\Delta_{KT}^{k,\alpha}(R,P)$ is performed thanks to a weighted variant of the dynamic programming algorithm proposed by Gilbert et al. [2020] for the k-wise Kemeny rule. The two steps are alternated until the same ranking is found in two consecutive R-steps.

The complexity of the dynamic programming algorithm that computes a ranking R minimizing $\Delta_{KT}^{k,\alpha}(R,P)$ is $O(2^n n^2 v)$, thus the heuristic is not polynomial time (but much faster than the exact algorithm, as will be seen later).

The output of the IO algorithm may depend on the chosen initial ranking. We investigated several ways of generating the initial ranking: we used rankings computed by, Borda or Spearman voting rules (both can be obtained in polynomial time); or by launching the algorithm from a given vector $\overrightarrow{\alpha}$ for the R-step (in the numerical tests, we have set the values α_i corresponding to $p_i = 1/i + (i-1)/(10i)$). These different variants are experimentally evaluated in the next section.

5.7 Numerical Tests

We report here the results of several experiments[4] to test the performance of our heuristic and the fitness of the k-wise Young's (k-wise) model compared to that of the Plackett-Luce (PL) model on synthetic and real-world data.

Instances. The tests are carried out both on real data sets from the Preflib library [Mattei and Walsh, 2013], and on three types of synthetic instances. The first type of synthetic instances are *uniform instances*, in which the preferences of each voter is a random ranking in the set X^S of all permutations. The second type of instances, called *PL instances*, are preference profiles generated thanks to the PL model [Plackett, 1975; Luce, 2012]. The third type of instances, called *k-wise instances*, are choice matrices generated with our model. Given a ground truth ranking R^*, the choice function of a voter is generated as follows: for each subset S of size i, the voter chooses the winner in S w.r.t. R^* with probability p_i, and chooses any other candidate in S with probability $(1-p_i)/(i-1)$. We set $k = n$ in all tests.

Performance of the heuristic. In order to evaluate the performance of the IO heuristic, we compare the log-likelihood (LL) of the returned pair $(R, \overrightarrow{\alpha})$ with the one obtained by the exact method. Denoting by $OPT(I)$ the value of the LL of an optimal pair for a given instance I and by $IO(I)$ the value of the LL of the pair returned by the IO method, we calculate the ratio $q = IO(I)/OPT(I)$. The optimal value is calculated using the exact algorithm described in Section 5.6. Since the performance of the heuristic may vary in function of the initial ranking, we try different initial rankings : an optimal ranking for the Borda rule (B), or for Spearman (SM). For all the real instances from the PrefLib library, and for all tested PL instances, the heuristic always returns an optimal pair $(R, \overrightarrow{\alpha})$. On uniform instances, the result is not always optimal, but it is very close to the optimal LL: the ratio q is above 0.9999 on average, regardless of the initial ranking. In fact, the log-likelihood obtained was optimal for 575 (resp. 581, 586) instances out of 600 with Borda start (resp. Spearman start, α start). The heuristic provides an excellent approximation of an optimal pair $(R, \overrightarrow{\alpha})$. As said earlier, the heuristic is not a polynomial time algorithm, but it remains much faster than the exact multi-objective algorithm.

[4] All algorithms have been implemented in C++, and the tests have been carried out on an Intel Core I5-8250 1.6GHz processor with 8GB of RAM.

Computation times. Regardless of the type of instance, the IO method is much faster than the exact multi-objective algorithm. For example, for $m = k = 8$, the IO method takes 260 (resp. 185) seconds on average to return a solution for uniform (resp. PL) instances whereas the exact algorithm requires 30000 (resp. 1000) seconds on average to determine an optimal pair for the same instances.

Figure 5.1 shows the computation time of the IO method in function of k for uniform (left) or PL (right) instances with 8 candidates. It shows that the initial ranking does not have a big impact on the computation time. In this figure, we see that the variant starting from a given vector $\overrightarrow{\alpha}$ is faster, although this initialization step takes more time that the computation of the Borda and Spearman rankings (which can be computed in polynomial time). Indeed, the variant starting from $\overrightarrow{\alpha}$ provides a better initial ranking than Borda and Spearman, which reduces the number of iterations and consequently the overall running time of the algorithm. This does not hold however for the PL instances, for which the Borda and the Spearman start lead to a faster resolution, because the Borda and Spearman rankings are computable in polynomial time and are, in this case, very good approximations of a ranking of maximum likelihood.

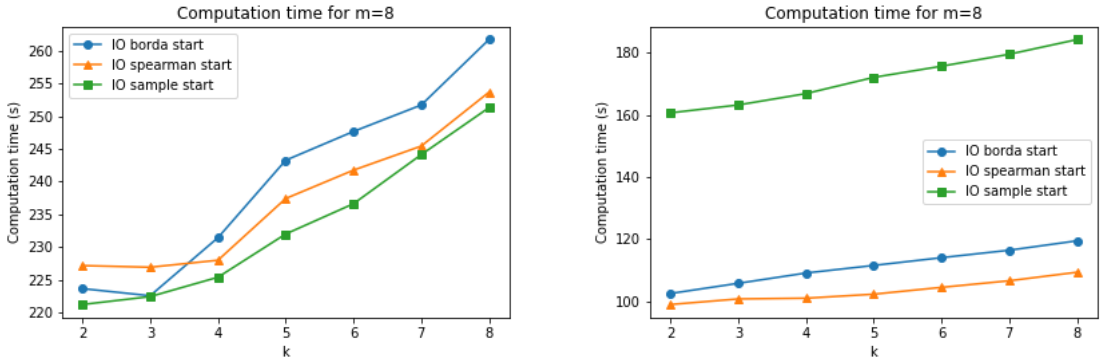


Figure 5.1: Computation time (in seconds) of the IO method on uniform (left) and PL (right) instances. "IO sample start" denotes for the variant that is launched from a given vector of values α_i.

Model fitness. We now compare our model with the PL model in terms of fitness with real-world data. We use instances from the sushi dataset [Kamishima, 2003], in which 5000 voters give their ranking over 10 kinds of sushis. We randomly draw $n \in \{50, 100\}$ voters among the 5000. We apply the exact solution procedure proposed above, and compare the results with those of the PL model. The likelihood of a choice matrix w.r.t. the PL model for choice functions is written as follows: $\prod_{i=2}^{k} \prod_{S \in S_i} \frac{v!}{\prod_{c \in S} v_c!} \prod_{c \in S} \left(\frac{u_c}{\sum_{d \in S} u_d} \right)^{v_c}$ where v_c denotes the number of voters choosing candidate c in S, and u_c the utility of c. To compare the fitness of the models, we use the Bayesian Information Criterion –BIC– [Schwarz, 1978]. Regarding the k-wise model, we consider the case of constant α_i's (model α) and the general case where the α_i may vary (model α_i). We compute the ratio $BIC(\mu)/BIC(PL)$ for $\mu \in \{\alpha, \alpha_i\}$. For 50 voters, the obtained ratio is 1.054 (resp.

1.047) for model α (resp. α_i). This improves to 1.052 (resp. 1.045) for 100 voters. This shows that for this dataset, the fitness of models α and α_i is close to the PL model, although the fitness of the latter is slightly better (by 5% at most).

Comparison with the PL model. We now compare the PL model and the k-wise model on PL instances and k-wise instances. In both cases, we compute a correlation factor ρ between the returned ranking and the ground truth ranking used for generation. The factor ρ is the Kendall-Tau distance normalized between 0 and 1 – 0 indicates that the two rankings are identical while 1 means that they are opposite. Figure 5.2 shows the mean value of ρ in function of the level of correlation of the voters' preferences, for k-wise instances (left) and PL instances (right). For k-wise instances, the correlation between the choice functions is controlled by setting $p_i = 1/i + x(1-1/i)$ for $x \in [0,1]$: all choice functions are equally likely and independent from the ground truth ranking for $x = 0$, while all choice functions are perfectly consistent with the ground truth ranking for $x = 1$. For PL instances, the correlation between the rankings is controlled by setting $u_p = 1 + (m-p)x$ as utility of the candidate in position p in the ground truth ranking: the higher x, the stronger the correlation. As one would expect, the MLE ranking for the k-wise (resp. PL) model is closer to the ground truth ranking on k-wise (resp. PL) instances. Interestingly, the k-wise model performs better on PL instances than the PL model on k-wise instances. When instances are correlated enough, the MLE ranking for both models always correspond to the ground truth.

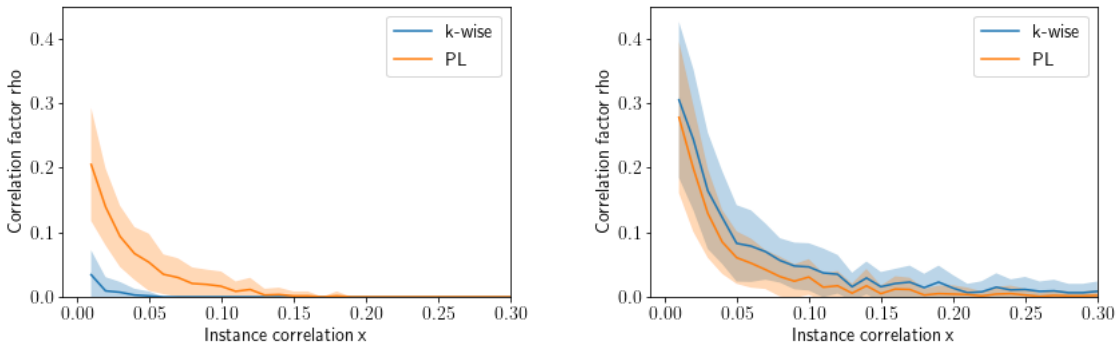


Figure 5.2: Mean ρ (and 68% confidence interval) between the returned ranking and the ground truth on k-wise instances (left) and PL instances (right).

5.8 Conclusion

We have studied here an extension of Young's model for pairwise preferences to choices in subsets of size at most k, showing that the maximum likelihood ranking w.r.t. this model coincides with a consensus ranking for the k-wise Kemeny rule under certain assumptions on the choice probabilities. Relaxing these assumptions, we have proposed inference algorithms for the model, learning the choice probabilities from the

data. The fitness of the model on real data is comparable with the Plackett-Luce model, although no utilities are embedded in our model. This is a first step towards the use of non-utilitarian discrete choice models for preference aggregation. Note that, among the statistical models based on a ground truth ranking R on alternatives, an interesting connection has been shown by Mallows [1957] between Young's model for binary relations (all pairwise preferences are independent and c is preferred to c' with probability $p > 1/2$ if c is preferred to c' in R) and Mallows' model for rankings (given a dispersion parameter $\theta \geq 0$, we have $Pr(R') \propto e^{-\theta \Delta_{KT}(R,R')}$, where $\Delta_{KT}(.,.)$ is the Kendall tau distance): sampling a ranking using Mallows' model is equivalent to sampling a binary relations R using Young's model with probability $p = e^{\theta}/(1 + e^{\theta})$ until a transitive binary relation R is obtained. This connection provides a simple but inefficient sampling method for Mallows' model. Efficient sampling and learning methods for Mallows' model have been proposed later [Doignon et al., 2004; Lu and Boutilier, 2014]. An interesting extension of the work presented here would therefore consist in determining if the k-wise Young's model can be related to a k-wise distance-based statistical model M for *rankings*, in the same manner as Young's model and Mallows' model, and to investigate effective sampling and learning methods for M from choice data.

Chapter 6

Detecting and taking Project Interactions into account in Participatory Budgeting

The aim of this chapter is to introduce models and algorithms for the Participatory Budgeting problem when projects can interact with each other. In this problem, the objective is to select a set of projects that fits in a given budget. Voters express their preferences over the projects and the goal is then to find a consensus set of projects that does not exceed the budget. Our goal is to detect such interactions thanks to the preferences expressed by the voters. Through the projects selected by the voters, we detect positive and negative interactions between the projects by identifying projects that are consistently chosen together. In presence of project interactions, it is preferable to select projects that interact positively rather than negatively, all other things being equal. We introduce desirable properties that utility functions should have in presence of project interactions and we build a utility function which fulfills the desirable properties introduced. We then give axiomatic properties of aggregation rules, and we study three classical aggregation rules: the maximization of the sum of the utilities, of the product of the utilities, or of the minimal utility. We show that in the three cases the problems solved by these rules are NP-hard, and we propose a branch and bound algorithm to solve them. We conclude the chapter by experiments.

6.1 Introduction

Participatory budgeting is a democratic process in which community members decide how to spend part of a public budget. Started in Porto Alegre, Brazil, in 1989, this process has spread to over 7,000 cities around the world, and has been used to decide budgets from states, cities, housing authorities, universities, schools, and other institutions[1]. The principle is the following one: the authorities of a given community

[1] https://www.participatorybudgeting.org/

(e.g. a city, or a university) decide to dedicate a budget l between projects proposed by the community members. Some community members (e.g. citizens, or students) propose projects, and write a proposal presenting their project and estimating its cost. All the community members are then asked to vote on the projects. There are several ways to collect voters' preferences. Due to its simplicity, the most widely used method is *approval voting*, in which voters are asked to approve or not each of the proposed projects. A variant of this method, called *knapsack voting* [Goel et al., 2019], and that we will consider in this chapter, asks the voters to approve projects up to the budget limit l: with knapsack voting, each voter is encouraged to give the set of projects that he or she would like to be selected, given the budget allocated. We start by reviewing existing work on participatory budgeting. Once the preferences of the voters have been expressed, the authorities use an algorithm which aggregates them and returns a set of projects (a bundle) of total cost at most l. In practice, e.g. in Warsaw, the projects are usually selected by decreasing number of votes.

6.1.1 Related work

Participatory budgeting is a very active field in computational social choice and numerous other algorithms have been proposed [Aziz and Shah, 2021; Aziz et al., 2017; Peters et al., 2021; Talmon and Faliszewski, 2019]. Several social welfare functions have been considered. The aim is usually either to maximize the minimal utility of a voter [Sreedurga et al., 2022]; to guarantee proportional representation to groups of voters with common interest [Peters et al., 2021; Aziz et al., 2017; Freeman et al., 2021], both aiming to return "fair" solutions; to maximize the sum of the utilities of the voters (utilitarian welfare); or to maximize the products of these utilities (Nash product) [Benade et al., 2021; Goel et al., 2019; Aziz and Shah, 2021]. In this chapter we are interested in optimizing three of the most classical criteria: the maximization of the sum of the utilities, of the product of the utilities, or of the minimal utility of the voters.

There are two main ways to define the utility of a voter. The first way defines the utility of a voter as the number of funded projects that he or she approves [Peters et al., 2021; Jain et al., 2020]. The second way defines the utility of a voter as the total amount of money allocated to projects approved by the voter [Goel et al., 2019; Talmon and Faliszewski, 2019; Freeman et al., 2021]. This second way of measuring the satisfaction of a voter is particularly relevant in the case of knapsack voting, where each voter can only approve a total budget of l: if a voter chooses to approve a project with a large cost at the expense of projects with smaller costs, it means that he or she prefers the large project to the smaller ones. We will consider this way to measure utilities.

Project interactions. Project interactions (also called synergies between projects) have been little explored so far. In almost all the papers, it is assumed that the utility of a bundle (a set of projects) for a given voter is the sum of the utilities of these projects (number of projects or total cost of these projects, depending on the model considered). In a recent paper, Fairstein et al. [Fairstein Roy and Kobi, 2023] do an empirical study of several voting formats, without considering synergies. However, they say in their

conclusion that "real voter utilities likely exhibit complementarities and externalities — a far cry from our utility proxies". Indeed, in practice, positive and negative synergies do exist. For example, two projects which are facilities that are planned to be built in the same location, or two projects which are very similar (e.g., two projects of playgrounds, or two skateboard parks) will have negative synergies: for a given voter, the utility of such two projects A and B will be smaller than the sum of the utilities of A and B. On the contrary, some projects are complementary and therefore have positive synergies. This is for example the case when a project aims to build a bicycle garage and another project aims to build a meeting place nearby. For a given voter, the utility of two projects A and B with positive synergies will be larger than the sum of the utilities of A and B. For two projects A and B which are independent, i.e., do not have positive neither negative synergies, the utility of the two projects A and B will be as usual the sum of the utilities of A and B.

To the best of our knowledge, there are only two papers which deal with projects interactions [Rey and Maly, 2023]. Jain et al. [2020] introduce a model in which they assume that the synergies between the projects are already known and are defined as a partition P over the projects. The projects which belong to a same set of the partition either have a substitution effect (i.e. a negative interaction) or a complimentary effect (a positive interaction). The authors define a utility function f such that $f(i)$ is the utility that a voter v gets from a set of the partition P if i projects from this set and approved by v are in the returned bundle. If f is concave (i.e. $f(i+1) - f(i) < f(i) - f(i-1)$) then projects in the same set of P have negative interaction; if f is convex (i.e. $f(i+1) - f(i) > f(i) - f(i-1)$) then projects in the same set of P have positive interaction. The utility of a voter is the sum of the utilities it has over the different sets of the partition. This model is the first one to consider project interactions. In a subsequent paper, Jain et al. [2021], assuming such an existing partition of the projects to interaction structures, take voter preferences to find such interaction structures (in their model, voters submit interaction structures, and the goal is to find an aggregated structure). Fairstein et al. [2021] also consider an underlying partition structure and ask the voters to give a partition of projects into groups of substitutes projects: in this setting only negative interactions are considered.

These papers are the first ones to consider and model project interaction. However, by partitioning the projects, their model cannot represent situations in which a project A can be both in positive interaction with a project B and in negative interaction with a project C, situation that we wish to take into account in this chapter. Furthermore, the authors of the previous mentioned papers assume that such a partition is either known [Jain et al., 2020], or computed thanks to the partitions of the projects asked to the voters [Jain et al., 2021; Fairstein et al., 2021], which can be a fastidious and complicated task for the voters.

6.1.2 Our approach to interaction detection

Our aim is not only to take into account interactions between projects into the utilities of voters, but also to detect the interactions through the preferences of the voters. De-

tecting such interactions through the votes is not possible if, as in [Jain et al., 2020], the voters use approval voting to give their opinions on the projects. Indeed, with approval voting, a voter tends to evaluate each project individually and to select the projects that he or she finds interesting according to his or her own criteria. Thus, it is likely that a voter who would like to see a playground built near his or her home will support all the playgrounds projects, even if such projects interact negatively. On the contrary, with knapsack voting, each voter is asked how he or she would spend the money if he or she had the opportunity to decide. In that context, it is unlikely that a voter selects projects that interact negatively, and on the contrary it is likely that projects that interact positively will be chosen. We think the best way to get reliable preferences (which express synergies) is to ask the following question to the voters: "How would you spend the budget if you could make the decision ?". Assuming most voters follow this recommendation, the synergies should be estimated quite accurately.

Detecting synergies can be done through the ballots approved by the voters, by looking at the frequencies of occurrence of groups of projects among the projects approved by the same voter, compared to the "expected" frequencies of this group of projects. If, for example, two projects A and B are selected together very often, we will deduce that they probably are in positive synergy. On the contrary, if two objects are never selected together, the synergy will be negative. Thus, by comparing the frequency of appearance of these projects A, B together with the product of the frequency of A and the frequency of B, we deduce synergies from the voters' choices.

Example 6.1.1: A simple example

Consider a budget $l = 9$ and 5 projects $\{A, \dots, E\}$ of costs $(2, 3, 3, 1, 1)$ (i.e. project A has cost 2, while project E has cost 1). Consider the following votes of 4 voters: $\{A, B, D, E\}$, $\{A, B, C\}$,$\{C, E\}$,$\{A, B, D\}$. Each project has been selected 2 or 3 times but projects A and B are always selected together, and projects C and D are never selected in a same ballot: we will deduce that projects A and B have a positive synergy while projects C and D have a negative synergy. Hence, whereas both bundles $\{A, B, C, E\}$ and $\{A, B, C, D\}$ are optimal for the utilitarian welfare, bundle $\{A, B, C, E\}$ is preferable because C and D have a negative synergy while C and E do not.

One could argue that two projects will not be chosen by the same voter because of the budget limit and not because they have a negative interaction. First, if the sum of the costs of these two projects is larger than l, then these two projects will anyway not be chosen in the returned bundle. Second, we examined the costs distribution of projects from the 247 real-world instances of knapsack voting from Pabulib [Stolicki et al., 2020]. These instances mainly have "small projects": the vector of costs of projects of these instances is in average: $(0.56, 0.18, 0.09, 0.06, 0.04, 0.02, 0.02, 0.01, 0.01, 0.01)$ – which means than 56% of the projects have a cost between 0 and 10% of the budget, 18% of the projects have a cost between 10 and 20% of the budget, and so forth. Additionally, on the same instances, the average (resp. median) total cost of

the projects selected by a voter represents 66% (resp. 75%) of the budget. This means that a majority of voters could have selected one more project, and this among most of the unapproved projects. Therefore, the overall low cost of the projects paired with the budget left unused in the votes suggests that if two projects are rarely selected together, it is usually not because of their costs.

Note that taking account of synergies between the projects may be interesting even if all the projects have the same cost, as shown by the following example.

Example 6.1.2: Projects of same cost

Let us consider a scenario with 12 voters, 8 projects of cost 1 and a budget of 4. Six voters select projects 1 and 2 plus a pair of projects in $\{5,6,7,8\}$, different for each one. The six other voters select projects 3 and 4 plus a pair in $\{5,6,7,8\}$, different for each one. Therefore, each project is selected exactly 6 times.

p_1	p_2	p_5	p_7	p_3	p_4	p_5	p_7
p_1	p_2	p_5	p_8	p_3	p_4	p_5	p_8
p_1	p_2	p_6	p_7	p_3	p_4	p_6	p_7
p_1	p_2	p_6	p_8	p_3	p_4	p_6	p_8
p_1	p_2	p_7	p_8	p_3	p_4	p_7	p_8
p_1	p_2	p_5	p_6	p_3	p_4	p_5	p_6

Figure 6.1: Example with $l = 4$

Without synergies, each bundle of 4 projects is optimal for the sum of the utilities. However, using synergies, we can detect that $\{1,2\}$ and $\{3,4\}$ are probably two strong pairs in comparison to the others. We can also see that the subset$\{1,2,3,4\}$ is never chosen as a whole which may indicate an antisynergy of the complete subset.

In the sequel, we will sometimes consider the k-additivity hypothesis, which means that there are synergies between groups of up to k projects. For example, with the 2-additivity hypothesis, we consider only interactions between pairs of projects, and not between more important groups of projects. In addition to the fact that it is realistic that synergies are important only for small values of k, considering this hypothesis will have repercussions on the complexity of our algorithms.

We conclude this introduction with an example showing that, in practice, positive (resp. negative) interactions may indeed be detected through the frequencies of co-occurrence of the projects in the same bundles.

Example 6.1.3: Real-life instances

By looking at real-world knapsack voting instances in the Pabulib library Stolicki et al. [2020], and by considering that there may be positive interactions between two projects (resp. negatively) when they are (resp. are not) chosen together, we identified several cases in which projects seem to interact positively or negatively[a]. For example, in Warsaw (poland_warszawa_2017_niskie-okecie.pb), two projects for the same neighbourhood, the first one being building a sport court and the second one building a playground, were chosen together less often than expected (given how often each one was individually approved). Our model says that they interact negatively, which makes sense, these projects being close to being susbtitutes. In another instance (poland_warszawa_2018_niskie-okecie.pb), two projects, the first one being building alleys in a park and the second one building public lightning in the same park, were consistently chosen together, which our model interpreted as a positive synergy. This also makes sense since these projects are clearly complementary.

[a] To be precise, to detect these interactions, we used the utility function u_M presented in Section 4, by considering 2-additivity hypothesis.

6.1.3 Overview of our results

We tackle the indivisible participatory budgeting problem, with knapsack voting, by considering that projects are not independent, but that there may have positive and negative synergies between them.

- In Section 6.3, we propose desirable properties for utility functions in presence of project interactions.
- In Section 6.4 we present a particular utility function, derived from Möbius transforms and denoted by u_M, that fulfills the axioms defined on the previous section.
- In Section 6.5, we study axiomatic properties of aggregation rules. We consider in particular three aggregation rules, which either maximize the sum of the utilities, the product of the utilities, or the minimal utility of the voters.
- In Section 6.6 we show that these rules solve NP-hard problems, and that synergies make the problem harder since it is NP-hard to maximize the sum of the utilities with unit size projects when there are synergies, whereas this problem can be solved easily without synergies. These results hold for utility function u_M but also for other very general synergy functions.
- In Section 6.7, we propose an exact branch and bound algorithm which can be used with any utility function, and we conclude with an experimental evaluation.

6.2 Preliminaries

We use the general framework for approval-based participatory budgeting proposed by Talmon and Faliszewski [2019] . A *budgeting scenario* is a tuple $E=(A,V,c,l)$ where $A=\{a_1,\dots,a_n\}$ is a set of *n projects*, or items, and $c:A\to\mathbb{N}$ is a *cost function*: $c(a)$ is the cost of project $a\in A$ – abusing the notation, given a subset S, we denote by $c(S)$ the total cost of S: $c(S)=\sum_{a\in S}c(a)$. The budget limit is $l\in\mathbb{N}$. The set $V=\{v_1,\dots,v_v\}$ is a set of v voters. Each voter $v_i\in V$ gives a set of approved projects $A_i\subseteq A$, containing a set of projects that she approves of and such that $c(A_i)\leq l$. We denote by $\mathscr{E}^A$ the set of all possible budgeting scenarios having A as a set of projects.

A *budgeting method* r is a function taking a budgeting scenario $E=(A,V,c,l)$ and returning a *bundle* $B\subseteq A$ such that $c(B)\leq l$. We consider that a budgeting method always returns a unique bundle (we can use usual tie-breaking techniques to handle instances with several winning bundles). The winning bundle for a budgeting scenario E is denoted by $r(E)$. A project is *funded* if it is contained in the winning bundle B. Given a bundle B and a voter v_i with her approval set A_i, we denote by $B_i=A_i\cap B$ the set of projects common to A_i and B.

Utility functions. A *utility function* $u:2^A\to\mathbb{R}_0^+$ is a set function which gives a value to each subset of items. A *linear utility function* is such that that the value of a bundle B is the sum of the utilities of its items: $u(B)=\sum_{a\in B}u(\{a\})$. The *overlap utility* function, introduced for the knapsack voting by Goel et al. [2019] , considers that the utility of a bundle B is the sum of the costs of the projects in B: $f(A_i,B)=\sum_{a\in B_i}c(a)$

Satisfaction functions. A *satisfaction function* f is a function $f:2^A\times 2^A\to\mathbb{R}$, which, given a voter $v_i\in V$ and a bundle $B\subseteq A$, returns the satisfaction that v_i gets from B. Given a selected bundle B and a utility function u, we will consider that the satisfaction of voter v_i from bundle B is the utility of $A_i\cap B$: $f(A_i,B)=u(B_i,E)$.

.

The utility function aims at associating to each possible bundle an evaluation of its quality. The satisfaction function indicates, given two sets of projects, the first one being the preference of a voter and the other being a potential solution, how satisfied the voter is given the solution.

In the sequel, we will consider generalizations of the overlap utility function that take into account potential projects interactions. Since these function may depend on the instance, we will denote the utility of the subset B_i as: $u(B_i,E)$.

Aggregating criterion. In order to obtain a solution satisfying the whole population, we study the three most classical aggregating methods: the sum ($\sum$), the product ($\prod$) and the minimum (min) of the satisfactions of the voters. We denote by $\alpha-r_u$ the budgeting method returning the α aggregation of the utility function u, where $\alpha\in\{\sum,\prod,\min\}$. This rule returns an optimal bundle of the associated maximization optimization problem, that we will call problem PB-Max–α–u (e.g. problem PB-Max–$\sum$–u consists in computing a bundle maximizing the sum of the utilities of the voters when the utility function used is u). These three aggregating concepts rely on different ideas of the collective satisfaction. The sum criterion maximizes the average satisfaction of a

voter. The minimum tries to satisfy as much as possible the least satisfied voter – this is an egalitarian view. Finally the product stands in between the two previous criteria: the product is very penalized by the presence of very low utility values, however, it still takes into account the larger values. This last criterion has been the favourite of the voters in an experimental study Rosenfeld and Talmon [2021]. These three criteria share several axiomatic and computational properties, as we will see in the following sections.

We will now discuss how to obtain satisfactory utility functions and how mathematical properties on such functions impact the budgeting methods.

6.3 Axioms for utility functions

In this section, we define desirable properties for utility functions in the presence of synergies.

The first property states that the utility of a single project should be proportional to its cost. This property is fulfilled by the *overlap utility* function Goel et al. [2019]. It is particularly meaningful in knapsack voting: since there is a budget constraint on the approval set of the voters, the approval of a project is done with full knowledge of its cost and the approval of a costly project is done at the expense of the budget for other projects.

Definition 6.3.1: Cost consistency (CC)

Given a budgeting scenario $E = (A, V, c, l)$, a utility function $u^E : 2^A \times \mathcal{E}^A \to \mathbb{R}_0^+$ is *cost consistent* if there exists a constant k such that for each project a in A, we have $u(\{a\}E,) = k \cdot c(a)$.

The factor k allows normalization. This property insures that the utility function follows the cost function for the sets containing only one project. Note that this property is only determining the behaviour of the utility function for subset of size one, it does not indicate anything about the remaining subsets.

The following classical property ensures that the utility of a set does not decrease when the set grows. This ensures that we cannot decrease a voter satisfaction by adding a project that she selected.

Definition 6.3.2: Super-set monotonicity (SSM)

Given a budgeting scenario $E = (A, V, c, l)$, a utility function u^E is *super-set monotone* if for any subset X_{sub} and X such that $X_{sub} \subset X$, we have $u(X_{sub}, E) \leq u(X, E)$.

Relaxing the *neutrality* principle Brandt et al. [2016], the next property states that two similar projects should be treated equally. Given a set S, we denote by $S_{(a_i \leftrightarrow a_j)}$ the set obtained from S by swapping a_i and a_j: a_i (resp. a_j) belongs to $S_{(a_i \leftrightarrow a_j)}$ if and only

a_j (resp. a_i) belongs to S, and each project $a_k \notin \{a_i, a_j\}$ belongs to $S_{(a_i \leftrightarrow a_j)}$ if and only if a_k belongs to S. We also denote by $E_{(a_i \leftrightarrow a_j)}$ the budgeting scenario obtained from E by swapping the approval of the projects a_i and a_j: a voter v_l approves a_i (resp. a_j) in $E^{(a_i \leftrightarrow a_j)}$ if and only if v_l approves a_j (resp. a_i) in E.

Definition 6.3.3: Cost-Aware Neutrality (CAN)

Given a budgeting scenario $E = (A, V, c, l)$, and two projects a_i and a_j of A such that $c(a_i) = c(a_j)$, a utility function u^E is *cost-aware neutral* if $u(S, E) = u(S_{(a_i \leftrightarrow a_j)}, E_{(a_i \leftrightarrow a_j)})$.

Note that this property is inspired by the *Processing Time Aware neutrality* property (see Definition 3.3.2) used in the collective schedules model: this property ensures that two tasks of equal processing time are treated equally. We restrict our analysis to cost-aware neutral utility functions since no pair of projects with the same cost should be treated differently.

If a subset of item is consistently chosen as a whole, then the utility it brings should be higher than the sum of the utilities of the items. On the opposite side, if projects are never chosen together, then the utility of the whole subset should be lower than the sum of utilities of the items. The third axiom states that the more a subset appear together, the more its utility should increase, everything else being equal.

The next property, the *effect of positive synergies* ensures that the utility of subsets of projects that always appear together is larger than the sum of the utilities of its components.

Definition 6.3.4: Effect of positive synergies (PS)

Given a budgeting scenario $E = (A, V, c, l)$, a utility function u^E fulfills the *effect of positive synergies* (resp. *strong effect of positive synergies*) property if, for each subset S in 2^A such that for each voter v_i we have either $S \subseteq A_i$ or $S \cap A_i = \emptyset$ and such that there exists $v_k \in V$ with $S \subseteq A_k$, then $u(S, E) \geq \sum_{a \in S} u(\{a\}, E)$ (resp. $u(S, E) > \sum_{a \in S} u(\{a\}, E)$).

The next property ensures that the utility of subsets of projects that never appear together is smaller than (or equal to) the sum of the utilities of its components.

Definition 6.3.5: Effect of negative synergies (NS)

Given a budgeting scenario $E = (A, V, c, l)$, a utility function u^E fulfills the *effect of negative synergies* (resp. *strong effect of negative synergies*) property if, for each subset S in 2^A such that for each voter $v_i \in V$ we have $|S \cap A_i| \leq 1$, then $u(S, E) \leq \sum_{a \in S} u(\{a\}, E)$ (resp. $u(S, E) < \sum_{a \in S} u(\{a\}, E)$).

The next property states that the utility of a subset should increase with the number of appearances of the whole subset in the preferences of voters with respect to a solution in which the number of approvals of the items is the same but the items are not approved by the same voters.

Definition 6.3.6: Regrouping monotonicity (RM)

Let $E = (A, V, c, l)$ be a budgeting scenario, $S \subseteq A$ be a subset such that $c(S) \leq l$, and let v_i and v_j be two voters of V such that $S \subseteq (A_i \cup A_j)$, $A_i \cap A_j = \emptyset$, $S \not\subseteq A_i$, $S \not\subseteq A_j$, and $c(A_i \cup A_j \setminus S) \leq l$. Let $V_S = V \cup \{v_k, v_l\} \setminus \{v_i, v_j\}$, where v_k and v_l are two voters who are not in V and such that $A_k = S$ and $A_l = (A_i \cup A_j) \setminus S$. Let $E' = (A, V_S, c, l)$ be a budgeting scenario. A utility function u^E satisfies *regrouping monotonicity* if $u(S, E) < u(S, E')$.

We can also imagine creating utility functions thanks to prior knowledge on the projects, however in such cases, it is possible that the last three properties are violated.

In the following section, we propose a utility function taking synergies into account, and that fulfills the properties that we have introduced in this section.

6.4 A utility function taking synergies into account

6.4.1 A function using Möbius transforms: u_M

Möbius transforms Rota [1964] are a classical tool for measuring synergies in sets of items. Given a utility function $u : 2^A \to \mathbb{R}_0^+$, the Möbius transform of a subset S, denoted by $m(S)$, expresses the level of synergy between the items in S. For a set $S = \{a, b\}$ of two elements, and if $u(\emptyset) = 0$, we have $m(S) = u(\{a, b\}) - u(\{a\}) - u(\{b\})$. More generally, the Möbius transform of a set S is calculated as follows:

$$m(S) = \sum_{C \subseteq S} (-1)^{|S \setminus \{C\}|} u(C)$$

The Möbius transform $m(S)$ expresses the level of synergy between the elements of the subset S. If it is negative, this indicates a negative interaction between the elements of S; if it is null, this indicates independence of the elements; and if it is positive, this indicates positive interaction between the elements.

Example 6.4.1: Computing Möbius transforms

Let us consider a utility function u over a set of items $\{1, 2, 3\}$. The utilities are as follows:

C	$\emptyset$	$\{1\}$	$\{2\}$	$\{3\}$	$\{1,2\}$	$\{1,3\}$	$\{2,3\}$	$\{1,2,3\}$
u(C)	0	0.2	0.4	0.5	0.5	0.7	0.8	1

Let us compute the Möbius transform of subset {1,2}

$$\begin{array}{rcl} m(\{1,2\}) & = & (-1)^0 u(\{1,2\}) + (-1)u(\{1\}) + (-1)u(\{2\}) + (-1)^2 u(\emptyset) \\ m(\{1,2\}) & = & u(\{1,2\}) - u(\{1\}) - u(\{2\}) + u(\emptyset) \\ m(\{1,2\}) & = & -0.1 \end{array}$$

We find a negative Möbius transform, indicating a negative interaction between elements 1 and 2.

A utility function from Möbius transforms. It is not only possible to find the Möbius transforms from a utility function, it is also possible to build a utility function from the Möbius transforms thanks to the following expression Rota [1964]:

$$u(S) = \sum_{C \subseteq S} m(C)$$

The utility of a subset S is then the sum of Möbius transforms of its elements – which is also the sum of their utilities – plus the Möbius transforms of the subsets included in S, representing their level of positive and negative synergies. Therefore, if we can measure the level of synergy of each subset, we can build a utility function.

We use a statistical approach in order to infer synergies from the preferences. Let $r(S,V)$ be the rate of occurrence of a subset S in the approval sets of voters in V (i.e. the ratio between the number of voters who selected all the projects of S, and the total number of voters). The expected rate of occurrence of a whole subset S if all of its elements were perfectly independent (ignoring possible cost constraints), would be $\prod_{a \in S} r(\{a\},V)$, the product of the appearance rates of each the elements of S. We use $(r(S,V) - \prod_{a \in S} r(S,V))$ as a marker of synergy. If it is null, then the projects appear as independent in the preferences. If it is positive, then the subset appears more frequently than expected if the preferences were random, indicating a positive interaction. If it is negative, it indicates on the contrary a negative interaction.

We set $u(\emptyset) = m(\emptyset) = 0$ and, to insure cost consistency, we set $u(\{a\}) = m(\{a\}) = c(a)$. Since $(r(S,V) - \prod_{a \in S} r(S,V))$ has a range included in $[-1;1]$, we multiply this difference by the cost of the subset. We obtain: $m(S,E) = (r(S,V) - \prod_{a \in S} r(S,V)) \cdot c(S)$. We finally adapt this definition so that the utility function obtained from the Möbius transforms fulfills super-set monotonicity:

$$m(S,E) = \begin{cases} 0 & \text{if } S = \emptyset \\ c(a) & \text{if } S = \{a\} \\ \max\{\left(r(S,V) - \prod\limits_{a \in S} r(S,V)\right)c(S), & \\ \quad \max\limits_{a \in S}\left(-\sum\limits_{C \subset S, a \in C} m(C,E)\right)\} & \text{otherwise} \end{cases} \tag{6.1}$$

The intuition is the following one: $\sum_{C \subset S, a \in C} m(C,E)$ is the sum of the Möbius transforms of subsets containing project a. By ensuring that the Möbius transform of S is larger than or equal to the opposite of this sum, we ensure that the utility of S is not

smaller than the utility of $S \setminus \{a\}$.

Note that guaranteeing super-set monotonicity implies that we know the Möbius transforms value of smaller sets. The utility function u_M is as follows:

$$u_M(S,E) = \sum_{C \subseteq S} m(C,E) \tag{6.2}$$

6.4.2 Properties of u_M, and remarks on its computation

We use Equation 6.2 to determine the utility of a bundle with function u_M. Because of its recursive nature, we compute first, as a preprocessing step, the utility of singletons, then pairs, then triplets and so forth. Determining the utilities in this way costs up to 2^n (since there are 2^n subsets) times $v \times n$ operations (since determining the appearance rate of a subset costs $v \times n$ operations). This calculation is much faster with the k-additivity hypothesis, since the Möbius transform associated to any subset of size larger than k is then 0. Therefore, with such an hypothesis, we simply need to know the Möbius transforms of the subsets of size at most k: the preprocessing part is polynomial if k is a constant.

We now state that the utility function u_M fulfills all the desirable properties stated in Section 3. This is true even with the k-additivity assumption, for any value of k.

Proposition 6.4.1: Axiomatic properties of u_M

The utility function u_M fulfills *cost consistency*, *super-set monotonicity*, the *effect of positive synergies* property, the *effect of negative synergies* property, *regrouping monotonicity* and *cost aware neutrality*. It also fulfills the strong effect of positive synergies property if for each project a, there is at least one voter who does not select a.

Proof.

- **Cost consistency.** As defined in equation 6.1, the Möbius transform of a single project is its cost. Since the utility of a single project is its Möbius transform, assuming the Möbius transform and the utility of the empty set is 0, the utility $u_M(\{a\},E) = c(a)$ for any project a. Therefore u_M fulfills cost consistency.

- **Super-set monotonicity.** As detailed earlier, the super-set monotonicity of the function u_M is insured by the definition of the Möbius transform. As a reminder, to fulfill super-set monotonicity, the function u_M has to verify the following property: $u_M(S,E) \geq u_M(S \setminus \{a\},E)$ for all $S \in 2^A \setminus \emptyset$ and all $a \in S$. Since, by definition, we have $m(S,E) \geq -\sum_{C \subset S, a \in C} m(C,E)$ for all $a \in S$, it means that $\sum_{C \subseteq S, a \in C} m(C,E) \geq 0$ for all $a \in S$ and therefore $\sum_{C \subseteq S} m(C,E) \geq \sum_{C' \subseteq S \setminus \{a\}} m(C',E)$ for all $a \in S$. By definition of u_M, this means $u_M(S,E) \geq u_M(S \setminus \{a\},E)$ for all $a \in S$. The utility function u_M fulfills super-set monotonicity.

- **Effect of positive synergies.** Let S be a subset of projects such that for any $a \in S$ and any $v_i \in V$, $a \in A_i \implies S \subseteq A_i$ and such that $\exists v_k \in V$ with $a \in A_k$. In other words, if a voter approves of one of the elements of S, she approves of all projects in S and such a voter exists in V. For such a subset, the value $r(S,V) - \prod_{a \in S} r(\{a\},V)$ is equal to $r(S,V) - r(S,V)^{|S|}$. Since the $r(S,V)$ value is larger than 0 and smaller than or equal to 1, the difference $r(S,V) - r(S,V)^{|S|}$ is positive or null. This means that $(r(S,V) - \prod_{a \in S} r(\{a\},V))c(S)$ is positive or null, this means that the Möbius transform of S is positive or null, $m(S,E) \geq 0$. The same remark can be said about all subset $C \subseteq S$ since all the projects of S are only selected together. Therefore, we have $\sum_{C \subseteq S, |C| \geq 2} m(C) \geq 0$. By definition, the Möbius transforms of the single projects are their cost, we then have: $\sum_{C \subseteq S} m(C) \geq \sum_{a \in S} c(a)$, and consequently: $u_M(S,E) \geq 0$. The utility function u_M fulfills the effect of positive synergies property. If we suppose that for each project a, there is at least one voter who does not select a, then for each subset S, there is at least one voter who does not select S. Then $r(S,V)$ is smaller than 1 and the difference $r(S,V) - r(S,V)^{|S|}$ is strictly positive. In this case, u_M fulfills the strong effect of positive synergies property.

- **Effect of negative synergies.** Let S be a subset of projects such that for any $a \in S$ and any $v_i \in V$, $a \in A_i \implies S \cap A_i = \{a\}$. In other words, if a voter selects an element a of S, then it is the only element of S she selects. For such a subset, the value $r(S,V) - \prod_{a \in S} r(\{a\},V)$ is negative or null, since S never appears but the element of S can appear individually. This is true for any subset $C \subseteq S$ such that $|C| \geq 2$. When summing the Möbius transforms all these subsets included in S,we will have the Möbius transforms of singleton that are positive and equal to the cost the project and then null or negative values. This means that the overall utility of S cannot be greater than the sum of the utility of its components. Therefore, u_M fulfills the effect of positive synergies property.

- **Regrouping monotonicity.** Let $E = (A,V,c,l)$ be a budgeting scenario and let $S \in 2^A$ be a subset of projects with $c(S) \leq l$. Let v_i and v_j be two voters in V such that $A_i \cap A_j = \emptyset$, $S \subseteq A_i \cup A_j$ and $c(A_i \cup A_j \setminus S) \leq l$. We consider voters v_k and v_l with $A_k = S$ and $A_l = A_i \cup A_j \setminus S$, and a set of preferences $V_S = V \cup \{v_k, v_l\} \setminus \{v_i, v_j\}$. Let $E' = (A, V_S, c, l)$ be a budgeting scenario. In V_S any subset $C \subseteq S$ appears at least as often than in V and any project appears as much in V_S than in V, therefore for any $C \subseteq S$, $r(C,V_S) \geq r(C,V)$, consequently $m(C,E) \geq m(C,E)$ and $u_M(C,E') \geq u_M(C,E)$. Since $u_M(C,E') \geq u_M(C,E)$, for all $C \subseteq S$ and $r(S,V_S) > r(S,V)$ we see that $m(S,E') > m(S,E)$ and therefore $u_M(S,E',>)u_M(S,E)$ from equation 6.2. Thus, u_M fulfills regrouping monotonicity.

- **Cost aware neutrality.** Given a budgeting scenario $E = (A,V,c,l)$, let $E_{(a_i \leftrightarrow a_j)} = (A, V_{(a_i \leftrightarrow a_j)}, c, l)$ be the budgeting scenario obtained from E by swapping the approval of two projects a_i and a_j such that $c(a_i) = c(a_j)$. For a given subset S, let $S_{(a_i \leftrightarrow a_j)}$ be the subset obtained from S by swapping a_i and a_j, i.e. $S_{(a_i \leftrightarrow a_j)}$ contains

the same projects than S except for a_i and a_j, if S contains a_i, $S^{(a_i \leftrightarrow a_j)}$ contains a_j and if S contains a_j, $S^{(a_i \leftrightarrow a_j)}$ contains a_i. Since $c(a_i) = c(a_j)$, and since for any subset C, $r(C,V) = r(C_{(a_i \leftrightarrow a_j)}, V_{(a_i \leftrightarrow a_j)})$, we can see that $m(C,E) = m(C_{(a_i \leftrightarrow a_j)}, E_{(a_i \leftrightarrow a_j)})$ and, because of equation 6.2 that $u_M(C,E) = u_M(C_{(a_i \leftrightarrow a_j)}, E_{(a_i \leftrightarrow a_j)})$. The utility function u_M fulfills cost aware neutrality.

□

6.5 Axioms for budgeting methods

In this section we discuss some axiomatic properties of the different aggregation rules, relying on the properties of the utility function used. We try, when it is possible, to have general results relying on the properties introduced in section 6.3 instead of on specific utility functions. We start with the *inclusion maximality* axiom Talmon and Faliszewski [2019], also known as *exhaustiveness* Aziz et al. [2017]. This axiom states that if a bundle B is a winning bundle according to a budgeting method r, then it is either exhaustive, in the sense that it is impossible to add a project without exceeding the budget limit, or any of its feasible superset is also a winning bundle.

Definition 6.5.1: Inclusion maximality (IM) [Talmon and Faliszewski, 2019].

A budgeting method $\mathscr{R}$ satisfies *inclusion maximality* if for any budgeting scenario $E = (A, V, c, l)$ and each pair of feasible bundles B and B' such that $B' \subset B$, it holds that $B' \in \mathscr{R}(E) \implies B \in \mathscr{R}(E)$.

Proposition 6.5.1: Super-Set Monotonicity and Inclusion Maximality

If a utility function u fulfills super-set monotonicity, then the budgeting method $\alpha\text{–}r_u$, for $\alpha \in \{\sum, \prod, \min\}$ fulfills inclusion maximality.

Proof. Let u be a utility function satisfying super-set monotonicity and α–r_u a budgeting method maximizing either the sum, the product or the minimum over all the voters utilities. For any voter v_i, and for any pair of feasible bundles B and B' such that $B' \subset B$, we call B_i and B'_i the common subsets between A_i and B and A_i and B' respectively. Since $B' \subset B$, we have $B'_i \subseteq B_i$. Since both the sum, product and the minimum utility of the voters are non decreasing with the utility of individual voters, if B' is optimal for any of these rules, then B is also optimal. The budgeting method $\alpha\text{–}\mathscr{R}_u$ thus satisfies inclusion maximality. □

Note that when a budgeting method is resolute, meaning that it returns only one winning bundle, this axioms requires that the only winning bundle is exhaustive. This means that if we use tie-breaking mechanism to choose a solution among several optimal ones, they should select an exhaustive solution. Note that it can be easily obtained by adding projects greedily from an optimal solution that is not exhaustive.

The next two axioms focus on robustness, especially when projects have a composite structure (i.e. a large project can be divided into several small projects, or small projects merged into one large project).

Definition 6.5.2: Splitting monotonicity (SM) [Talmon and Faliszewski, 2019].

A budgeting method r satisfies *splitting monotonicity* if for every budgeting scenario $E = (A, V, c, l)$, for each $a_x \in r(E)$ and each budgeting scenario E' which is formed from E by splitting a_x into a set of projects A' such that $c(A') = c(a_x)$, and such that the voters which approve a_x in E approve all items of A' in E' and no other voters approve items of A', it holds that $r(E') \cap A' \neq \emptyset$.

Proposition 6.5.2: Splitting monotonicity - u_M

For $\alpha \in \{\sum, \prod, \min\}$, the budgeting method $\alpha - r_{u_M}$ fulfills splitting monotonicity.

Proof. Let $E = (A, V, c, l)$ be a budgeting scenario, and let B be the bundle returned by $\alpha - r_u$ for E. Let a_x be a project in the bundle $\alpha - r_u(E)$. Let $E' = (A', V', c', l)$ be the budgeting scenario formed from E in which a_x is divided into a set X' of projects such that $c(X') = c(a_x)$. Voters in V' are the same than in V except that any voter approving project a_x in V approves all the projects of X' in V'. The bundle B maximizes the objective of the rule $\alpha - r_u$. Note that the utility of any subset that does not contain a_x is identical for E and E', and brings the same satisfaction to each voter: it therefore has the same quality regarding the aggregating criterion of $\alpha - r_u$for E and E'. Let B' be the bundle B in which a_x is replaced by all the projects in X'. Bundle B' is a feasible solution for E'. Any voter v'_i in V' has a corresponding voter v_i in V. We recall that B_i denotes the set of projects that are common between a bundle B and the approval set of a voter v_i. There are two cases:

- $X' \cap B'_i = \emptyset$: in this case, $u_M(B'_i, E') = u_M(B_i, E)$
- $X' \subseteq B'_i$: we have $c(B'_i) = c(B_i)$ and $r(B'_i, V) = r(B_i, V)$. Additionally, we have $\prod_{b' \in B'_i} r(b', V') \leq \prod_{b \in B_i} r(b, V)$, since the rates do not change but the number of projects is larger in B'_i than in B_i. This is also true for any subset $C \subseteq B'_i$ such that $X' \subseteq C$. Therefore, because of the super-set monotonicity property, we have $u_M(B'_i, E') \geq u_M(B_i, E)$.

Overall, B' is at least as good as any solution containing no element of X', meaning that either B' maximizes the rule criterion or a solution containing at least one project in X' does. Therefore there is a a in X' such that a is in $\alpha - r_u(E')$: the $\alpha - r_u$ rule fulfills splitting monotonicity. □

Definition 6.5.3: Merging monotonicity (MM) Talmon and Faliszewski [2019]

A budgeting method r satisfies *merging monotonicity* if for each budgeting scenario $E = (A, V, c, l)$, and for each $A' \subseteq r(E)$ such that for each $v_i \in V$ we either have $A_i \cap A' = \emptyset$ or $A' \subseteq A_i$ – i.e. a voter approves either all projects from A' or none – it holds that $a \in r(E')$ for $E' = (A \setminus \{A'\} \cup \{a\}, V', c', l)$, $c'(a) = c(A')$, and each voter $v_i \in V$ for which $A' \subseteq A_i$ in E approves a in E', and no other voter approves a.

Proposition 6.5.3: Effect of positive synergies and merging monotonicity

Let $\alpha \in \{\sum, \prod, \min\}$. If a utility function u fulfills the strong effect of positive synergy property and cost consistency, then the budgeting method $\alpha - r_u$ does not fulfill merging monotonicity.

Proof. • Case where $\alpha = \Sigma$. Let T be an even positive integer. Let us consider a budgeting scenario $E = (A, V, c, l)$ with $A = x_1, x_2, y$, $c(x_1) = c(x_2) = T/2$, $c(y) = T$ and $l = T$. There are two types of voters in V. There are v_1 voters of the first type, and each one of them approves x_1 and x_2. There are v_2 voters of type 2, and they all approve y as shown in Figure 6.2. By cost consistency, we know that there exists a constant k such that $u(x_1, E) = u(x_2, E) = kT/2$ and $u(y, E) = kT$. By strong effect of positive synergies, we have $u(\{x_1, x_2\}, E) > u(\{x_1\}, E) + u(\{x_2\}, E)$ and consequently $u(\{x_1, x_2\}, E) > kT$. Let $\epsilon = u(\{x_1, x_2\}, E) - kT > 0$. The bundle $\{x_1, x_2\}$ has a total utility of $v_1(kT + \epsilon)$, the bundle $\{y\}$ has a utility of $v_2 kT$. If $v_1 kT - v_2 kT + v_1 \epsilon > 0$, then $\{x_1, x_2\}$ is the best bundle.

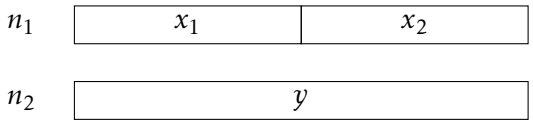


Figure 6.2: First budgeting scenario E

We now consider $E' = (A', V', c', l)$ another budgeting scenario such that $A' = \{x, y\}$, $c'(x) = c(x_1) + c(x_2) = c'(y) = c(y) = T$. In V' we create v_1 voters approving x and v_2 voters approving y. Note that the budgeting scenario E' is similar to E except that the projects x_1 and x_2 have merged in a project of size T. Because of cost consistency, we have $u(\{x\}, E') = u(\{y\}, E') = kT$. Therefore the bundle $\{x\}$ has a total utility of $v_1 kT$ and the bundle $\{y\}$ still has a utility of $v_2 kT$. If $v_1 < v_2$, $\{y\}$ is the winning bundle.

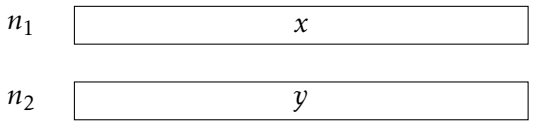


Figure 6.3: Second budgeting scenario E'

By setting $v_1 = \lceil 2kT/\epsilon \rceil$ and $v_2 = v_1 + 1$, $\{x_1, x_2\}$ is the winning bundle for E and $\{y\}$

is the winning bundle for E', giving us an instance for which the $\sum\text{–}r_u$ rule does not fulfill merging monotonicity.

• Case where $\alpha \in \{\prod, \min\}$. Let us consider a budgeting scenario $E = (A, V, c, l)$ with $A = \{x_1, x_2, x_3, x_4, y\}$, $c(x_1) = c(x_2) = c(x_3) = c(x_4) = (T-2)/4$, $c(y) = T/2 + 1$ with T an even integer and $l = T$. There are two voters in V: the first one approves of x_1, x_2, x_3 and x_4, the second one approves of y.

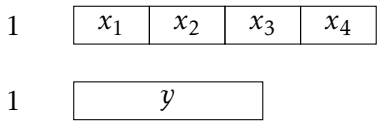


Figure 6.4: First budgeting scenario E

When maximizing either the min utility or the product, for any utility function u fulfilling cost consistency and the strong effect of positive synergies, the winning bundle will be y plus two projects x_i and x_j. Let us assume, without loss of generality that the projects x_1 and x_2 are part of the winning bundle. Let $E' = (A', v', c', l)$ be a budgeting scenario formed from E in which projects x_1 and x_2 are merged into one project X of cost (and therefore utility) $T/2 - 1$.

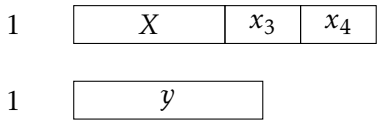


Figure 6.5: Second budgeting scenario E'

The utilities of x_3 and x_4 are still $(T-2)/4$. By strong superadditivity of groups, the utility of $\{x_3, x_4\}$ is strictly larger than $2(T-2)/4$ and strictly larger than the utility of X consequently. Therefore the winning bundle for E' is $\{y, x_3, x_4\}$. Since X is not in this bundle, merging monotonicity is not fulfilled. □

The next axiom states that if the cost of a funded project decreases, it is still guaranteed to be funded. It is easy to see that this axiom is not compatible with the cost consistency property.

Definition 6.5.4: Discount Monotonicity (DM) Talmon and Faliszewski [2019]

A budgeting method r satisfies *discount monotonicity* if for each budgeting scenario $E = (A, V, c, l)$ and each item $b \in r(E)$, it holds that $b \in r(E')$ for $E' = (A, V, c', l)$ where for each item $a \neq b$, $c'(a) = c(a)$ and $c'(b) = c(b) - 1$.

Proposition 6.5.4: Cost Consistency and Discount Monotonicity

Let $\alpha \in \{\sum, \prod, \min\}$. If a utility function u fulfills cost consistency, then the budgeting method $\alpha\text{–}r_u$ does not fulfill discount monotonicity.

Proof. Let us consider a budgeting scenario $E = (A, V, c, l)$ with $A = x_1, x_2, y$, such that $c(x_1) = 4, c(x_2) = 3$, $c(y) = 4$ and $l = 8$. There are two voters in V: the first one approves x_1 and x_2, and the second one approves y. When maximizing either the $\sum$, the min or the $\prod$ of utilities, for any utility function u fulfilling cost consistency, the winning bundle will be y plus project x_1. Let $E' = (A, V, c', l)$ be a budgeting scenario formed from E in which project x_1 now has a cost of 2 instead of 4. The winning bundle is now $\{y, x_2\}$. The cost of project x_1 was reduced and it was removed from the winning bundle, therefore discount monotonicity is not fulfilled. □

This last axiom states that any funded project in a winning bundle is still funded when the budget limit increases.

Definition 6.5.5: Limit Monotonicity (LM) Talmon and Faliszewski [2019]

A budgeting method r fulfills *limit monotonicity* if for each pair of budgeting scenarios $E = (A, V, c, l)$ and $E' = (A, V, c, l + 1)$ with no item which costs exactly $l + 1$, it holds that $a \in r(E) \implies a \in r(E')$.

Proposition 6.5.5: Cost consistency and Limit Monotonicity

Let $\alpha \in \{\sum, \prod, \min\}$. If a utility function u fulfills cost consistency, then the budgeting method $\alpha\text{-}r_u$ does not fulfill limit monotonicity.

Proof. • Case where $\alpha = \sum$: Let us consider a budgeting scenario $E = (A, V, c, l)$ with $A = x_1, x_2, x_3$, $c(x_1) = 2, c(x_2) = 5$, $c(x_3) = 6$ and $l = 6$. There are three voters in V: the first one approves of x_1, the second one approves of x_2 and the third one approves of x_3. When maximizing the sum of utilities, for any utility function u fulfilling cost consistency, $\{x_3\}$ will be the winning bundle. Let $E' = (A, V, c, l')$ be a budgeting scenario formed from E but such that the budget limit l' is now 7 instead of 6. The winning bundle is now $\{x_1, x_2\}$. The budget limit was increased and project x_3 was removed from the winning bundle, therefore limit monotonicity is not fulfilled.

• Case where $\alpha \in \{\min, \prod\}$: Let us consider a budgeting scenario $E = (A, V, c, l)$ with $A = x_1, x_2, x_3$, $c(x_1) = 1, c(x_2) = 2$, $c(x_3) = 3$ and $l = 4$. There are two voters in V: the first one approves of x_1 and x_2, the second one approves of x_3. When maximizing either the min or the product of utilities, for any utility function u fulfilling cost consistency, $\{x_1, x_3\}$ will be the winning bundle. Let $E' = (A, v, c, l')$ be a budgeting scenario formed from E but such that the budget limit l' is now 5 instead of 4. The winning bundle is now $\{x_2, x_3\}$. The budget limit was increased and project x_1 was removed from the winning bundle, therefore limit monotonicity is not fulfilled. □

From Proposition 6.4.1 and propositions from Section 6.5, we get the following corollary.

Corollary 6.5.1: Axiomatic properties $\alpha - r_{u_M}$

The rules $\alpha - r_u$ for $\alpha \in \{\sum, \min, \prod\}$ and $u = u_M$ fulfill *inclusion maximality* and *splitting monotonicity*. They do not fulfill *merging monotonicity*, *discount monotonicity* and *limit monotonicity*.

6.6 Complexity

We show in this section that, for each $\alpha \in \{\sum, \prod, \min\}$, problem PB-Max–$\alpha - u$ is NP-hard when there are synergies, and this even if all the projects have unitary cost and when the utility function fulfills only very mild conditions. This shows that synergies add complexity, since problem PB-Max–$\sum$–u is polynomially solvable when projects have the same cost and without synergies (i.e. when the function u is linear). Indeed, without synergies and with unitary size projects, selecting the projects by decreasing number of votes maximizes the sum of the utilities of the voters. Let us now show that, with synergies, this problem is NP-hard even with very general utility functions. We start by proving a preliminary result for the Clique problem.

Lemma 6.6.1: Clique with $d_{\max} < \sqrt{m}$

The Clique problem is strongly NP-complete even if it is restricted to graphs G in which $d_{\max} < \sqrt{m}$, where m is the number of edges and $d_{\max}$ is the maximum degree of a vertex of G.

Proof. The Clique problem is the following one. We are given an undirected graph $G = (V, E)$, with V the set of n vertices and E the set of m edges. We denote by d_i the degree of a vertex i, and by $d_{\max}$ the maximum degree of any vertex in V. We are also given an integer K. The question is: does there exist a clique of size K in G?

This problem is known to be strongly NP-complete Garey and Johnson [1979], and we now show that it is still strongly NP-complete when the graph G is such that $\sqrt{m} > d_{\max}$.

We reduce the Clique problem in any graph into the Clique problem in a graph where $\sqrt{m} > d_{\max}$. Let G and K be an instance of the Clique problem without any constraint on m and $d_{\max}$. We first transform graph G into a graph G', as follows. Graph G' is built from graph G by "copying" G m times, obtaining m connected components: for any vertex v_i in V, we create $m+1$ vertices $\{v_{i,0}, v_{i,1} \cdots v_{i,m}\}$ in V', and for each edge (v_i, v_j) in E, we create $m+1$ edges $\{(v_{i,0}, v_{j,0}) \cdots (v_{i,m}, v_{j,m})\}$ in E'. We denote by $d^G_{\max}$ (resp. $d^{G'}_{\max}$) the maximum degree of a vertex of G (resp. G'), and by m (resp. m') the number of edges in G (resp. G'). We have $d^{G'}_{\max} = d^G_{\max}$ and $m' = (m+1)m$. Since $m \geq d^G_{\max}$ and $d^{G'}_{\max} = d^G_{\max}$, we have: $m' = m(m+1) \geq d^G_{\max}(d^G_{\max} + 1)$. Therefore, $\sqrt{m'} > d^{G'}_{\max}$. We now show that there is a clique of size K in G' if and only if there is a clique of size K in G.

- Let us first assume that there is a clique $C = \{v_1, v_2 \cdots v_K\}$ of size K in G. In that case, the set $C' = \{v_{1,0}, v_{2,0} \cdots v_{K,0}\}$ is clique of size K in G' since for any edge connecting two vertices v_i and v_j in G we created an edge connecting $v_{i,0}$ and $v_{j,0}$ in E'.
- Let us now assume that there exists a clique of size K in G'. Such a clique can only be formed by a set of vertices $\{v_{1,i}, v_{2,i} \cdots v_{K,i}\}$ with a fixed i since no edges in E' connect two vertices $v_{k,i}$ and $v_{l,j}$ with $i \neq j$ by construction. If such a clique exists, then the subset $C = \{v_1, v_2 \cdots v_K\}$ in G is a clique as well since if an edge $(v_{k,i}, v_{l,i})$ exists in E', an edge (v_k, v_l) exists in E. Therefore C is a clique of size K in G and the answer to the Clique problem is yes.

Since our problem is in NP, and that there exists a polynomial time reduction of the strongly NP-complete Clique problem into the Clique problem when $\sqrt{m} > d_{\max}$, we conclude that the Clique problem is strongly NP-complete even when $\sqrt{m} > d_{\max}$. □

Proposition 6.6.1: PB-Max–$\sum$–u with synergies

Problem PB-Max–$\sum$–u is strongly NP-hard, even if all the projects have unit costs. This is true if $u = u_M$, as well as for any utility function u such that the utility of two projects that have been selected together by at least one voter is strictly larger than the utility of two projects approved by the same number of voters but that have never been selected together by a same voter.

Proof. The decision version of our problem, that we will denote PB-Max–$\sum$–u-dec, is the following one. We are given a number $R \in \mathbb{Z}$ and a budgeting scenario $E = (A, V, c, l)$ with c a cost function such that the cost of each project of A is exactly one. We consider that the utility function u is such that the utility of a pair of projects selected at least once together is strictly larger than the utility of any other pair of projects that have been selected the same number of times but that have never been selected together. The set A is a set of v voters $\{v_1, \ldots, v_v\}$, having each one approved up to l projects of A. The question is: does there exist a set $B \subset A$ of up to l projects such that the utility of B, $\sum_{v_i \in V} u(B_i, E)$, is at least R ?

We reduce the strongly NP-complete problem Clique to this problem. We will assume that the instance of the Clique problem is a graph such that $\sqrt{m} > d_{\max}$ (the Clique problem is still NP-complete in this case, as shown by Lemma 6.6.1). The Clique problem is as follows: given an graph $G = (V, E)$, such that $\sqrt{m} > d_{\max}$, and an integer K, the question is: does there exist a clique of size K?

Given an instance (G, K) of the Clique problem, we create an instance of PB-Max–$\sum$–u-dec as follows. We first transform graph G into a graph G', as follows. We start by setting $G' = G$, and we assume that the $|V|$ vertices of G' are labelled $\{1, \ldots, |V|\}$. For each vertex i of degree $d_i < d_{\max}$, we add $(d_{\max} - d_i)$ new neighbor vertices, denoted by $Dummy(i, 1), \ldots Dummy(i, d_{\max} - d_i)$. By doing this, the vertices of $\{1, \ldots, |V|\}$ are all of degree $d_{\max}$. Let $G' = (V', E')$ be the graph obtained. Each newly added vertex

$Dummy(i,j)$ is of degree 1 in G'. Therefore, the number of newly added vertices in G' is $n_{dummy} = \sum_{i=1}^{|V|} d_{\max} - d_i = |V|d_{\max} - 2|E|$, and the number of newly added edges is the same value. We label the newly added vertices (if any) as $\{|V|+1,\ldots,|V|+n_{dummy}\}$.

We now create from G' a set of projects A as follows. To each vertex $i \in \{1,\ldots,|V'|\}$ we create a corresponding project P_i of cost 1: there are thus $|V|$ projects corresponding each one to one vertex of V, and n_{dummy} projects corresponding each one to one dummy vertex. We create a set $\mathcal{V}$ of $m_V = |E'| + (d_{\max}-1)n_{dummy}$ voters. To each edge $\{x,y\} \in E'$, we create a voter which approves exactly two projects: projects P_x and P_y, corresponding to vertices x and y. For each dummy vertex, we create $(d_{\max}-1)$ voters that approve only the project corresponding to the dummy vertex.

We fix the maximum budget to $l = K$ (since all the projects have a unitary cost, this means that up to K projects can be selected). The value of R, the target utility, depends of the synergy function. We observe that in our instance of PB-Max–$\sum$–u-dec each project is chosen by the same number of voters $(d_{\max}-1)$. Let $u_{together}$ be the utility that a voter obtains for a set of two projects which have both been chosen by the voter. The sequel of the proof works for all utility function such that $u_{together} > 2$. This is in particular true for u_M, as shown by the following fact.

Fact 1: If the utility function is u_M, then $u_{together} > 2$.

Proof of the fact: Let us show that the utility function u_M count positive interactions for pairs of projects corresponding to vertices connected by an edge in G'. For function u_M, we have:

$$m(\{x,y\},\mathcal{V}) \geq r(\{x,y\},\mathcal{V}) - r(\{x\},\mathcal{V})r(\{y\},\mathcal{V})$$

$$m(\{x,y\},\mathcal{V}) \geq \frac{1}{m_V} - \frac{(d_{\max})^2}{(m_V)^2} = \frac{1}{m_V}\left(1 - \frac{(d_{\max})^2}{m_V}\right)$$

Furthermore:

$$m_V = |E| + \sum_{i=1}^{|V|} d_{\max}(d_{\max} - d_i) = |E| + |V|(d_{\max})^2 - \sum_{i=1}^{|V|} d_{\max} d_i$$

Since $d_i d_{\max} \leq (d_{\max})^2$, we get $\sum_{i=1}^{|V|} d_i d_{\max} \leq |V|(d_{max})^2$, and thus $m_V \geq |E|$. Since $\sqrt{|E|} > d_{max}$, $m_V > (d_{\max})^2$ and the Möbius transform of the pair is larger than 0, meaning that the utility of the subset $\{x,y\}$ is larger than the sum of their costs: $u_{together} > 2$.

Let us now show that it possible to select a set of at most K projects of total utility larger than or equal to $R = Kd_{max} + \frac{K(K-1)}{2}(u_{together} - 2)$ if and only if there is a clique of size K in G.

• Let us first assume that there is a clique C of size K in G. Let S^{clique} be the set of the K projects which correspond to the K vertices of C. Note that for each couple of projects x and y of S^{clique}, exactly one voter has approved both x and y. The utility of S^{clique} is thus $u_{together}$ for each of these $K(K-1)/2$ voters. Note also that each project has been selected by exactly d_{max} voters. Therefore, for the $Kd_{max} - 2\times K(K-1)/2$ voters who approve exactly one project of S^{clique}, the utility of S^{clique} is 1. The other voters do not approve any project of S^{clique} and have a utility of 0. The total utility of S^{clique}

is thus $1 \times (Kd_{max} - 2 \times \frac{K(K-1)}{2}) + u_{together}\frac{K(K-1)}{2} = Kd_{max} + \frac{K(K-1)}{2}(u_{together} - 2) = R$. The answer to our problem is thus 'yes'.

• Let us now assume that there is a set C of at most K projects of total utility at least $R = Kd_{max} + \frac{K(K-1)}{2}(u_{together} - 2)$. Note that each project is approved by exactly d_{max} voters. The utility of C for a given voter is 0 if the voter does not select any project of C, 1 if it selects exactly one project, and $u_{together} > 2$ if it approves exactly two projects (recall that a voter approves at most 2 projects). The utility of C is thus equal to n_1, the number of voters who approve exactly one project of C, plus $n_2 \times u_{together}$, where n_2 is the number of voters who approve exactly two projects of C. We have $R = Kd_{max} - 2\frac{K(K-1)}{2} + \frac{K(K-1)}{2}u_{together} \leq n_1 + n_2 u_{together}$, and $n_1 + 2n_2 \leq Kd_{max}$ (since $n_1 + 2n_2$ is equal to the total number of votes for projects of C and C is of size at most K). Therefore, $n_1 = Kd_{max} - 2\frac{K(K-1)}{2}$, and $n_2 = \frac{K(K-1)}{2}$. This means that there are exactly K projects in C and that for each couple of projects of C, there is a voter who approves both projects (recall that there is exactly one voter by edge in G'). Therefore, there exists a clique of size K in G', and thus a clique of size K in G. There exists a polynomial time reduction of the strongly NP-complete Clique problem into the decision version of our problem: PB-Max–$\sum$–u-dec is thus strongly NP-hard. □

The next result extends the result from Sreedurga et al. [2022], which proves that the maxmin participatory budgeting problem is strongly NP-hard for approval voting when the utility function is the sum of the costs of the funded approved projects. We generalize this result by proving that this is true for both the maxmin and the product of utilities and we show that we only need a very weak condition on the utility function for this to be true. Additionally, it holds for knapsack voting, which is more specific that approval voting. We also show that the problem is hard to approximate. We first prove the following lemma.

Lemma 6.6.2: SetCover - Any element is in at most K different sets

The Set cover problem is strongly NP-complete even when restricted to instances in which the number of subsets containing the same element is bounded by K, the size of a feasible solution.

Proof. The Set cover problem is the following one: we are given a set $\mathcal{U}$ of n elements, called the universe, and a collection S of m sets whose union is $\mathcal{U}$. Given an integer $K < m$, the question is: does there exist a set $\mathcal{S}$ of elements in S, such that $\cup_{s\in\mathcal{S}} = \mathcal{U}$ and $|\mathcal{S}| \leq K$?

From an instance $\mathcal{U}$,S,K, we create a new instance $\mathcal{U}'$,S',K'. In this new instance, we create m dummy elements $\{x^1_{dummy} \cdots x^m_{dummy}\}$ and m dummy sets $\{s^1_{dummy} \cdots s^m_{dummy}\}$ such that s^i_{dummy} contains x^i_{dummy}. We then have $n' = n + m$ and $\mathcal{U}' = \mathcal{U} \cup \{x^1_{dummy}, \cdots, x^m_{dummy}\}$, $m' = 2m$ and $S' = S \cup \{s^1_{dummy} \cdots s^m_{dummy}\}$ and $K' = K + m$. In this new instance, it is easy to see that each element is contained by at most $m < K'$ sets.

We now prove that there exists a set cover of $\mathcal{U}'$ with subsets of S' and of size K' at most if and only if there exists a set cover of $\mathcal{U}$ with subsets of S and of size K at most.

- Let us first suppose that there exist a cover C' of $\mathcal{U}'$ with subsets of S' and of size K' at most. This cover necessarily contains the m dummy sets since these sets are the only one containing the m dummy vertex. The $K'-m<K$ other sets of the cover form a feasible cover of the n elements of $\mathcal{U}$ and all of these sets are in S.
- Now, we suppose that that there exist a cover C of $\mathcal{U}$ with subsets of S and of size K at most. The elements of $\mathcal{U}'$ that are not covered by C are the dummy elements. By adding the m dummy sets of S' to C, we obtain a cover C' covering all the elements from $\mathcal{U}$ plus the m dummy elements and of size of $|C|+m$. Since $|C|\leq K$, we have $|C|+m\leq K+m=K'$, we therefore have a feasible cover of $\mathcal{U}'$.

There exist a polynomial time reduction between any instance of Set cover to a version of the Set cover problem in which the number of sets containing the same element is bounded by K. Therefore the Set cover is still strongly NP-complete in that case. □

Proposition 6.6.2: min and $\prod$

Problems PB-Max–min–u and PB-Max–$\prod$–u are strongly NP-hard for any utility function u such that $u(\emptyset,E)=0$ and $u(S,E)>0$ for each $S\neq\emptyset$. For any $\delta>1$, there is no polynomial time δ-approximate algorithm if $P\neq NP$.

Proof. The decision version of our problem is the following one. We are given a real number R and a budgeting scenario $E=(A,V,c,l)$ with A a set of n projects and c a cost function such that the cost of each project is exactly one. We consider a utility function u such that $u(S,E)>0$ if $S\neq\emptyset$. The set V is a set of v voters $\{v_1,\dots,v_v\}$, having each one approved up to l projects of A. The question is: does there exist a set $B\subset A$ of up to l projects such that the utility of B, $\prod_{v_i\in V}u(B_i,E)$ (or $\min_{v\in V}u(B_i,E)$), is at least R ?

We will reduce the strongly NP-complete problem Set cover Garey and Johnson [1979] to this problem. The Set cover problem is the following one: we are given a set $\mathcal{U}$ of n elements, called the universe, and a collection S of m sets whose union equals the universe. Given an integer K, the question is: does there exist a set $\mathcal{S}$ of sets in S, such that $\cup_{s\in\mathcal{S}}=\mathcal{U}$ and $|\mathcal{S}|\leq K$? We suppose that the number of subsets containing the same element is bounded by K – as shown by Lemma 6.6.2, the problem is still strongly NP-complete in that case.

Let $\mathcal{U}$, S and K be an instance of the Set cover problem. Let us create an instance of our problem.

For each element s in $\mathcal{U}$, we create a voter v_e. For every subset s in S, we create a project a_s of cost 1. This project is approved by any voter v_e such that $e\in s$. Note that, since the number of sets containing the same element is smaller than or equal to K, the number of projects approved by a voter is smaller than or equal to K. We set $l=K$ and $R=\epsilon$ with $\epsilon>0$. The question is now: does there exist a bundle B of projects such that

the product (or minimum) of the voters' utilities for bundle B is greater than or equal to ϵ ? Since ϵ can be as small as we want, we can simply look for a solution with value strictly larger than 0.

We show that there is a positive answer to this question if and only if there exists a cover of size K in S.

- Let us first assume that there is a cover C of size K in S. Let B^{cover} be the set of the K projects which correspond to the K sets of S. All voters have at least one of their approved projects in the bundle B^{cover}, since the projects corresponding to the sets have been chosen by the voters matching with the elements. Therefore, if a voter did not have at least one approved project in B^{cover}, then the cover C would not cover the element corresponding to the voter. The answer to our problem is thus 'yes'.
- Let us now assume that it possible to select at most K projects such that the total utility is at least $R=\epsilon$. Since we use the product or the min, this means that every voter has at least one of her approved projects in the funded bundle B. We know that for each $v_e \in V$, there is one project of A_e in B. If we consider the cover C^B formed by the sets corresponding to the projects in B, this means that for every element e, there is a subset s in C^B such that $e \in s$. Since the size of B is at most K, the size of C^B is at most K, which means that C^B is a feasible cover for the Set cover problem. The answer is thus 'yes'.

There exists a polynomial time reduction of the strongly NP-complete Set cover problem into our problem: our problem is strongly NP-hard. Furthermore, a δ-approximate algorithm, with δ, would allow to detect whether there exist a solution with a product (or minimum) of utilities strictly larger than 0, and thus would allow to solve the Set cover problem. Therefore, for any $\delta > 0$, there does not exist polynomial time δ-approximate solution for our problem, unless $P=NP$. □

6.7 A branch and bound algorithm

In this section, we propose an exact branch and bound algorithm for $\alpha - r_u$for $\alpha \in \{\sum, \prod, \min\}$ since, as shown in the previous section, this is NP-hard. We also run experiments on real-life instances.

6.7.1 Description of the algorithm

Let us now present a branch and bound algorithm which solves PB-Max–$\alpha - u$ exactly, for $\alpha \in \{\sum, \min, \prod\}$. Each level of the decision tree corresponds to a project: we either add it to the funded projects – if it fits in the remaining budget, or we ban it for the current node and all of its sons. In such a decision tree, each leaf corresponds to a feasible bundle. Since every decision is binary and there are n consecutive decisions, corresponding to the n projects, there are 2^n leaves corresponding to the 2^n possible

subsets. Since the cost of an optimal bundle is at most l, at a current node, we add a project only if its cost is at most l minus the cost of the currently funded projects – this allows us to prune the tree. Moreover, at each node, we compute a feasible solution, and an upper bound of the value of the quality (w.r.t. the objective function of PB-Max–α–u) of a bundle that is reachable from this node. If the upper bound of the value of a reachable bundle is smaller than the value of a feasible solution we already know, then exploring the node's sons is useless, and we prune the tree.

Case where $\alpha = \sum$. We compute a new feasible solution using a greedy rule, called $\mathcal{R}^{g}_{|B_v|}$ by Talmon and Faliszewski [2019], and which simply selects the projects by decreasing number of selections. At each node we consider the not yet considered projects by decreasing number of selections, and we add a project if it fits in the remaining budget. As we will see in Section 6.7.2, using this algorithm at the root of the tree can also be used as a good and fast heuristic.

The upper bound follows the same principle than the classic upper bound for the Knapsack problem, it is a linear relaxation. In order to compute our upper bound, we need an upper bound on the utility that each project can give to a voter. By multiplying it by the number of voters who selected this project, we obtain an upper bound of the utility that a project can bring to the whole set of voters.

Before starting the exploration of the decision tree, for each project a, we compute the sum of the Möbius transforms of each feasible subset in which a appears, divided by the size of this subset. This is an upper bound of how much utility a project can provide to one voter, we multiply it by the number of voters who selected this project, and obtain an upper bound of how much utility the project can bring to the whole set of voters. Note that this can be applied to other utility functions since the Möbius transforms can be computed for any utility function.

At each node, we then run the greedy algorithm selecting the (non yet selected nor eliminated) projects by decreasing upper bounds and we relax the integrity constraint, obtaining a fractional solution. This gives us an upper bound of the best solution that can be obtained at the current node. Note that the k-additivity assumption is particularly useful here since the maximum utility a project can give decreases when k decreases, since all the Möbius transforms of subsets of size strictly greater than k are null.

Case where $\alpha \in \{\min, \prod\}$. We compute a feasible solution as follows: we look for the set of least satisfied voters. We choose the most frequently selected project by these voters, among projects that fits into the remaining budget. We repeat this process until there is no budget left.

For the upper bound: at the root of the decision tree, we assume that each voter gets her favorite set of projects. At each node, we consider that each voter gets the projects that she voted for among the already selected projects, plus all the projects that she selected among projects that still fits in the budget and which have not been considered yet. For example, if the selected projects cost half the budget, then any project costing more than half the budget could not be chosen and is therefore banned. If a project is banned, then we simply add it to the ban list. Then, we remove all newly

Function	$n=5$	$n=8$	$n=10$	$n=12$	$n=15$
1-additive	0.013	0.057	0.10	0.26	0.77
2-additive	0.015	0.076	0.15	0.60	3.60
3-additive	0.016	0.11	0.25	1.43	9.85

Table 6.1: Completion time (s) of the branch and bound algorithm.

banned project from the best reachable subsets of the voters. This gives us an upper bound of the value of any reachable solution.

Computing the utilities. The utility provided by a given solution B to a voter v_i is the utility of B_i. Determining B_i and computing its utility can be done in polynomial time if we know the utility function. Therefore, for each node of the decision tree, computing solutions and determining their value as upper and lower bounds can be done in polynomial time.

To determine the utility of a bundle with function u_M, we use Equation 6.2. Because of its recursive nature, we compute first, as a preprocessing step, the utility of singletons, then pairs, then triplets and so forth. Determining the utilities in this way cost up to 2^n (since there are 2^n subsets) times nv operations (since determining the appearance rate of a subset costs nv operations). This calculation is much faster with the k-additivity hypothesis, stating, as seen earlier, that we can consider interactions only in subsets of projects of size at most k.

With the k-additivity hypothesis, it is possible to know the utility of a subset of size j in $O(j^k)$ operations, since its utility is the sum of all the Möbius transforms of its parts, and there are at most j^k parts with a non null Möbius transform. This hypothesis has great implications on the computational side.

6.7.2 Experiments

We use real instances from the Pabulib Stolicki et al. [2020] library with a budget limit on the approbation sets of the voters. Experiments are run on an Intel Core i5-8250U processor with 8GB of RAM. We study the completion time of our algorithm and the impact of the synergies on the returned solutions. We consider that $\alpha = \sum$ for the experiments since the sum is the most common aggregator.

Quality of the heuristic. On average, the solution returned by the exact (branch and bound) algorithm has an overall utility 0.28% higher than the utility of the solution returned by the heuristic $\mathcal{R}^g_{|B_v|}$ for the u_M function: the heuristic returns, on the instances of Pabulib, very good solutions with regards to our optimization criterion.

Impact of the k-additivity assumption. The k-additivity assumption allows to decrease the calculation time significantly – the lower k is, the fastest is the algorithm. Table 6.1 indicates the computation times obtained when $k=1$ (no synergy), and when $k=2$ and $k=3$ with utility function u_M.

Impact of considering synergies. We compare the optimal solution for the overlap utility function (1 additive) and the u_M function with no k-additivity assumption. The optimal solutions are different in 35% of the instances, and the amount of money spent differently on average for all the instances is of 28.5%. Therefore, taking synergies into account impacts the returned bundle in a little bit more than a third of the instances, and this impact may be important since the returned bundle considering synergies then differs significantly from an optimal bundle ignoring synergies.

6.8 Conclusion

This chapter represents a first step towards taking project interactions into account in participatory budgeting problems. We introduced a utility function u_M based on the frequency of selection of groups of projects by the voters, and we showed that it fulfills desirable axioms. We furthermore showed that taking into account synergies is NP-hard with the main aggregation criterion, and this for very general utility functions. We designed an exact algorithm that we implemented with u_M but which can also be used with others utility functions.

Whereas, for very costly projects, decision makers will probably identify synergies "by hand", when there are numerous small projects, the authorities will likely be unable or unwilling to identify the synergies. In such settings, identifying the synergies thanks to the preferences of the voters, is promising. We could also imagine settings where a community decides to use a participatory budgeting approach to set a program of a maximum fixed total duration l among various events (presentations, courses, documentaries, etc), each event having a duration (considered as a cost). Members of the community could be asked to select the events they prefer, using knapsack voting: this situation is a participatory budgeting problem for which it would be particularly interesting to take into account synergies between the events.

Chapter 7

Conclusion and perspectives

In this final chapter, we propose a summary of the results presented in the thesis and we present several research directions.

General conclusion. We studied several collective decision problems, from ranking aggregation to multi agent scheduling problems. We used different tools to study collective decision processes: axioms, fairness criteria, computational complexity, objective functions measuring the satisfaction of the agents, fitness to real-world data. An ideal decision process would perform well according to all these evaluation tools, however, as seen in this thesis we most of the time have to chose. This choice can either be between axioms when some axioms are incompatible, or between polynomial complexity and guarantee of an optimal solution when optimization problems are NP-hard, and so on.

Multi agent scheduling problems can have another particularity that does not exist in all collective decision problem. In some cases, satisfying the agents involved in the process can be inefficient for the system as a whole. This is the case when the goal of the system is to minimize some objective, e.g. the makespan or the sum of the completion times of the tasks, and agents have preferences that deviate from this objective. In such cases, decision processes should also take this in consideration in order to return a solution that satisfies the agents and is as efficient as possible for the system. This trade off between efficiency and fairness can take different forms, as seen in this thesis.

We now give a little more details regarding each chapter.

Multi-Organization Scheduling Problem. In Chapter 2, we focused on the Multi-Organization Scheduling Problem (MOSP). We have studied the necessary trade off between efficiency (in term of makespan minimization) and fairness, either expressed with the rationality constraints or as an optimization criterion. We have also shown the interest of cooperation, that can benefit to all the organizations. We have seen that finding a solution in which each organization is as satisfied as possible is a NP-hard problem, that is even NP-hard to approximate. We have also seen that such a solution can be inefficient regarding the overall makespan minimization.

There are numerous work directions:

1. Online tasks: in such a setting, the algorithm scheduling the tasks does not have knowledge about the tasks from the beginning: the information arrives along the execution. There are several interesting questions: How do we define the rationality constraint in this context ? How do we define fairness ? How do we maintain fairness over time ? These questions arise both if the algorithm is completely unaware of the upcoming tasks or if we have some predictions on upcoming tasks.

2. Additional scheduling constraints: several of our results can be extended to the case where the tasks have release dates since "negative" results hold in a more general case and some of our "positive" results can be directly adapted. Some constraints, like deadline, are pretty easy to handle for organizations when they schedule their own tasks on their own machines but it would be interesting to see how to handle such constraints when machines are shared.

3. Different scheduling objectives: we considered that all organizations wanted to minimize their makespan, however it may also be possible that some organizations have different objectives, like minimizing the sum of the completion times of their tasks or some due date criterion like deviation or tardiness. In such cases, the rationality constraint can be directly extended (by replacing the makespan objective by the organization's objective) but the way the system handles such a situation remains to be defined. This question is particularly interesting when the organizations have very different objectives, and when their objectives are very different from the system one.

4. Different machines: Different organizations having different clusters and possibly different machines is also an interesting context to consider. In that case organization having powerful machines may be more demanding since they bring more processing power.

5. Fairness: we focused on two fairness criteria: the rationality constraint and the Maximum Minimum Gain. There are a lot of other fairness criteria: one of the most commonly used for sharing resources is the envy-freeness criterion. An allocation is said to be envy-free if no agent prefers the share of another agent to her share. Studying whether this criterion can be extended to MOSP is a very interesting research direction.

6. Distributed algorithms: the algorithms we studied suppose that there is an entity gathering all the information from all organizations, computing a schedule and returning it to the organizations. Even if this algorithm has guarantees, like fulfilling the rationality constraint, it is not obvious that organizations would be willing to give all their information to a central entity. It would be interesting to design algorithms where organizations build progressively a common schedule by sharing information with each other and progressively building the final

schedule. Such an algorithm is called distributed since it relies on agents executing the algorithm instead of a central entity. For example, studying how the heuristic MCEDD behaves in such a context would be interesting.

Collective Schedules. In Chapters 3 and 4, we studied the Collective Schedules problem. We took an axiomatic approach in Chapter 3 and studied several aggregation rules. Three of these rules minimize the sum of the dissatisfaction of the agents, according to some notions of dissatisfaction: two which generalize scheduling criteria and one extending the Kendall-Tau distance. In addition to this minimization aspect, these rules satisfy certain axiomatic properties, some of them being meaningful in EB (Earlier is Better) settings, i.e. settings in which scheduling tasks as early as possible is a good thing. We showed incompatibility results between some axioms. These three rules solve NP-hard problems. We also studied a fourth rule, called EMD, for (Earliest Median Date) which schedules tasks by increasing median completion time in the preferences of the voters. This rule does not have the same theoretical guarantees than the first three but can be computed in polynomial time and is also easier to understand for the voters.

There are several research perspectives and open questions:

1. Length Reduction Monotonicity (LRM axiom): firstly, we conjectured that the Length Reduction Monotonicity axiom (see Definition 3.3.4) is fulfilled by two of the rules we studied (the PTA Kemeny and ΣT rules). Showing that this is indeed the case (or not) is an open problem.

2. Domain restriction: in voting theory, it is sometimes useful to consider that the preferences of the voters follow some underlying structure. One of the most studied case is the Single-Peaked preferences case [Black, 1948]. In such a setting, candidates can be placed on an axis and the preferences of the voters have to follow this axis. For example, if we have 3 candidates placed on the axis in the order $1, 2, 3$, then a voter cannot express the preference $1 \prec 3 \prec 2$ since candidate 2 is between 1 and 3 in the axis but not in the preference. This same idea of structured preferences can be extended to handle cases in which candidates and voters can be positioned on a map, voters being more favorable to candidates that are close to them. The reason it is interesting for collective schedules is that when we model public works in that way, there may be some underlying structure on the preferences. If we assume that a municipality wants to build infrastructures, a voter may prefer the municipality to build the closest one to his or her place first. When we have this knowledge about the instance, some problems may be easier to solve and some rules may fulfill certain axioms that they do not necessarily fulfill in the general case.

3. Axiomatic characterization: It is sometimes possible to characterize a voting rule with a set of axioms. This means that the set of axioms defines perfectly the aggregation rule r in the sense that any rule that fulfills this set of axioms always return

the same solution that rule r. Such a result exists for the Kemeny rule, which is known to be the only aggregation rule which fulfills reinforcement, Condorcet consistency and neutrality [Fishburn, 1981]. Studying whether such results can be found in the collective schedules problem is an interesting perspective.

4. Scheduling extensions: There are several potential extensions of this problem. We could study the case in which there are several machines that can process the tasks. In such a case we have to know how voters express their preferences, do they still provide a complete schedule ? When there are several machines, all the schedules do not have the same makespan, is it important and if so how do we balance makespan minimization and voter satisfaction ? We could also imagine that the processing time of the tasks may be uncertain, e.g. a project may be announced as lasting 4 months but because of some factors end up being longer than that. In this case, can we find algorithms that are robust ?

In Chapter 4, we focused on the particular case in which all tasks have the same length. We considered a more general way of expressing preferences since voters can express time intervals for each task instead of giving a complete schedule. We studied two aggregation rules, one based on a distance criterion and one on a binary criterion. These two criteria extend the criteria introduced in the previous chapter. This allows the voter to give very precise preferences. We studied classic scheduling constraints, namely the release dates, deadlines and precedence constraints. We investigated two settings, one in which voters are aware of such constraints and express preferences that fulfill them, and a second one in which the preferences are not constrained but the solution we return is. We showed that adding release dates and deadlines do not increase the complexity of the problem, regardless of whether the preferences fulfill them or not. Precedence constraints on the other hand add complexity if the preferences do not fulfill them. We also showed that the EMD algorithm introduced earlier is 2-approximate for the minimization of the distance criterion when tasks are of the same length.

Among the research directions, we mention:

1. Complexity: whether the minimization of the binary criterion is an NP-hard problem when preferences fulfill precedence constraints remains an open question.

2. Approximation: several of the problems studied are NP-hard. Looking for approximation algorithms, maybe by extending algorithms from the scheduling literature, is a promising research direction.

3. New aggregation rules: we showed that the rules we studied did not fulfill two axioms, namely the deadline consistency and the release date consistency. Designing an aggregation rule that fulfills such axioms and has other theoretical guarantees is an interesting perspective.

Ranking aggregation In Chapter 5 we studied the ranking aggregation problem with a probabilistic approach. We have studied here an extension of Young's model for pairwise preferences to choices in subsets of size at most k, showing that the maximum likelihood ranking w.r.t. this model coincides with a consensus ranking for the k-wise Kemeny rule under certain assumptions on the choice probabilities. Relaxing these assumptions, we have proposed inference algorithms for the model, learning the choice probabilities from the data. The fitness of the model on real data is comparable with the Plackett-Luce model, although no utilities are embedded in our model. Regarding our model, there are a few work directions:

1. Extension of Mallows' model: an interesting connection has been shown by Mallows [1957] between Young's model for binary relations and Mallows' model for rankings: sampling a ranking using Mallows' model is equivalent to sampling a binary relations R using Young's model with probability $p = e^{\theta}/(1 + e^{\theta})$ until a transitive binary relation R is obtained. An interesting extension of the work presented in this thesis would therefore consist in determining if the k-wise Young's model can be related to a k-wise distance-based statistical model M for *rankings*, in the same manner as Young's model and Mallows' model, and to investigate effective sampling and learning methods for M from choice data.

2. Rank dependent probabilities: an extension of our model could consist in considering that the probability of choosing a candidate of the subset that is not the winning candidate according to the ground truth ranking depends on its position in the subset. For example, in a triplet choosing the second candidate would be more likely than choosing the third candidate. This yields a new model and a new aggregation rule in which disagreements are weighted. Given a subset ordered as $1 \prec 2 \prec 3$ in the ranking we want to evaluate, we would count a disagreement if candidate 2 or candidate 3 is ranked above the others in a preference but if candidate 3 is ranked first in this subset, the disagreement would have a greater weight that if 2 was ranked first in this subset. Studying the fitness of such a model and the properties of this aggregation rule are interesting research directions.

3. Probabilistic models for collective decision: on a more general note, this probabilistic approach could be used for other voting relating problems, like the collective schedules problem or the participatory budgeting problem. A lot of real world data is available for the participatory budgeting problem [Stolicki et al., 2020], making it a prime candidate to be studied with a probabilistic approach.

Participatory Budgeting. We introduced a model for the Participatory Budgeting problem when there are interactions between projects. We introduced a utility function u_M based on the frequency of selection of groups of projects by the voters, and we showed that it fulfills desirable axioms. We furthermore showed that taking into account synergies is NP-hard with the main aggregation criteria, and this for very general utility functions. We designed an exact algorithm that we implemented with u_M but which

can also be used with other utility functions. This automatic detection of interactions is relevant in contexts in which there are many projects or in contexts in which the authority running the participatory budgeting is not willing or able to measure the interactions on its own. We could also imagine settings where a community decides to use a participatory budgeting approach to set a program of a maximum fixed total duration l among various events (presentations, courses, documentaries, etc), each event having a duration (considered as a cost). Members of the community could be asked to select the events they prefer, using knapsack voting: this situation is a participatory budgeting problem for which it would be interesting to take into account synergies between the events.

We mention a few research perspectives:

1. Proportionality: our model tends to consider that the utility of groups of projects chosen together is larger than the utility of projects that are not chosen together. This means that group of projects that often appear together in the preferences are more likely to be chosen together. This idea goes against one resolution principle: the proportionality [Aziz et al., 2017]. The main idea behind proportionality is the following one: a part of x% of the population should be allocated a part of x% of the budget. For example, this means that if half the population approve a project that costs half of the budget, then all these voters should be satisfied at least as much as if this project was funded (since there is a way of satisfying all of them by selecting the project). Although the utility function we describe does not seem to favor proportionality, it does not mean that it is impossible to combine this principle with the use of interactions between projects and this is an interesting research perspective.

2. Simpler models for interactions: our model, as well as the other models taking synergies into account, seem to be hard to grasp for voters. Designing a simpler model which allows the decision maker to gain information regarding the interactions between the projects thanks to the preference of the voters is an interesting perspective. As seen in Proposition 6.6.1, maximizing voters' utility when the utility function takes interactions into account is an NP-hard problem. However, even within the model we described, it may be possible to design aggregation rules fulfilling the axioms presented and that are easier to understand for voters.

3. Probabilistic approach: as said earlier, the amount of available data for participatory budgeting makes a probabilistic approach possible. It would be particularly interesting to compare a probabilistic model considering interactions between projects with another one which considers that the projects are independent. Comparing these two model in terms of fitness could show that there are synergies between projects and that the votes expressed by the citizens take this into account.

Papers

Martin Durand and Fanny Pascual. Efficiency and equity in the multi organization scheduling problem. *Theoretical Computer Science*, 864:103–117, 2021.

Martin Durand and Fanny Pascual. Collective schedules: Axioms and algorithms. In *Algorithmic Game Theory: 15th International Symposium, SAGT 2022, Colchester, UK, September 12–15, 2022, Proceedings*, pages 454–471. Springer, 2022.

Martin Durand, Fanny Pascual, and Olivier Spanjaard. A non-utilitarian discrete choice model for preference aggregation. In *International Conference on Scalable Uncertainty Management*, pages 157–171. Springer, 2022.

Bibliography

Alessandro Agnetis, Jean-Charles Billaut, Stanislaw Gawiejnowicz, Dario Pacciarelli, and Ameur Soukhal. *Multiagent Scheduling. Models and Algorithms*. Springer, 2014.

Alessandro Agnetis, Bo Chen, Gaia Nicosia, and Andrea Pacifici. Price of fairness in two-agent single-machine scheduling problems. *European Journal of Operational Research*, 276(1):79–87, 2019. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2018.12.048. URL https://www.sciencedirect.com/science/article/pii/S0377221719300025.

Fuad Aleskerov. *Arrovian aggregation models*. Kluwer Academic, 1999.

Kenneth J Arrow. A difficulty in the concept of social welfare. *Journal of political economy*, 58(4):328–346, 1950.

Kenneth J Arrow. Social choice and individual values, 1951.

Abolfazl Asudeh, H. V. Jagadish, Julia Stoyanovich, and Gautam Das. Designing fair ranking schemes. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1259–1276. Association for Computing Machinery, 2019. ISBN 9781450356435. doi: 10.1145/3299869.3300079. URL https://doi.org/10.1145/3299869.3300079.

Haris Aziz and Nisarg Shah. Participatory budgeting: Models and approaches. In *Pathways Between Social Science and Computational Social Science*, pages 215–236. Springer, 2021.

Haris Aziz, Barton Lee, and Nimrod Talmon. Proportionally representative participatory budgeting: Axioms and algorithms. *arXiv preprint arXiv:1711.08226*, 2017.

Haris Aziz, Xin Huang, Nicholas Mattei, and Erel Segal-Halevi. Computing welfare-maximizing fair allocations of indivisible goods. *European Journal of Operational Research*, 307(2):773–784, 2023. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2022.10.013. URL https://www.sciencedirect.com/science/article/pii/S0377221722007822.

Katherine A Baldiga and Jerry R Green. Assent-maximizing social choice. *Social Choice and Welfare*, 40(2):439–460, 2013.

John Bartholdi, Craig A Tovey, and Michael A Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and welfare*, 6(2):157–165, 1989.

Richard Bellman. Mathematical aspects of scheduling theory. *Journal of the Society for Industrial and Applied Mathematics*, 4(3):168–205, 1956.

Gerdus Benade, Swaprava Nath, Ariel D Procaccia, and Nisarg Shah. Preference elicitation for participatory budgeting. *Management Science*, 67(5):2813–2827, 2021.

Nadja Betzler, Michael R Fellows, Jiong Guo, Rolf Niedermeier, and Frances A Rosamond. Fixed-parameter algorithms for kemeny rankings. *Theoretical Computer Science*, 410(45):4554–4570, 2009.

Asia J. Biega, Krishna P. Gummadi, and Gerhard Weikum. Equity of attention: Amortizing individual fairness in rankings. In *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '18, page 405–414. Association for Computing Machinery, 2018. ISBN 9781450356572. doi: 10.1145/3209978.3210063. URL https://doi.org/10.1145/3209978.3210063.

Duncan Black. On the rationale of group decision-making. *Journal of political economy*, 56(1):23–34, 1948.

Jacek Błażewicz, Klaus H Ecker, Erwin Pesch, Günter Schmidt, and Jan Weglarz. *Scheduling computer and manufacturing processes*. springer science & Business media, 2001.

Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973. ISSN 0022-0000. doi: https://doi.org/10.1016/S0022-0000(73)80033-9. URL https://www.sciencedirect.com/science/article/pii/S0022000073800339.

Linus Boes, Rachael Colley, Umberto Grandi, Jérôme Lang, and Arianna Novaro. Collective discrete optimisation as judgment aggregation. *CoRR*, abs/2112.00574, 2021. URL https://arxiv.org/abs/2112.00574.

Sylvain Bouveret, Yann Chevaleyre, Nicolas Maudet, and Hervé Moulin. Fair allocation of indivisible goods., 2016.

Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia. *Handbook of computational social choice*. Cambridge University Press, 2016.

Peter Brucker. *Scheduling Algorithms*. Springer, 5th edition, 2010. ISBN 3642089070.

Ioannis Caragiannis, Ariel D Procaccia, and Nisarg Shah. When do noisy votes reveal the truth? *ACM Transactions on Economics and Computation (TEAC)*, 4(3):1–30, 2016.

L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi. Ranking with fairness constraints. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 107 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi: 10.4230/LIPIcs.ICALP.2018.28. URL https://doi.org/10.4230/LIPIcs.ICALP.2018.28.

Anirudh Chakravorty, Neelima Gupta, Neha Lawaria, Pankaj Kumar, and Yogish Sabharwal. Algorithms for the relaxed multiple-organization multiple-machine scheduling problem. In *20th Annual International Conference on High Performance Computing, HiPC 2013, Bengaluru (Bangalore), Karnataka, India, December 18-21, 2013*, pages 30–38. IEEE Computer Society, 2013. doi: 10.1109/HiPC.2013.6799127. URL https://doi.org/10.1109/HiPC.2013.6799127.

Johanne Cohen and Fanny Pascual. Scheduling tasks from selfish multi-tasks agents. In *Euro-Par 2015: Parallel Processing: 21st International Conference on Parallel and Distributed Computing, Vienna, Austria, August 24-28, 2015, Proceedings 21*, pages 183–195. Springer, 2015.

Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. Analysis of multi-organization scheduling algorithms. In Pasqua D'Ambra, Mario Rosario Guarracino, and Domenico Talia, editors, *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part II*, volume 6272 of *Lecture Notes in Computer Science*, pages 367–379. Springer, 2010. doi: 10.1007/978-3-642-15291-7_34. URL https://doi.org/10.1007/978-3-642-15291-7_34.

Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. Coordination mechanisms for selfish multi-organization scheduling. In *18th International Conference on High Performance Computing, HiPC 2011, Bengaluru, India, December 18-21, 2011*, pages 1–9. IEEE Computer Society, 2011a. doi: 10.1109/HiPC.2011.6152720. URL https://doi.org/10.1109/HiPC.2011.6152720.

Johanne Cohen, Daniel Cordeiro, Denis Trystram, and Frédéric Wagner. Multi-organization scheduling approximation algorithms. *Concurr. Comput. Pract. Exp.*, 23 (17):2220–2234, 2011b. doi: 10.1002/cpe.1752. URL https://doi.org/10.1002/cpe.1752.

Johanne Cohen, Daniel Cordeiro, and Pedro Luis F. Raphael. Energy-aware multi-organization scheduling problem. In Fernando M. A. Silva, Inês de Castro Dutra, and Vítor Santos Costa, editors, *Euro-Par 2014 Parallel Processing - 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings*, volume 8632 of *Lecture Notes in Computer Science*, pages 186–197. Springer, 2014. doi: 10.1007/978-3-319-09873-9_16. URL https://doi.org/10.1007/978-3-319-09873-9_16.

Vincent Conitzer and Tuomas Sandholm. Common voting rules as maximum likelihood estimators. In *Proceedings of UAI 2005*, pages 145–152, 2005.

Vincent Conitzer, Andrew Davenport, and Jayant Kalagnanam. Improved bounds for computing kemeny rankings. In *AAAI*, volume 6, pages 620–626, 2006.

Vincent Conitzer, Matthew Rognlie, and Lirong Xia. Preference functions that score rankings and maximum likelihood estimation. In *IJCAI*, pages 109–115, 2009.

Daniel Cordeiro, Pierre-François Dutot, Grégory Mounié, and Denis Trystram. Tight analysis of relaxed multi-organization scheduling algorithms. In *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May, 2011 - Conference Proceedings*, pages 1177–1186. IEEE, 2011. doi: 10.1109/IPDPS.2011.112. URL https://doi.org/10.1109/IPDPS.2011.112.

Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *SIGIR*, pages 758–759, 2009.

Nicolas De Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*. Cambridge University Press, 2014.

P. Diaconis and R.L. Graham. *Spearman's Footrule as a Measure of Disarray*. Stanford University. Department of Statistics, 1976.

Persi Diaconis and Ronald L Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(2):262–268, 1977.

Jean-Paul Doignon, Aleksandar Pekeč, and Michel Regenwetter. The repeated insertion model for rankings: Missing link between two subset choice models. *Psychometrika*, 69(1):33–54, 2004.

Mohamed Drissi-Bakhkhat and Michel Truchon. Maximum likelihood approach to vote aggregation with variable probabilities. *Social Choice and Welfare*, 23(2):161–185, 2004.

Pierre-François Dutot, Fanny Pascual, Krzysztof Rzadca, and Denis Trystram. Approximation algorithms for the multiorganization scheduling problem. *IEEE Trans. Parallel Distrib. Syst.*, 22(11):1888–1895, 2011.

Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In Vincent Y. Shen, Nobuo Saito, Michael R. Lyu, and Mary Ellen Zurko, editors, *Proceedings of the Tenth International World Wide Web Conference, WWW*, pages 613–622. ACM, 2001. doi: 10.1145/371920.372165. URL https://doi.org/10.1145/371920.372165.

Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972. doi: 10.1145/321694.321699. URL https://doi.org/10.1145/321694.321699.

Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer, 2005.

Edith Elkind and Arkadii Slinko. Rationalizations of voting rules. *Handbook of Computational Social Choice*, 2016.

Edith Elkind, Piotr Faliszewski, and Arkadii M Slinko. On the role of distances in defining voting rules. In *AAMAS*, volume 10, pages 375–382, 2010.

Edith Elkind, Piotr Faliszewski, and Arkadii M Slinko. Homogeneity and monotonicity of distance-rationalizable voting rules. In *AAMAS*, volume 2011, pages 821–828, 2011.

Edith Elkind, Sonja Kraiczy, and Nicholas Teh. Fairness in temporal slot assignment. In Panagiotis Kanellopoulos, Maria Kyropoulou, and Alexandros A. Voudouris, editors, *Algorithmic Game Theory - 15th International Symposium, SAGT 2022, Colchester, UK, September 12-15, 2022, Proceedings*, volume 13584 of *Lecture Notes in Computer Science*, pages 490–507. Springer, 2022. doi: 10.1007/978-3-031-15714-1_28. URL https://doi.org/10.1007/978-3-031-15714-1_28.

Roy Fairstein, Reshef Meir, and Kobi Gal. Proportional participatory budgeting with substitute projects. *CoRR*, abs/2106.05360, 2021. URL https://arxiv.org/abs/2106.05360.

Benadè Gerdus Fairstein Roy and Gal Kobi. Participatory budgeting design for the real world. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Multiwinner voting: A new challenge for social choice theory. *Trends in computational social choice*, 74(2017):27–47, 2017.

Peter C Fishburn. An analysis of simple voting systems for electing committees. *SIAM Journal on Applied Mathematics*, 41(3):499–502, 1981.

Rupert Freeman, David M. Pennock, Dominik Peters, and Jennifer Wortman Vaughan. Truthful aggregation of budget proposals. *Journal of Economic Theory*, 193:105234, 2021. ISSN 0022-0531. doi: https://doi.org/10.1016/j.jet.2021.105234. URL https://www.sciencedirect.com/science/article/pii/S002205312100051X.

M. R. Garey and D. S. Johnson. Scheduling tasks with nonuniform deadlines on two processors. *J. ACM*, 23(3):461–467, jul 1976. ISSN 0004-5411. doi: 10.1145/321958.321967. URL https://doi.org/10.1145/321958.321967.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, 1979.

Sahin Cem Geyik, Stuart Ambler, and Krishnaram Kenthapadi. Fairness-aware ranking in search and recommendation systems with application to linkedin talent search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '19, page 2221–2231. Association for Computing Machinery,

2019. ISBN 9781450362016. doi: 10.1145/3292500.3330691. URL https://doi.org/10.1145/3292500.3330691.

Hugo Gilbert, Tom Portoleau, and Olivier Spanjaard. Beyond pairwise comparisons in social choice: A setwise Kemeny aggregation problem. In *AAAI*, pages 1982–1989, 2020.

Ashish Goel, Anilesh K Krishnaswamy, Sukolsak Sakshuwong, and Tanja Aitamurto. Knapsack voting for participatory budgeting. *ACM Transactions on Economics and Computation (TEAC)*, 7(2):1–27, 2019.

Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

Leslie A. Hall and David B. Shmoys. Approximation schemes for constrained scheduling problems. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 134–139. IEEE Computer Society, 1989. doi: 10.1109/SFCS.1989.63468. URL https://doi.org/10.1109/SFCS.1989.63468.

Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, January 1987. ISSN 0004-5411. doi: 10.1145/7531.7535. URL https://doi.org/10.1145/7531.7535.

Alexandru Iosup, Catalin Dumitrescu, Dick Epema, Hui Li, and Lex Wolters. How are Real Grids Used? The Analysis of Four Grid Traces and Its Implications. In *2006 7th IEEE/ACM International Conference on Grid Computing*, pages 262–269, September 2006. doi: 10.1109/ICGRID.2006.311024. ISSN: 2152-1093.

J.R. Jackson. *Scheduling a production line to minimize maximum tardiness*. Research report. Office of Technical Services, 1955.

Pallavi Jain, Krzysztof Sornat, and Nimrod Talmon. Participatory budgeting with project interactions. In *IJCAI*, pages 386–392, 2020.

Pallavi Jain, Nimrod Talmon, and Laurent Bulteau. Partition aggregation for participatory budgeting. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 665–673, 2021.

Toshihiro Kamishima. Nantonac collaborative filtering: recommendation based on order responses. In *SIGKDD*, pages 583–588, 2003.

John G. Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity results for scheduling chains on a single machine. *European Journal of Operational Research*, 4(4):270–275, 1980. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(80)90111-3. URL https://www.sciencedirect.com/science/article/pii/0377221780901113. Combinational Optimization.

Joseph Y-T Leung and Gilbert H Young. Minimizing total tardiness on a single machine with precedence constraints. *ORSA Journal on Computing*, 2(4):346–352, 1990.

Tyler Lu and Craig Boutilier. The unavailable candidate model: a decision-theoretic view of social choice. In *Proceedings of EC 2010*, pages 263–274, 2010.

Tyler Lu and Craig Boutilier. Effective sampling and learning for mallows models with pairwise-preference data. *J. Mach. Learn. Res.*, 15(1):3783–3829, 2014.

Uri Lublin and Dror G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11): 1105–1122, November 2003. ISSN 07437315. doi: 10.1016/S0743-7315(03)00108-4. URL https://linkinghub.elsevier.com/retrieve/pii/S0743731503001084.

R. Duncan Luce. *Individual Choice Behavior: A Theoretical Analysis*. Courier Corporation, 2012. ISBN 978-0-486-15339-1.

Colin L Mallows. Non-null ranking models. i. *Biometrika*, 44(1/2):114–130, 1957.

Nicholas Mattei and Toby Walsh. Preflib: A library of preference data http://preflib.org. In *ADT 2013*, Lecture Notes in Artificial Intelligence. Springer, 2013.

Harikrishna Narasimhan, Andy Cotter, Maya Gupta, and Serena Lutong Wang. Pairwise fairness for ranking and regression. In *33rd AAAI Conference on Artificial Intelligence*, 2020.

April Niu, Agnes Totschnig, and Adrian Vetta. Fair algorithm design: Fair and efficacious machine scheduling. *arXiv preprint arXiv:2204.06438*, 2022.

Fukuhito Ooshita, Tomoko Izumi, and Taisuke Izumi. A generalized multi-organization scheduling on unrelated parallel machines. In *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2009, Higashi Hiroshima, Japan, 8-11 December 2009*, pages 26–33. IEEE Computer Society, 2009. doi: 10.1109/PDCAT.2009.26. URL https://doi.org/10.1109/PDCAT.2009.26.

Fukuhito Ooshita, Tomoko Izumi, and Taisuke Izumi. The price of multi-organization constraint in unrelated parallel machine scheduling. *Parallel Process. Lett.*, 22 (2), 2012. doi: 10.1142/S0129626412500065. URL https://doi.org/10.1142/S0129626412500065.

Fanny Pascual, Krzysztof Rzadca, and Denis Trystram. Cooperation in multi-organization scheduling. In *European Conference on Parallel Processing*, pages 224–233. Springer, 2007.

Fanny Pascual, Krzysztof Rzadca, and Denis Trystram. Cooperation in multi-organization scheduling. *Concurr. Comput. Pract. Exp.*, 21(7):905–921, 2009. doi: 10.1002/cpe.1378. URL https://doi.org/10.1002/cpe.1378.

Fanny Pascual, Krzysztof Rzadca, and Piotr Skowron. Collective schedules: Scheduling meets computational social choice. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 667–675, 2018.

David M Pennock, Eric Horvitz, C Lee Giles, et al. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *AAAI/IAAI*, pages 729–734, 2000.

Paz Perez-Gonzalez and Jose M Framinan. A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs: Multi-agent scheduling problems. *European Journal of Operational Research*, 235(1):1–16, 2014.

Dominik Peters, Grzegorz Pierczyński, and Piotr Skowron. Proportional participatory budgeting with additive utilities. *Advances in Neural Information Processing Systems*, 34:12726–12737, 2021.

Michael L Pinedo. *Scheduling*, volume 29. Springer, 2012.

R. L. Plackett. The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(2):193–202, 1975. ISSN 0035-9254. doi: 10.2307/2346567. URL https://www.jstor.org/stable/2346567.

Anthony Przybylski, Kathrin Klamroth, and Renaud Lacour. A simple and efficient dichotomic search algorithm for multi-objective mixed integer linear programs. *arXiv*, 2019.

Karthik Raman and Thorsten Joachims. Methods for ordinal peer grading. In *SIGKDD*, pages 1037–1046, 2014.

Simon Rey and Jan Maly. The (computational) social choice take on indivisible participatory budgeting, 2023.

Ariel Rosenfeld and Nimrod Talmon. What should we optimize in participatory budgeting? an experimental study. *CoRR*, abs/2111.07308, 2021. URL https://arxiv.org/abs/2111.07308.

Gian-Carlo Rota. On the foundations of combinatorial theory i. theory of möbius functions. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 2(4):340–368, 1964.

Erik Saule and Denis Trystram. Multi-users scheduling in parallel systems. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS*, pages 1–9. IEEE, 2009. doi: 10.1109/IPDPS.2009.5161037. URL https://doi.org/10.1109/IPDPS.2009.5161037.

Gideon Schwarz. Estimating the dimension of a model. *Ann. of stat.*, pages 461–464, 1978.

Amartya Sen. The possibility of social choice. *American eco. rev.*, 89(3):349–378, 1999.

Ashudeep Singh and Thorsten Joachims. Fairness of exposure in rankings. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, page 2219–2228. Association for Computing Machinery, 2018. ISBN 9781450355520. doi: 10.1145/3219819.3220088. URL https://doi.org/10.1145/3219819.3220088.

Piotr Skowron and Krzysztof Rzadca. Non-monetary fair scheduling: a cooperative game theory approach. In *SPAA*, pages 288–297, 2013.

Piotr Skowron, Martin Lackner, Markus Brill, Dominik Peters, and Edith Elkind. Proportional rankings. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, pages 409–415. ijcai.org, 2017. doi: 10.24963/ijcai.2017/58. URL https://doi.org/10.24963/ijcai.2017/58.

Hossein Azari Soufiani, David C. Parkes, and Lirong Xia. Random utility theory for social choice. In *NeurIPS*, NIPS'12, page 126–134. Curran Associates Inc., 2012.

Gogulapati Sreedurga, Mayank Ratan Bhardwaj, and Yadati Narahari. Maxmin participatory budgeting. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 489–495. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/70. URL https://doi.org/10.24963/ijcai.2022/70. Main Track.

Dariusz Stolicki, Stanislaw Szufa, and Nimrod Talmon. Pabulib: A participatory budgeting library. *CoRR*, abs/2012.06539, 2020. URL https://arxiv.org/abs/2012.06539.

Nimrod Talmon and Piotr Faliszewski. A framework for approval-based budgeting methods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2181–2188, 2019.

N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971. doi: 10.1002/net.3230010206. URL https://doi.org/10.1002/net.3230010206.

Long Wan and Jinjiang Yuan. Single-machine scheduling to minimize the total earliness and tardiness is strongly np-hard. *Operations Research Letters*, 41(4):363–365, 2013.

Lirong Xia. Learning and decision-making from rank data. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13(1):1–159, 2019.

Ronald R Yager and Janusz Kacprzyk. *The ordered weighted averaging operators: theory and applications*. Springer Science & Business Media, 2012.

H Peyton Young. Condorcet's theory of voting. *American Political science review*, 82(4): 1231–1244, 1988.

H Peyton Young and Arthur Levenglick. A consistent extension of condorcet's election principle. *SIAM Journal on applied Mathematics*, 35(2):285–300, 1978.

Summary

This thesis focuses on several collective decision making problems, from multi agent scheduling to participatory budgeting. For each of these problems, the goal is to take a decision that impacts several agents. These agents can represent citizens, companies, members of a research laboratory, ... Such a solution can be a schedule of tasks of interest for the agents, a ranking of items that the agents have to sort or a selection of common projects to fund. Each agent has his or her own interest over the possible solutions and our goal is to find a solution that satisfies the agents as much as possible.

Any solution can be evaluated thanks to different tools. We will mostly focus on fairness and efficiency: a solution has to be efficient for the whole set of agents and fair in the sense that no single agent should be too unsatisfied. Fairness and efficiency can be formulated in different ways, from objective functions to axiomatic properties.

We study several problems in this thesis and put an emphasis on scheduling problems.