Algorithms Aspects of " Multistage " Optimization

Alexandre Teiller

SORBONNE UNIVERSITÉ
LIP6
(Équipe Recherche Opérationnelle)

THÈSE DE DOCTORAT EN INFORMATIQUE

Aspects Algorithmiques de l'Optimisation "Multistage"

Présentée et soutenue publiquement le 01/12/2020 par

ALEXANDRE TEILLER

devant le jury composé de:

Directeurs de thèse:	Evripidis Bampis Professeur à Sorbonne Université, Paris
	Bruno Escoffier Professeur à Sorbonne Université, Paris
Rapporteurs:	Cedric Bentz Maître de Conférences au CNAM, Paris
	Denis Trystram Professeur à Grenoble INP, Grenoble
Examinateurs:	Carlos Agon Professeur à Sorbonne Université, Paris
	Christian Laforest Professeur à ISIMA, Aubière
	Valia Mitsou Maîtresse de Conférences à l'Université de Paris, Paris

Contents

Introduction

In a classical combinatorial optimization setting, given an instance of a problem one needs to find a good feasible solution. However, in many situations, the data may evolve over time and one has to solve a sequence of instances. The natural approach of solving every instance independently may induce a significant transition cost, for instance for moving a system from one state to another. Gupta et al. (2014) and Eisenstat et al. (2014) proposed a *multistage* model where given a time horizon $t = 1, 2, \ldots, T$, the input is a sequence of instances $I_1, I_2, \ldots, I_T$, (one for each time step), and the goal is to find a sequence of solutions $S_1, S_2, \ldots, S_T$ (one for each time step) reaching a trade-off between the quality of the solutions in each time step and the stability/similarity of the solutions in consecutive time steps. The *multistage* framework is the main subject of the thesis in which we addressed some maximization problems in the offline setting as well as in the online one and study a direct application of the framework in a musical context. Roughly speaking, in the offline setting, one has a complete knowledge of the data over the time horizon whereas in the online setting, one only knows the data at a time t and the intances of future time steps $t + 1, \ldots, T$ are unknown.

In Chapter 1 of the thesis, we will present an overview of optimization problems tackling evolving data. First we present some notions of complexity theory that will be used during the whole document. Then, the framework of this thesis is presented as well as its current state of the art. Note that the framework was introduced fairly recently (in 2014) and more and more articles adressing it are published each year. In addition of being interesting in practical situations, some surprising behaviours were observed in the multistage context: some problems, polynomially solvable in their classical version, become **NP**-hard and/or inapproximable in their multistage version even in very "easy" restricted settings; some other problems, **NP**-hard in their classical version, tend to be easier to approximate in their multistage version than polynomially solvable problems (in their classical version). Finally are presented some other approaches of the literature dealing with evolving data and

sharing some properties with the multistage framework.

Then, in Chapter 2, the MULTISTAGE KNAPSACK problem is addressed in the offline setting. Note that the studies of this chapter have been presented at MFCS (Mathematical Foundations of Computer Science) 2019 (Bampis et al. (2019c)). The main contribution is a polynomial time approximation scheme ($PTAS$) for the problem in the offline setting. Up to the best of our knowledge, this is the first $PTAS$ presented for a problem in its multistage version, contrasting with some inapproximability results showed for some polynomially solvable problems in their static (classic) version.

In Chapter 3, the multistage framework is studied for multistage problems in the *online setting*, i.e. at time t, instances at time $t+1, \ldots, T$ are unknown. When the lack of knowledge of the future has too much (bad) influence on the results one can obtain, the k-lookahead setting was studied, where at time t, instances at time $t+1, \ldots, t+k$ are known. The goal of these research were to measure, in the multistage framework, the impact of the absence of information on the future of data evolution. Contrasting with the multistage literature which mainly focused on minimization problems, we addressed a large family of problems in the multistage setting known as the subset maximization problems (note that the MULTISTAGE KNAPSACK problem is in this family of problems). The main contribution of this chapter was the introduction of a structure for these problems (tackling different kinds of models) and almost tight upper and lower bounds on the best-possible competitive ratio for these models. Note that these works have been presented at ESA (European Symposium on Algorithms) 2019 (Bampis et al. (2019b)).

Finally in the chapter 4 is presented a direct application of the multistage framework in a musical context. In the first part of this chapter is presented the musical problem studied, i.e. the TARGET-BASED COMPUTED-ASSISTED ORCHESTRATION problem, with some basic notions of musical theory and the history behind the study of the problem. Then our work is presented, which is a theoretical analysis of the TARGET-BASED COMPUTED-ASSISTED ORCHESTRATION problem, with **NP**-hardness and approximation results as well as some experimentations. Note that these works will be put on arXiv and submitted to a journal by the time of the thesis defense.

Chapter 1

Optimization with temporal aspects, a state of the art

1.1 Combinatorial optimization preliminaries

In the first section of this chapter, we will present some fundamental definitions and notions of combinatorial optimization used during the thesis. We will define a certain terminology and vocabulary for the entire document. First and foremost, we will address the theory of complexity with its main classes, then, we will develop the notion of approximation algorithms.

1.1.1 A brief introduction to the theory of complexity

As we said in the introduction of the thesis, in a classical optimization problem, given an instance, one is asked to find a feasible solution optimizing an objective function. The feasibility of a solution is defined by a set of constraints applied to, depending on the nature of the problem, nodes, edges, precedence and so on for graphs, objects for packing problems...

Let us define two types of problems, the decision problems and the combinatorial optimization problems.

Definition 1.1. (Decision problem)
A decision problem is defined as a set $\mathcal{I}$ of instances which is partitioned into $\mathcal{I}^+$ (YES-instances) and $\mathcal{I}^-$ (NO-instances). The goal is to determine if a given $I \in \mathcal{I}$ is a YES-instance or a NO-instance.

Definition 1.2. (Combinatorial Optimization problem) *A combinatorial optimization problem A is defined by:*

- *a given set of instances* $\mathcal{I}$
- *for each instance* $I \in \mathcal{I}$*, a set* $\mathcal{F}(I)$ *of feasible solutions*
- *for each solution* $x \in \mathcal{F}(I)$ *a value* $c_I(x)$*, assumed to be greater than or equal to* 0

One needs to find a feasible solution in order to optimize the objective function

Note that, any optimization problem can be associated with a decision problem where the problem is to determine, for a given k, whether there exists a solution of value at least (for a maximization problem) or at most (for a minimization problem) k.

An algorithm, based on Turing machines that we won't present here, is used to solve either the optimization problem or the decision problem. We call the *time complexity* of an algorithm the number of steps one has to go through in order to run it. We say that a problem is solvable in polynomial time if there exists some $c > 0$ such that any instance I of the problem can be solved by an algorithm running in time $O(|I|^c)$ with $|I|$ the size of the instance. Space is another type of resource addressed in complexity theory, defined by the *space complexity*, which is the space in memory an algorithm needs to run. We focus extensively on the *time complexity* in this thesis (we will sometimes use the term complexity of an algorithm and will be referring in this case to its *time complexity*).

That being defined, we can introduce two central classes of problems of the theory of complexity. The first one is called the class **P** (The following definitions of this section are for most of them taken from Escoffier (2005) which we refer the reader to for a more detailed introduction on complexity classes and approximation algorithms and techniques).

Definition 1.3. (The complexity class of problems **P**)
The complexity class **P** *of problems is defined as the set of all decision problems solvable in polynomial time by a deterministic Turing machine.*

We will encounter quite a lot of problems belonging to this class in the next sections. However, a large amount of problems are not known to be in the class **P**, i.e., we don't know any polynomial time algorithm able to solve them.

This observation leads us to introduce the second central class of problems, called the complexity class **NP**.

Definition 1.4. (The complexity class of problems **NP**)
The complexity class **NP** *is defined as the set of all decision problems solvable in polynomial time by a non deterministic Turing machine.*

Roughly speaking, the complexity class **NP** of problems is defined as the set of all decision problems for which it is possible to verify in polynomial time if an answer to a given instance of the decision problem question is yes.

The class **NP** gathers a large amount of problems, in addition to all the problems of the class **P**, solvable and thus verifiable in polynomial time.
These definitions give us the possibility to introduce the central question of the theory complexity: does **P**=**NP**? The hypothesis stating **P**≠**NP** being widely accepted, the majority of the results presented in this thesis are relevant and presented under this hypothesis.
In order to be able to gather a large amount of problems either in **P** or **NP** and given a problem to be able to determine if it is in one, the other or none of the class, Cook and Karp introduced and used, in two of the most fundamental articles of the theory of complexity (Cook (1971) and Karp (1972)), the notion of reduction between two decision problems.

Definition 1.5. (The Karp reduction)
Let A and B be two decision problems. We say that A is reducible to B if there is a function f such that:

- *for all instances $I \in A$, $f(I) \in B$*
- *the answer to the question of the decision problem A is yes for an instance I^* if and only if the answer to the question of the decision problem B is yes for the instance $f(I^*)$*
- *it takes a polynomial time to compute f*

Such a reduction gives the possibility to transfer the result of belonging to a class from one problem to another. Indeed, following Definition 1.5, if a decision problem B is in **P** and a decision problem A is reducible to B, then A is in **P** too. This affirms that if it is possible to solve a decision problem B in polynomial time, thus it is also possible to solve an easier decision problem A in polynomial time.
In Cook (1971), the author showed that all problems in **NP** are reducible to the well known SAT problem, i.e. all problems in **NP** are easier than the SAT problem. This observation leads to the definition of the complexity classes **NP**-hard and **NP**-complete.

Definition 1.6. (The complexity class of **NP**-hard problems)
*The complexity class **NP**-hard is defined as the set of all the decision problems B such that any decision problem $A \in$ **NP** is (Karp)-reducible to B.*

Later in the document, we will also refer to **NP**-hard optimization problems, an optimization problem is **NP**-hard if its decision version is **NP**-hard.

Definition 1.7. (The complexity class of **NP**-complete problems)
The complexity class **NP**-complete *is defined as the set of all decision problems that are both in* **NP** *and* **NP**-hard.

We introduced some central notions of the theory of complexity (see Garey and Johnson (1979) for a more detailed presentation on theory of complexity), let us now present approximation algorithms and their motivations.

1.1.2 Approximation algorithms

As said in the previous section, a vast amount of decision problems are known to be NP-hard i.e. not polynomially solvable if **P**≠**NP**, which is the central hypothesis of the theory of complexity. The hardness of those problems make real world applications and implementations really difficult or even impossible. It is then naturally that about 30 years ago approximation algorithms were introduced. Those algorithms give some solutions in reasonable time, more precisely in polynomial time, even if they are not optimal. The quality of a solution output by such an algorithm is defined by a ratio, comparing the value of its solution to the optimal value.

Formally, for an optimization problem P and an instance I of P, the quality of an approximation algorithm is given by the ratio:

$$\frac{SOL(I)}{OPT(I)}$$

with *SOL(I)* the value of the solution returned by the algorithm on the instance I and computed in polynomial time and *OPT(I)* the value of an optimal solution. The value of this ratio has to be at most (resp. at least) one for maximization (resp. minimization) problems. Note that in this document, we only consider problems with positive objective functions.
We say that an algorithm is a ρ-approximation algorithm for an optimization problem if it has been shown that for any instance I of the problem, i.e. especially in the worst case scenario, the ratio $\rho(I)$ is verified, i.e. if $SOL(I) \leq \rho(I)OPT(I)$ for minimization problems and $SOL(I) \geq \rho(I)OPT(I)$ for maximization problem. One is looking for an approximation algorithm with a ratio as close as possible to 1.

Some problems have a known constant approximation ratio and said to be in a complexity class called **APX**:

Definition 1.8. (The complexity class of problems **APX**)
The complexity class **APX** *of problems is defined as the set of all optimization problems solvable by a polynomial time algorithm A with a constant approximation ratio, i.e., there exists $\rho \in \mathbb{R}^*$ such that A is a ρ-approximation algorithm.*

Let us finally introduce two last classes of complexity that we will largely use in this thesis and that are known to be in **APX**.

The first one is called **PTAS**.

Definition 1.9. (The complexity class **PTAS**)
The complexity class of problems **PTAS** *is defined as the set of all optimization problems admitting a Polynomial Time Approximation Scheme (PTAS). A PTAS is an algorithm that takes as input an instance of an optimization problem and any $\epsilon > 0$, outputs a $(1-\epsilon)$-approximate solution for maximization problems, ($(1+\epsilon)$-approximate solution for minimization problems) and has a complexity polynomial in the instance size $|I|$. The dependency in ϵ can be arbitrary.*

We say that a problem is **APX**-hard if there is no $PTAS$ under the hypothesis that **P**$\neq$ **NP**.

The other one is the more restricted class called **FPTAS** where the complexity of the algorithm is required to be polynomial in both the instance size and $\frac{1}{\epsilon}$. Formally:

Definition 1.10. (The complexity class **FPTAS**)
The complexity class of problems **FPTAS** *is defined as the set of all optimization problems admitting a Fully Polynomial Time Approximation Scheme (FPTAS). A FPTAS is a PTAS whose complexity is polynomial both in the instance size $|I|$ and in $\frac{1}{\epsilon}$*

A lot of other complexity classes dealing with approximation algorithms are studied in the literature (see Ausiello and Paschos (2018) for a detailed survey on the approximation preserving reductions).

Let us now give the definition of a pseudo polynomial problem:

Definition 1.11. (Pseudo-polynomial problem)
Let I be an instance of an optimization problem A (with its decision version being ***NP****-complete) and $C(I)$ the absolute value of the largest number in I. We say that A is pseudo-polynomial if there exists an algorithm whose complexity is polynomial in the size of the instance, i.e. $|I|$, and in $C(I)$.*

Finally, let us give the definition of the complexity class strong **NP**-hard.

Definition 1.12. (The complexity class Strong **NP**-hard)
The complexity class strong ***NP****-hard is defined as the set of all decision problems that remain* ***NP****-hard even when all their numerical values are bounded by a polynomial in the length of the input.*

Results on the complexity classes defined in this section will be presented in the different chapters of this thesis.

1.2 Multistage optimization

Beyond the complexity issues, some real world problems naturally induce some difficulties regarding their own data. Indeed, in some cases, one only has a partial knowledge of the problem data or in some other cases the data will be brought to evolve over time. To deal with these kind of temporal difficulties, a lot of approaches have been developed, most of them since the early 1980's, responding to different applications and needs.
We will first develop the multistage optimization which is the main topic of the thesis, illustrate it with an example and give its definition in Section 1.2.1. Then we will present the actual state of the art of the framework in Section 1.2.2.
We will next focus on a few other optimization frameworks coping with evolving data, being extensively studied and/or sharing some properties with the multistage one in Section 1.3.

1.2.1 Introduction and definition of the multistage framework

In the multistage framework, a decision maker is given a time horizon with T discrete time steps and a sequence of instances of a problem, one for each time step, he needs to solve. To do so, one needs to build a sequence of solutions, i.e. a set of feasible solutions, one for each time step, optimizing the objective function of the problem.

First, let us illustrate this approach with an example. Consider a company owning a set $N = \{u_1, \ldots, u_n\}$ of production units. Each unit can be used or not; if u_i is used, it spends an amount w_i of a given resource (energy, raw material,...), and generates a profit p_i. Given a bound W on the global amount of available resources, the static Knapsack Problem aims at determining a feasible solution that specifies the chosen units in order to maximize the total profit under the constraint that the total amount of the resource does not exceed the bound of W.

	u_1	u_2	u_3
w_i	1	1	2
p_i	3	1	7
W	2		

	u_1	u_2	u_3
w_i	1	2	3
p_i	2	5	5
W	3		

1.1: Two instances of the Knapsack problem

Two distinct instances of the Knapsack problem are presented in Figure 1.1. To get the optimal solution, one has to take in the left instance the production unit u_3, obtaining a profit of 7 and using all the resources, called also capacity of the knapsack, i.e. 2. Otherwise, the profit would be lower or the amount w_i of resources

would exceed the global available resources. In the right instance, one has to take the units u_1 and u_2, obtaining a profit of 7 and using again all of the resources, i.e. 3.

In a temporal setting and more precisely in the multistage setting we focus on this thesis, a company would have to decide a production plan over a time horizon $t = 1, 2, \ldots, T$, of, let us say, T days. The company here needs to decide a production plan for each day of the time horizon, given that data (such as prices, level of resources,...) usually change over time. This a typical situation, for instance, in energy production planning (like electricity production, where units can be nuclear reactors, wind or water turbines,...), or in data centers (where units are machines and the resource corresponds to the available energy). Moreover, in these examples, there is an extra cost to turn ON or OFF a unit like in the case of turning ON/OFF a reactor in electricity production (Rottner (2018)), or a machine in a data center (Scheideler and Hajiaghayi (2017)). Obviously, whenever a reactor is in the ON or OFF state, it is beneficial to maintain it in the same state for several consecutive time steps, in order to avoid the overhead costs of state changes (even if it is important to note that the real problem is much more complicated). Therefore, the design of a production plan over a given time horizon has to take into account both the profits generated each day from the operation of the chosen units, and the potential *transition profits* from maintaining a unit in the same state for two consecutive days. Thus, in the MULTISTAGE framework, instead of having only an instance of a problem and seeking a feasible solution optimizing its objective function, we have a sequence of instances of a problem, one for each time step and one is asked to seek a sequence of feasible solutions, one for each time step, reaching a trade-off between the optimality of the solutions and the stability of consecutive solutions.
The problem can be formalized as follows. We have a given time horizon $t = 1, 2, \ldots, T$, and a sequence of knapsack instances $I_1, I_2, \ldots, I_T$, one for each time step, defined on a set of n productions units, also called objects in the KNAPSACK problem. In every time step t we have to choose a feasible knapsack S_t of I_t, which gives a *knapsack profit*. Taking into account transitions costs, we measure the stability/similarity of two consecutive solutions S_t and S_{t+1} by identifying the objects for which the decision, to be picked or not, remains the same in S_t and S_{t+1}, giving a *transition profit*. We are asked to produce a sequence of solutions $S_1, S_2, \ldots, S_T$ so that the total *knapsack profit* plus the overall *transition profit* is maximized.
There are a lot of other applications where it is beneficial to use a multistage framework. One of them is the FACILITY LOCATION problem, which we will develop later in this section. Another one is a musical application called the target-based computer-assisted orchestration. We will focus on this latter application in details in the last chapter of the document.

Let us now look at the example of Figure 1.1 and consider both instances as a

sequence of instances of a MULTISTAGE KNAPSACK problem with two time steps, the left instance being the instance at the time step 1 and the right instance the one at the time step 2. In order to compute a *transition profit*, we need to introduce the notion of the bonus. Let us say that for this example, one gets a bonus $B = 1$ for each object if the object is taken or not taken for two consecutive time steps, i.e. if the decision remains the same between time steps 1 and 2. Thus, the global *transition profit* is equal to B times the total number of decisions that remain the same between two consecutive time steps. For instance, if we take the objects of the static optimal solution, i.e., object u_3 for the first time step and objects u_1 and u_2 for the second time step, the *knapsack profit* is equal to 14, 7 at both time steps. However, the value of the *transition profit* is equal to 0, none of the decisions remains the same (the object u_1 is taken only at time step 1 whereas the objects u_2 and u_3 are taken only at time step 2), so we have a global reward equal to 14. The optimal solution consists of taking only the object u_3 at both time steps, getting a *knapsack profit* equal to $7 + 5 = 12$ and the max value of the *transition profit*, as object u_1 and u_2 are not taken at both time steps and u_3 is taken at $t = 1$ and $t = 2$, i.e. $3B = 3$; the global reward is then equal to $12 + 3 = 15$.

The example introduces the KNAPSACK problem in its multistage configuration. This problem is the main subject of the second chapter where we will develop its particularities and properties.

Let us now give a formal definition of an optimization problem in its multistage version.

Definition 1.13. (Multistage Optimization problem). *In a Multistage Optimization problem, given*

- *a combinatorial optimization problem $\mathcal{P}$;*
- *a number $T \in \mathbb{N}^*$ of time steps;*
- *for any $t \in T$, an instance I_t of the optimization problem $\mathcal{P}$ with a feasible set $\mathcal{F}_t$*
- *For each $t = 1, \ldots, T-1$ a function $b(S_t, S_{t+1})$ associating to each couple of feasible solutions $S_t \in \mathcal{F}_t$ and $S_{t+1} \in \mathcal{F}_{t+1}$ the transition bonus/cost for resp. maximization/minimization problems between two solutions at consecutive time steps*

With $\mathcal{S} = (S_1, \ldots, S_T)$ a sequence of feasible solutions over the time horizon, the

objective function is the value of a solution sequence $\mathcal{S}$*:*

$$f(\mathcal{S}) = \sum_{t=1}^{T} p_t(S_t) + \sum_{t=1}^{T-1} b(S_t, S_{t+1})$$

We will use the term profit/cost for $p_t(S_t)$ *for resp. maximization/minimization problems, transition bonus/cost for the transition bonus/transition cost* $b(S_t, S_{t+1})$ *for resp. maximization/minimization problems, and value of a solution* $\mathcal{S}$ *for* $f(\mathcal{S})$*; The goal is to determine a solution sequence of maximum/minimum value for resp. maximization/minimization problems.*

Let us look back at the MULTISTAGE KNAPSACK problem example. The *profit* function $p_t(S_t)$ would be the *knapsack profit* for $t = 1, 2$, with S_t a solution at a time step t. The *transition* function, here a *bonus* function, would be $b(S_1, S_2) = |(S_1 \cap S_2) \cup (N \setminus S_1 \cap N \setminus S_2)|$. The goal is then to maximize the sum $p_1(S_1) + p_2(S_2) + b(S_1, S_2)$.

The definition presented above is one of the several possible ways to define a problem in a multistage version.
Indeed, here the global objective function consists of the sum of the *profit/cost* function and the *transition* bonus/cost.
This definition is debatable but seemed relevant for the problems encountered during the thesis. Moreover, it follows the original definition of the multistage framework presented in Gupta et al. (2014) and in Eisenstat et al. (2014). It is in fact possible to define the objective function this way when one is able to compare the profit/cost function value and the transition bonus/cost function value. For example, it is possible in the previously introduced applications, i.e. in, energy production planning or in data center problems, since the cost of the production and the cost of switching ON/OFF a reactor/server are directly comparable.
Another way of defining a multistage version is with a multi-objective approach. Indeed, we will see later in this section that for some problems such as the VERTEX COVER problem, a multistage version of the problem was introduced where one is asked to minimize a number of selected nodes (the classical VERTEX COVER problem objective function) and at the same time to minimize the number of modifications regarding the selected subset of nodes for two consecutive time steps. In some cases it is also relevant to bound the number of changes between two consecutive time steps or even to bound them on the whole time horizon.

This latter point on the objective function is one aspect subject to differ between several possible definitions of the framework, depending on the nature of the studied problem.

Another aspect is the way data may evolve during the time horizon. There are indeed different possibilities of data evolution that will be developed later in chapter 3.

In the third chapter, we will present some results for a class of problems called the Multistage Subset Maximization problems (a formal definition will be given in the corresponding chapter) and study these problems in terms of their different types of data evolution.

Then, another key point in the definition of a problem in its multistage version is the definition of the transition bonus/cost (maximization/minimization version respectively) function.
Indeed, several ways of defining such a transition bonus/cost function in order to measure the stability of a sequence of solutions can be relevant in real world applications.
For example, for the Facility location problem that we will develop into details in the next section, it could be interesting to take into account in the transition function:

- the cost induced by the opening of a facility
- and/or the cost induced by clients switching from different facilities

Two kinds of transition bonus, namely the *Intersection Bonus* and the *Hamming Bonus*, will be studied in the third chapter of the thesis. We will see that, depending on the bonus studied, different results are observed.

Finally, in order to go further into the development of the multistage framework and present the current state of the art of the multistage framework, we need to present different temporal settings on the knowledge of the data over the time horizon. We will focus on three settings studied in the literature:

1. The offline setting: one has a complete knowledge of the instance over the time horizon (this was the case of the example of Figure 1.1 where we know the whole sequence of instances of the Multistage Knapsack problem, it would be the case for the electricity planning problem where one is asked to find a solution given a predicted fixed data set; a musical application will be developed in the fourth chapter in this setting, where one has a given data set over a time horizon);

2. The online setting: at a time step t, one only knows the data for today, i.e. we have no information regarding the instances at time steps $t+1, \ldots, T$. In our definition, we also assume that we know the number T of time steps of the time horizon. The online setting will be developed later in the chapter as it is

an extensively studied case of temporal optimization. (Note that this setting will be presented in detail in Subsection 1.3.2)

3. The k-lookahead setting: at a time step t, one knows the data for today and the next k days. This setting is tightly linked to the online case as it is often used when no results can be obtained in the online case. In our definition, we again assume that one knows the total number of time steps T of the time horizon.

The third chapter deals with MULTISTAGE SUBSET MAXIMIZATION problems in the online and k-lookahead settings, with different possible types of data evolution and transition bonus functions.

1.2.2 State of the art: the multistage framework

In this section, we will cover the current state of the art of the multistage framework. To do so and for the sake of clarity, we will present it problem by problem. We will give a definition of the problems and develop their approaches and current results. The multistage framework defined formally in the previous section follows the direction presented fairly recently by Gupta et al. (2014) and Eisenstat et al. (2014) who covered different problems.

Matching and Perfect Matching problems

In the static *matching* problem, given a graph, we need to find a set of edges with no vertices in common. A *matching* is called a *perfect matching* if all vertices of a graph are covered by the set of selected edges.
In its multistage version, the *perfect matching* problem is called the PERFECT MATCHING MAINTENANCE and consists of keeping the perfect matching property over the time horizon, i.e. at each time step, while the cost function on the edges of the graph and cost value for adding new elements is subject to changes over the time horizon.

In Gupta et al. (2014), they showed that the PERFECT MATCHING MAINTENANCE problem becomes surprisingly inapproximable, even in the offline case. This negative result is the first observation of the hardness induced by the multistage framework. Indeed, the majority of the problems studied for now and considered as easy, i.e., polynomially solvable or easily approximable, in their static form, become really hard in this framework, even for limited restricted instances, in the offline case and for a small number of time steps.

The negative result on the PERFECT MATCHING MAINTENANCE was improved a few years later in Bampis et al. (2018a). Indeed, in Gupta et al. (2014), it was shown that the problem was inaproximable for instances with as least 8 time steps but the question for less time steps and for specific instances such as bipartite graphs was left open. Bampis et al. (2018a) addressed this open question and proved that the problem is hard to approximate even for 2 time steps and in bipartite graphs. Then, they showed other negative results. Even the metric version of the problem where the triangle inequalities are satisfied, called MINIMUM MULTISTAGE PERFECT MATCHING, is **APX**-hard. However, in the case where the number of time steps is equal to 2 or 3, they presented an algorithm with constant approximation ratio. Finally, they also showed that the problem in its maximization version, with the complementary objective function, was also **APX**-hard even though it has a constant approximation ratio.

Very recently, in Chimani et al. (2020), the authors looked again at the MULTISTAGE MATCHING problem and a variant where the number of overall modifications is as small as possible. They improved the results presented in Bampis et al. (2018a) by showing the NP-hardness of the problem in an even more restricted case than the one presented in Bampis et al. (2018a). They also presented a new approximation algorithm that does not require the restrictions needed before.

Facility Location problem

In the FACILITY LOCATION problem, one is asked, given a set of clients and facilities, to find the best connections of clients to facilities such that the tradeoff between, here a sum of two objectives is minimum. The first objective is the distance objective, corresponding to the sum of distances from the clients to facilities, each client has to be connected to a facility and as close as possible to their facilities. The second is an opening cost paid for each opened facility. Thus, one has to select the least amount of facilities to open such that the sum of distances between clients and facilities and paid opening costs is minimum.
The multistage version of the problem, called the DYNAMIC FACILITY LOCATION PROBLEM, shares the same two objectives with the static version. However, it has a third objective, also summed with the other two objectives, called the non-negative client switching cost. Indeed, a cost is paid for switching clients between different facilities and two consecutive time steps. This last function measures the stability of the solution over the time horizon.

For both the static and the multistage versions of the problem, there exists a variant in the definition of the opening cost per facility. Indeed, one has to pay either a *fixed* opening cost to open a facility, the facility remains open for the whole

time horizon, or a *hourly* opening cost that in which case must be paid for each facility opened at each time step.

In Eisenstat et al. (2014), they addressed the FACILITY LOCATION problem in the multistage framework. The application underlying the study of this problem is another example of a possible application and need of stability in the decisions made over a time horizon. In our era, a huge amount of data are collected on social networks and their studies are more and more important. These networks quickly evolve in time and it is thus important to be able to analyze such data in a dynamic environment. Even though the FACILITY LOCATION problem has been widely studied in temporal settings, the notion of stability was not really looked at. Taking into account this stability, here represented by clients moving or not among a set of facilities, gives the possibility to understand better clients behaviour and highlights the impact of clients moving through different facilities. It thus offers better results in realistic situations, giving a stable group partition of the network. Eisenstat et al. (2014) proposed a logarithmic approximation for the problem and gave a matching inapproximability result in restricted instances respecting the triangle inequality, with only one client, two possible positions and a fixed opening cost. These instances admit a constant approximation ratio in the static framework.

In An et al. (2017), the authors treated one open question left in Eisenstat et al. (2014) and presented a constant factor approximation algorithm using LP-rounding techniques for the version of the problem where opening costs are paid *hourly*.

Spanning Tree problem

In the SPANNING TREE problem, one is asked to find a tree A in a graph such that all vertices of the graph are connected in A and that the total edge weight of the selected edges is minimum.
In its multistage version, given a set of instances in a graph and a number T of time steps, we need to find a set of spanning trees A_t for $t \in T$, one for each time step. Indeed, for each time step, we pay the price for the tree, as in the classical static version of the problem and also $|A_t \setminus A_{t-1}|$ for the modification of edges between two consecutive time steps.

In Gupta et al. (2014), they studied the MULTISTAGE MATROID MAINTENANCE problem, a problem with some costs induced by changing decisions at some time steps on some edges, and with an application quite similar to the one presented in our example. As a special case, the problem can be seen as a natural multistage version of the SPANNING TREE problem. They looked at both the online and offline versions of the problem and gave logarithmic approximation algorithms in both

cases using some LP-rounding, randomized algorithms and matroid techniques (see Vazirani (2013) for details on approximation algorithms). They improved a result from Buchbinder et al. (2016) and Buchbinder et al. (2014) who looked at a fractional version and later a more general version of the problem.

List Update problem

In the LIST UPDATE problem, given a set of items with values corresponding to their distances to the head of a track, a set of requests (these requests can be in the offline or in the online setting) and a constraint on a fixed position for each item at each time step, one has to give for each item an assignment to a position in the track for the whole time horizon minimizing the cost of all the requests.
In a multistage version of a closed variant of the problem, called the DYNAMIC MINIMUM LINEAR ARRANGEMENT problem, there is no constraint on the position of an item at the beginning of each time step and one has to pay a cost for moving an item from one position to another between two time steps, measuring this way the stability of the solution.

In Olver et al. (2018), the authors studied this multistage problem in both the offline setting, presenting a polylogarithmic approximation algorithm, and the online setting, giving a logarithmic lower bound on the competitive ratio of any randomized algorithm against an oblivious adversary.

Cut, Vertex Cover and some prize-collecting problems

Last year, Bampis et al. (2020) studied a wide variety of discrete minimization problems in a multistage framework. The idea was to highlight the possibility of using some LP-rounding techniques in a multistage framework.
They presented some surprisingly positive results. Indeed, for some minimization integer programming problems refereed to as *monotone* problems, polynomially solvable in their static version, such as the MIN CUT problem, they proved that the problems remain polynomially solvable in their multistage version. These results contrasts with the hardness results presented before on problems solvable by a polynomial algorithm in their static version and becoming **NP**-hard in the multistage framework.
They also showed that VERTEX COVER, as well as some other problems, remain 2-approximable in the multistage framework. Finally they introduced a new rounding technique designed specially for multistage problems, and proved some constant approximation ratio for multistage versions of the PRIZE-COLLECTING STEINER TREE problem and the PRIZE-COLLECTING TRAVELING SALESMAN problem.

Santa Claus problem

In the SANTA CLAUS problem, also called MAX MIN FAIR ALLOCATION problem, given a set of resources and agents, one is asked to find an allocation of resources to agents such that the value of the *worst-off* agent is maximum.
The problem in the multistage framework, called the OVER-TIME MAX MIN FAIR ALLOCATION problem, in addition of sharing the same objective as its static version, has a *transition revenue* evaluating the stability of solutions between two consecutive time steps. Indeed, one has a bonus for keeping the same decision over the time horizon, which corresponds here to resources remaining on a same agent for two consecutive time steps. The global objective function sums these two objectives.

In Bampis et al. (2018b), the authors addressed this problem. They studied the problem in the offline, online and *1-lookahead* settings and for different kinds of evolving instances. The study of different kinds of evolving instances is crucial in temporal optimization and will be developed in the third chapter. They showed that in its offline version, the problem is much harder than its static version. It becomes **NP**-hard even for simple instances without restriction whereas these instances are trivially solved in the static version of the problem (instances where the set of feasible solutions is static over the time horizon, i.e. in this case instances where the set of resources and agents are the same during the whole time horizon and every resource can be allocated to any agent). Regarding the online version of the problem, they proposed a constant competitive ratio for instances without restriction using an approximation algorithm for the static case as a subroutine. For instances where the feasible set of solutions may change between different time steps, they showed that the problem has no bounded competitive ratio in the online setting. Finally, they looked at the *1-lookahead* version of the problem, in the same instance evolving setting and proposed a constant approximation algorithm using again an approximation algorithm for the static case as a subroutine.

The OVER-TIME MAX MIN FAIR ALLOCATION problem is the unique multistage problem presented in the literature up to our knowledge addressing a multistage maximization problem; all other presented multistage problems are minimization problems. We will develop in the second and third chapters of this thesis a study on a large class of multistage maximization problems called the MULTISTAGE SUBSET MAXIMIZATION problems and a special offline study on the MULTISTAGE KNAPSACK PROBLEM.

1.2.3 State of the art: Parameterized multistage optimization studies

Let us now develop an alternative definition of the global objective function of a multistage problem different from the one presented in the previous section, since here such a problem is addressed as a multi-objective problem. In the following problems the global objective function is still a mono-objective function but with some stability constraint. Indeed, whereas the problems presented before share the notion of *transition profit* or *transition cost*, ensuring that the solution does not change too much over the time horizon, here the stability is represented by a *bound* over the number of changes in a decision one can make between two time steps. Let us illustrate this function more precisely with the VERTEX COVER problem.

Informally, we say that a problem is (or is in) **FPT** (being a complexity class) if it is a fixed parameter tractable problem, i.e. if it is solvable in time $f(k).|x|^{O(1)}$ with f any computable function and $|x|$ the size of the instance. Furthermore, we say that a problem is **W[1]**-hard if is not in **FPT**.

Vertex Cover problem

In the classical static version of the VERTEX COVER problem, given a undirected graph, one is asked to find the smallest subset of vertices such that all edges contain at least one endpoint in the cover.
In the multistage version of the VERTEX COVER problem studied in Fluschnik et al. (2019), one is asked to find a small subset of vertices covering the edges of a temporal graph, i.e. a subset of vertices at each time step in a set of graphs with a fixed set of vertices but with a set of edges evolving over the time horizon, such that the number of changes between two solutions of two consecutive time steps *does not exceed a given parameter*.
Note that a set of graphs, with either the set of vertices or the set of edges changing over a time horizon is called a temporal graph. It is very well studied in the temporal optimization literature and will be presented into more details further in this chapter.

In Fluschnik et al. (2019), the authors studied the above version of the MULTISTAGE VERTEX COVER problem. They showed that in the case where the number of time steps of the problem can be arbitrary, i.e. not a constant, the MULTISTAGE VERTEX COVER is **NP**-hard even for instances with only two vertices and one edge at every time step. Note that these instances are trivial in the static case. They also proved that the problem becomes **NP**-hard even for two time steps and on restricted instances where the graph of the first time step is a path and the one of the second time step is a tree. Then, they showed that if the parameter corresponding to the number of changes allowed between two time steps is smaller than two

times the size of the cover, the problem is not fixed-parameter tractable according to this transition parameter (i.e. **W[1]**-hard). The problem appears to be **FPT** when parameterized by the same transition parameter otherwise (see Downey and Fellows (2012) for a survey on parameterized complexity).

In Heeger et al. (2019), the authors also looked at the multistage framework in the parameterized version, with a constraint on the number of allowed modifications in the solutions selected in two consecutive time steps. A contrario to the one presented before where the constraint was local, they introduced a global constraint with a bound on the total number of modifications. They introduced the GLOBAL MULTISTAGE VERTEX COVER problem and proved that it is **FPT** when parameterized by both the upperbound of the size of the solution and by the number of time steps, but **W[1]**-hard when only parameterized by the upperbound of the solution size.
The authors also addressed a global multistage version of the MIN CUT problem giving some **W[1]**-hardness results. They finally highlighted that some polynomial-time solvable problems are harder, computationally speaking, in a global multistage framework than global multistage versions of **NP**-hard problems.

$s-t$ Path problem

In the $s-t$ PATH problem, given a directed weighted graph containing two nodes s and t, one is asked to find a shortest path between s and t.
In the multistage version of the problem, the MULTISTAGE $s-t$ PATH problem, given a temporal graph with the same vertex set, but with changing edges over the time horizon, one is asked to find a minimum path between s and t in the temporal graph so that it is as stable as possible. The stability here is represented by a bound on the number of authorized modifications between two time steps, either on the vertices or on the edges.

In Fluschnik et al. (2020), the authors proved that for very restricted instances with only two time steps and maximal degree of any vertex of the temporal graph smaller than or equal to 4, the problem becomes NP-hard (for both variants of the problem). They also looked at the parameterized complexity and showed some results **W[1]**-hardness, when the parameter is the size of the returned solution. They were also the first to address the notion of dissimilarity, in contrast with the similarity, stability, of two consecutive solutions previously studied. In this variant, they presented this time an **FPT** algorithm in the size of the returned solution.

Committee Election problem

In Committee Election problems, given a set of agents, a set of candidates and a voting function, called voting profiles, one is asked to find the smallest committee such that it has a sufficient number of approvals.
Two variants of the Multistage Committee Election exist (Bredereck et al. (2020)). In the first one, given a set of agents, a set of candidates, a sequence of voting profiles over a time horizon and an integer k, one is asked to find a sequence of small committees, one for each time step, such that the number of approvals is sufficiently large and the size of the symmetric difference of two consecutive committees over the time horizon is lower than k. This version is called the Conservative Multistage Plurality Voting problem.
In the other variant called the Revolutionary Multistage Plurality Voting problem, the problem is the same as the one presented above, but this time the symmetric difference of two consecutive committees in the time horizon has to be greater than a given integral parameter.

In Bredereck et al. (2020), the authors looked at both variants of the multistage problem. They showed **NP**-hardness of both problems when the number of agents is fixed and gave some parameterized complexity results. Indeed, they showed that both problems are **W[1]**-hard when parameterized by the number of stages.
At last, they presented a polynomial algorithm for the revolutionary problem, while the conservative variant remains **NP**-hard, when the fixed parameter on the symmetric difference between two consecutive time steps is a constant.

1.2.4 Contributions

We presented the current state of the multistage framework. This setting is fairly recent but more and more studies appear with a lot of publications within the last few years. Indeed, authors keep improving the results and observations introduced in 2014 by Gupta et al. (2014) and Eisenstat et al. (2014). Some surprising properties are being highlighted such as strong negative results (**NP**-hardness, inapproximability, **W[1]**-hardness) for very restricted instances (even sometimes for trivial instances in the static framework) and only a few time steps, problems polynomially solvable in a static version becoming harder than **NP**-hard problems, and also a few positive results, some problems keeping the properties of their static version in the multistage framework. The community looked at a few different settings regarding the evolution of the instances, the bonus/cost representing the stability, the knowledge over the time horizon, or the parameterized version of the framework. Some of them will be developed later in the second and third chapters. The framework has still a lot of open questions and settings to be looked at.

At the beginning of the thesis, the vast majority of the problems studied in the framework were minimization multistage problems. This was one of the reasons that motivated us to focus on maximization problems. As said previously, we studied a large class of maximization problems in different settings; this will be detailed in the second and third chapters of this document.

1.3 Some other approaches tackling evolving data

As stated in the introduction of this chapter, optimization in dynamic environment gathers a wide variety of research branches. They were, for most of them, developed in the last 30 years in parallel of a lot of different approaches such as the approximation framework, feeding one another continuously.
We will develop here different branches of research taking into account optimization problems with evolving data sets. We will see that some of them are widely studied in the literature and that some others are closely linked to the multistage framework. For each temporal approach presented, we will focus on the similarities with and differences from the multistage framework.

The structure of this section on other temporal approaches was strongly inspired by the survey on temporal optimization presented in Boria and Paschos (2011).

1.3.1 Reoptimization

We will first develop the reoptimization paradigm. The notion was introduced in Schäffter (1997) where a scheduling problem was addressed and later developed in Archetti et al. (2003) where the authors addressed the TRAVELLING SALESMAN problem, a very well-known **NP**-hard problem. In both articles, two distinct steps were considered. In a first step, an algorithm gives an optimal or approximate solution for a **NP**-hard problem on an initial instance. The reoptimization focuses on the second step where the instance is perturbed, i.e., we perform vertex or edge deletions for graph problems, changing values for numerical problems... One is then asked to maintain the optimality/approximation ratio of the solution. Note that the perturbation in the reoptimization context is small, e.g., only one vertex or edge is deleted for a graph problem.
These two steps give direct properties for the problem in their reoptimization version. Indeed, the result of **NP**-hardness of a problem typically holds in the reoptimization framework, otherwise one could solve any **NP**-hard problem using reoptimization algorithms and starting from an empty instance. However, some problems are known to be hard to approximate in their static version and become **APX** or even admit a PTAS in the reoptimization framework (it is the case for example for the MAX

INDEPENDENT SET problem). This is why the majority of the results presented in this framework concern approximation algorithms. More generally, two kinds of results can be achieved for **NP**-hard problems in their reoptimization version:

- The reoptimization algorithm gives with a better running time an optimal or approximate solution as good as the solutions given by the best known algorithm for the problem in its static version;
- The reoptimization algorithm gives in polynomial time a better approximation ratio than the best one known for the problem in its static version.

A reoptimization algorithm is thus closely linked to the multistage framework, as it has to maintain a certain quality in the solution. To do so, one has to look at the solution already found and try to adapt it for the new instance. This often implies keeping a huge part of the solution in the new instance and thus keeping the solution stable. The main differences with the multistage framework are:

- the instance modifications appear locally and affect only one object or constraint at each time step;
- There are (generally) two steps: one has to use the solution found at the previous time step to find another solution locally without taking into account in its value the solution value for the previous time step. Thus, the solution is local for every time step.

The reoptimization approach was also addressed with the notion of stability. In Cohen et al. (2016), the authors looked at a close variant of the reoptimization approach where one, in addition to looking for a good reoptimization solution, has to find a solution close to the initial one.

Example 1.1. *Train scheduling*
In a train station, an algorithm has to find a assignment over a time horizon for trains to its platforms, so that the corresponding schedule problem is optimized. It is relevant here to consider spending a lot of computing time in order to output the best possible solution. However, in real world situations, some perturbations and problems can occur, malfunction of a train, breakdown. . ., affecting the feasibility and quality of the presented solution. In this case, spending a lot of time seeking for a new optimal solution from scratch is not a relevant option and a simple adaptation of the previous optimal solution would be way more beneficial. Indeed, here, dealing with only one perturbation at a time, one can use some reoptimization algorithm and output a solution in a short amount of time, i.e. in polynomial time, based on the previous solution.

Note that in the presented example, both the quality of the reoptimization solutions and the number of modifications matter, as in the multistage framework.

1.3.2 Online optimization

Let us now address the theory of online algorithms. We already discussed it briefly in the previous section and will develop it into more details. A complete survey, on which we based ourselves is presented in Albers (1997).
The central idea behind a lot of frameworks dealing with evolving environments is the ignorance of the decision maker regarding the behaviour of the problem instances over a (future) time horizon. It emphasizes the fact that a pure offline analysis is often not very realistic and can not be computed directly to a vast amount of real world applications. In an online setting, one has only a partial knowledge of the problem instance; the instance will be revealed step by step over a given time horizon and the algorithm needs to build the solution step by step, without the complete knowledge of the data. It is important to notice that, once a decision is made by an online algorithm, it can not be changed when new data are revealed. It contrasts here with reoptimization algorithms, for example, which are allowed to modify their decisions over the time horizon.
The study of the online theory began in the end of the seventies; in Sleator and Tarjan (1985) the authors addressed the main criteria used to analyse online algorithms: comparing the online solution and the offline optimal solution. This analysis was later given the name of competitive analysis in Karlin et al. (1988), with a ratio for the *online algorithm* called the competitive ratio. The offline optimal solution is computed with a full knowledge of the instances over the whole time horizon. This ratio is given in the worst case scenario, and one is looking for a tight ratio, i.e. no competitive algorithms can do better.
In specific cases when only bad competitive ratios can be obtained (some problems are not competitive in the online setting) the k-lookahead setting, introduced previously, where at a time step, one knows the data for today and the next k days, can be useful.
Another approach leading to better competitive results is the use of randomness. Indeed, random online algorithms are very studied in the online theory. Their competitive ratio are defined according to an adversary, that does not necessarily have a full knowledge of the decisions made by the randomized online algorithm.
Three adversary variants are extensively looked at:

- The oblivious adversary: the adversary knows the randomized online algorithm but has to generate its decisions over the whole time horizon without knowing the decisions made by the randomized online algorithm;

- the adaptive online adversary: the adversary knows the randomized online algorithm and has access to the decisions made by the randomized online algorithm in the previous time steps.

Example 1.2. *Ski rental problem*
As a toy example, let us present the SKI RENTAL *problem where one has two options for his winter holidays:*

- *pay* 1€ *per day to rent skis, or*
- *buy the skis for* 10€

The online principle behind this example is that the decision maker cannot predict the future conditions or complications he can get during a journey and thus does not know the number of days he will go for skiing. If the person is skiing less than 10 *days it would be beneficial to only rent the skis, otherwise it would be best to buy them. This algorithm is the optimal algorithm for this problem, in a deterministic setting, getting a competitive ratio equals to* 1.9 *in the worst case scenario.*
As we said previously, it is possible to get a better competitive ratio by using some randomized online algorithm. Indeed, using randomized techniques, one can get a $\frac{e}{e-1}$*-competitive algorithm. See Karlin et al. (1990) for a detailed presentation of this example.*

Let us now, to conclude this presentation of the online framework, talk about online learning algorithms. The study of these algorithms is quite recent and share some concepts with the multistage framework.
An online learning algorithm, given a set of feasible decisions and an adversary, makes an action iteratively inducing some loss/reward for each decision (made either by the algorithm or by the adversary). The goal of such an algorithm is to minimize the regrets represented by the difference between the cost of the decision made by the algorithm and the cost of the optimal static solution that does not evolve over the time horizon. The similarity with the multistage framework is within the objective functions of such problems, called the regret functions, where one seeks to minimize an accumulative cost step by step. Multiple problems have been studied in this framework, such as routing problems in Awerbuch and Kleinberg (2008), some min-max discrete problems in Bampis et al. (2019a), some online learning tree problems in Buchbinder et al. (2016), these authors were the first to study a variant of the multistage framework (see previous section) to cite a few of them.

1.3.3 Dynamic algorithms

Let us now present the studies on dynamic algorithms. We chose to present this framework after the online and reoptimization ones, the latter being a restrictive case of the more general dynamic setting.
Indeed, in a dynamic setting, that we will present on graph problems (a complete study of the dynamic graph problems are addressed in Eppstein et al. (1999)), one

is asked to give an algorithm able to maintain the quality of a solution as the graph evolves locally. The algorithm also has to be more efficient than computing a solution of the new instance from scratch. This local evolution of the graph can be anything, from insertions or deletions of nodes and edges to changes in the edges weights, that can all happen simultaneously. This contrasts with the reoptimization setting, which is a restrictive case of the dynamic setting and where only one local modification occurs between two time steps, and typically only two time steps are being considered.
The graph evolution in dynamic setting results in the *update* of its underlying data structure. Actually, in dynamic algorithms, one is interested in being able to *update* and maintain a given data structure on a new instance efficiently, i.e., faster than computing a new solution with a static algorithm. A *query*, using this data structure, will then be asked when the decision maker wants to solve the problem. Note that dynamic algorithms always start with trivial instances, i.e., cliques or independent sets for graph problems ... Less formally, one can say that, in the dynamic framework, when a solution is taken, the instance is modified and the decision maker has to adapt his solution.
The main difference between the online setting and the dynamic one is that in the dynamic setting, one can completely change its decision on the next time steps, which is not allowed in online algorithms, where, once a decision is taken, it can not be changed afterwards.
As a quick example of the *update* and *query* procedures, we can mention the MINIMUM SPANNING TREE problem. Indeed, we will use as an example an algorithm that solve this problem by sorting the list of elements of the instance. The *update* procedure will remove an element of the list or add the element at its sorted position, for deletions or insertions of elements in a new instance respectively, hence ensuring that the data structure is sorted list of elements. The *query* procedure will then only need to output a solution based on the sorted list with its underlying data structure, which is faster than sorting the list for each new instance and then finding a shortest path using this list.
While presenting the dynamic setting, we need to introduce briefly the notion of amortized complexity. Indeed, in the precedent frameworks and examples and in all the rest of the thesis, we present some complexity results that all consider the worst case scenario. In dynamic algorithms, the amortized complexity is often looked at, being the complexity one needs to compute a solution on average over all the different time steps. Indeed, in this framework, it is more relevant to analyze the complexity of the *update* and *query* procedures on average at each time step, rather than to look at the complexity of solving an instance every time from scratch.
The results one seeks to obtain with dynamic algorithms differ for polynomially solvable and **NP**-hard problems.

- For polynomially solvable problems, it is possible to find an optimal solution efficiently. Thus, in the dynamic setting literature, authors did not looked at improving the solution value, but instead presented faster ways of maintaining the optimal solution while the instances were subject to perturbations, improving the complexity. Note that here the complexity is studied in terms of the number of updates and the number of queries one needs to do.
  A lot of work has been done on tree and path problems, since the first results in the dynamic setting were presented in (Frederickson (1985)), looking at the MIN SPANNING TREE problem and in (Even and Gazit (1985)), addressing the SHORTEST PATH problem.

- For **NP**-hard problems, in a dynamic setting, one seeks an approximation algorithm giving the same ratio as the static one and such that the approximate solution is computed faster than if the static algorithm was used. Less results were found for those problems in a dynamic setting. This negative observation is due to the fact that we seek to obtain an approximation ratio from an approximate solution. Indeed, the data structure is guaranteeing a certain approximation ratio and thus the *update* procedure is applied to an approximate solution and not an optimal one, making it hard to maintain some properties in most of the cases.
  The VERTEX COVER problem was studied in Ivkovic and Lloyd (1993b) and the BIN PACKING problem in Ivkovic and Lloyd (1993a) to name two of them.

1.3.4 Temporal graphs

Let us now introduce the temporal graphs.
The concept of temporal graphs was formally introduced in Berman (1996) and then developed formally in Kempe et al. (2000) where the authors addressed some network problems and presented this new dynamic setting. In temporal graphs, one is given a discrete time horizon and a pair (G, λ) where G is a graph with a fixed vertex set over the time horizon and λ a function on the edge set of G, i.e. each edge $e \in G$ has a time label $\lambda(e)$ and is present only at the fixed time step $\lambda(e)$. A more general approach of temporal graphs deals with the same pair (G, λ) but allows edges to be present at different time steps, i.e. to have several time labels per edge (Mertzios et al. (2019)). Then, one is asked to solve a problem on this temporal graph that can model a lot of different types of real world situations such as network problems, path problems...

The main difference with the multistage framework is that here one needs to move on the graph over time in order to build a unique solution, and not a sequence of solutions as in the multistage framework.

On the other hand, it shares a global time property with the multistage frame-

work leading to the study of offline problems with instances subject to discrete changes over a given time horizon.

Most of the problems studied in this framework concern path problems and are presented in the survey (Michail (2016)).
However, some authors also addressed other types of graph optimization problems. In Himmel et al. (2017), a variant of the CLIQUE problem called $\Delta-$ CLIQUE was studied. In this problem, one is asked to find subsets of vertices connected for Δ consecutive time steps, echoing the multistage framework.
Very recently, in Akrida et al. (2020) two variants of the VERTEX PROBLEM were addressed. In the first one, one is asked, given a temporal graph, to find a sequence of subsets of vertices such that it covers the edges of the graph at least once during the time horizon, to be minimized (note that the set of edges changes between consecutive time steps). In the second one, the edges of the graph have to be covered once during a small time window and over the whole time horizon, assuring thereby a stability in the solution.

Example 1.3. *The* SHORTEST PATH *problem*
In the SHORTEST PATH *problem in temporal graphs, given a temporal graph such that at each time step of the time horizon $t = 1, \dots, T$ there is a set E_t of usable edges one needs to decide whether there exists a path from s to t, i.e. $v_{i1}, \dots, v_{iT}$, such that $v_{i1} = s$, $v_{iT} = t$ and $\forall t = 1, \dots, T-1$ either $v_{it} = v_{i(t+1)}$ (no movement) or $(v_{it}, v_{i(t+1)}) \in E_t$ (we use the edge present at the time step t).*

1.3.5 Stochastic Optimization

To conclude our presentation on different approaches dealing with evolving data over a time horizon we will now present the Stochastic Optimization theory. This approach addresses data uncertainty with the introduction of probabilities in the problem definition. Once again, it is not possible for the decision maker to have a complete knowledge of the problem instances.
The stochastic optimization community first focused on problems with a probability distribution on the problem parameters, e.g., the weights of edges for graph problems or the bound constraints for linear problems, but with a fixed structure, i.e., with a fixed set of nodes/edges for graph problems, fixed constraints for linear programming problems,... The study of problems with this approach began a long time ago. In Beardwood et al. (1959) the authors looked at the TSP problem when probabilities are applied to the node positions. Later, in Frieze (1985), the authors also addressed path problems under this presented configuration, looking at the SPANNING TREE problem with probabilities applied to the weights of the edges. Note that in the context of stochastic optimization there is also a literature dedicated to

some multistage problems. However, the multistage term denotes here a different model. Indeed, in Ravi and Sinha (2006), the authors addressed several optimization problems in the multistage stochastic setting and especially the two-stage stochastic model. In the two stage model, at a first stage, part of the data is known and a probability distribution is characterizing the uncertain future. Then, at a second stage, the data are revealed. One needs to optimize its first stage decisions in order to minimize both the cost of the first stage and the expected cost of the second stage.

It was in Jaillet (1985) that was addressed a new way of using probabilities in optimization problems. This time, the probabilities were not applied on parameters on fixed elements of the problem, but on the presence or absence of the elements themselves. Indeed, for example, a probability distribution was applied to the existence or not of nodes/edges for graph problems, of constraints for linear programming problems,etc.
Soon after, in the thesis of Bertsimas (1988), the notion of *a priori* optimization was presented (see also Bertsimas et al. (1990) for a detailed paper on *a priori* optimization).
Let us develop this technique on graph problems, and let us observe that it shares some properties with the reoptimization techniques presented before. When dealing with *a priori* optimization, one is given a graph with probabilities on its nodes and an *a priori* solution f on an initial instance of the graph. One is asked to find an updating method, that is able to update the a a priori solution to any possible feasible solution for all possible instances, (i.e., instances composed of all the combinations of presence/absence of nodes following the probability distribution), in such a way that the average cost on all instances is minimum (for minimization problems). This updating method has to be done in a reasonable computational time. In this case and in real world applications, it can be beneficial, for the average value to be minimum, and for the solutions to remain as stable as possible over all the feasible solutions, hence echoing the multistage framework.

Example 1.4. *We will illustrate stochastic optimization with an example of the* TSP *problem.*
A company needs to deliver some products in different cities and wishes to be as quick as possible. This problem can indeed be modeled as a TSP *problem. In a graph, where cities are represented by nodes and roads between two cities by weighted edges, the edges weights are the distances between the cities. One then needs to find a shortest path, going through all of the cities exactly once and getting back to its initial city.*
Let us suppose now that the company has some knowledge of the traffic and possible accidents occurring each day. It would associate probabilities to the values of the distances between two cities, i.e. probabilities on the values of edges of the graph.

The weight of the edges being a parameter of the graph, this problem would be solved using the first strategy presented in this stochastic optimization presentation.
Let us this time suppose that the company knows that, sometimes, some products will not need to be delivered in some cities. Thus, the company would associate this time probabilities to cities, corresponding to probabilities on the presence or not of nodes of the graph. To solve this variant, one needs to use the second presented stochastic strategy. Indeed, in order to be as efficient as possible, it is interesting for the company to develop an updating method from a priori solution and thus being sure to adapt efficiently its decisions and deliveries and to remain stable.

Now that we presented the multistage framework and some other approaches tackling evolving data, we will present in the next chapter the first study of the thesis focuses on the MULTISTAGE KNAPSACK problem in the offline setting, which are presented in the Section 1.2.1.

Chapter 2

Multistage Knapsack Problem

2.1 Introduction

We presented and motivated the study of the Multistage Knapsack Problem in section 1.2.1. Note that, in this chapter, the multistage framework is studied in the offline setting.

Briefly, let us recall the definition of the problem. We are given a time horizon $t = 1, 2, \ldots, T$, and a sequence of knapsack instances $I_1, I_2, \ldots, I_T$, one for each time step, defined on a set of n objects. In every time step t we have to choose a feasible knapsack S_t of I_t, which gives a *knapsack profit*. Taking into account transition costs, we measure the stability/similarity of two consecutive solutions S_t and S_{t+1} by identifying the objects for which the decision, to be picked or not, remains the same in S_t and S_{t+1}, giving a *transition profit*. We are asked to produce a sequence of solutions $S_1, S_2, \ldots, S_T$ so that the total knapsack profit plus the overall transition profit is maximized.

Our main contribution is a polynomial time approximation scheme ($PTAS$) for the multistage version of the Knapsack problem. As we said earlier, up to the best of our knowledge, this is the first approximation scheme for a multistage combinatorial optimization problem and its existence contrasts with the inapproximability results for other combinatorial optimization problems that are even polynomial-time solvable in the static case (e.g. the multistage Spanning Tree problem (Gupta et al. (2014)), or the multistage Bipartite Perfect Matching problem (Bampis et al. (2018a))).

An extended abstract of this chapter has been presented at MFCS (Mathematical Foundations of Computer Science) 2019 (Bampis et al. (2019c)).

2.1.1 Problem definition

Formally, the MULTISTAGE KNAPSACK problem can be defined as follows.

Definition 2.1. *In the* MULTISTAGE KNAPSACK *problem (MK) we are given:*

- *a time horizon* $T \in \mathbb{N}^*$*, a set* $N = \{1, 2, \ldots, n\}$ *of objects;*
- *For any* $t \in \{1, \ldots, T\}$*, any* $i \in N$*:*
  - $p_{ti} \geq 0$ *the profit of taking object* i *at time* t
  - $w_{ti} \geq 0$ *the weight of object* i *at time* t
- *For any* $t \in \{1, \ldots, T-1\}$*, any* $i \in N$*:*
  - $B_{ti} \in \mathbb{R}^+$ *the bonus of the object* i *if we keep the same decision for* i *at time* t *and* $t+1$*.*
- *For any* $t \in \{1, \ldots, T\}$*: the capacity* C_t *of the knapsack at time* t*.*

We are asked to select a subset $S_t \subseteq N$ *of objects at each time* t *so as to respect the capacity constraint:* $\sum_{i \in S_t} w_{ti} \leq C_t$*. To a solution* $S = (S_1, \ldots, S_T)$ *are associated:*

- *A knapsack profit* $\sum_{t=1}^{T} \sum_{i \in S_t} p_{ti} = \sum_{t=1}^{T} p_t(S_t)$ *corresponding to the sum of the profits of the* T *knapsacks;*
- *A transition profit* $\sum_{t=1}^{T-1} \sum_{i \in \Delta_t} B_{ti} = \sum_{t=1}^{T-1} b(S_t, S_{t+1})$ *where* Δ_t *is the set of objects either taken or not taken at both time steps* t *and* $t+1$ *in* S *(formally* $\Delta_t = (S_t \cap S_{t+1}) \cup (\overline{S_t} \cap \overline{S_{t+1}})$*).*

The value of the solution S *is the sum of the knapsack profit and the transition profit, to be maximized.*

2.1.2 Related works

Knapsack variants.

Our work builds upon the KNAPSACK literature (see Kellerer et al. (2004)). It is well-known that there is a simple 2-approximation algorithm as well as a fully polynomial-time approximation scheme (FPTAS) for the static case (see Ibarra and Kim (1975); Lawler (1979); Magazine and Oguz (1981); Kellerer and Pferschy (1999) for a detailed presentation of the KNAPSACK problem). There are two variants that are of special interest for our work:

(i) The first variant is a generalization of the KNAPSACK problem known as the k-DIMENSIONAL KNAPSACK ($k - DKP$) problem:

Definition 2.2. *In the k-dimensional* KNAPSACK *problem ($k - DKP$), we have a set $N = \{1, 2, \ldots, n\}$ of objects. Each object i has a profit p_i and k weights w_{ji}, $j = 1, \ldots, k$. We are also given k capacities C_j. The goal is to select a subset $Y \subseteq N$ of objects such that:*

- *The capacity constraints are respected: for any j, $\sum_{i \in Y} w_{ji} \leq C_j$;*
- *The profit $\sum_{i \in Y} p_i$ is maximized.*

It is well known that for the usual KNAPSACK problem, in the continuous relaxation (variables in $[0, 1]$), at most one variable is fractional. Caprara et al. (2000) showed that this can be generalized for $k - DKP$.

Let us consider the following ILP formulation ($ILP - DKP$) of the problem:

$$\begin{cases} \max \sum_{i \in N} p_i y_i & & & \\ s.t. \left| \begin{array}{lll} \sum_{i \in N} w_{ji} y_i & \leq & C_j \quad \forall j \in \{1, ..., k\} \\ y_i \in \{0, 1\} & & \forall i \in N \end{array} \right. & & & \end{cases}$$

Theorem 2.1. *(Caprara et al. (2000)) In the continuous relaxation ($LP - DKP$) of ($ILP - DKP$) where variables are in $[0, 1]$, in any basic solution at most k variables are fractional.*

Note that with an easy affine transformation on variables, the same result holds when variable y_i is subject to $a_i \leq y_i \leq b_i$ instead of $0 \leq y_i \leq 1$: *in any basic solution at most k variables y_i are such that $a_i < y_i < b_i$.*

Caprara et al. (2000) use the result of Theorem 2.1 to show that for any fixed constant k ($k - DKP$) admits a polynomial-time approximation scheme (PTAS). Other PTASes have been presented in Oguz and Magazine (1980); Frieze et al. (1984). Korte and Schrader Korte and Schrader (1981) showed that there is no FPTAS for $k - DKP$ unless **P=NP**.

(ii) The second related variant is a simplified version of ($k - DKP$) called CARDINALITY($2 - KP$), where the dimension is 2, all the profits are 1 and, given a K, we are asked if there is a solution of value at least K (decision problem). In other words, given two knapsack constraints, can we take K objects and verify the two constraints? The following result is shown in Kellerer et al. (2004).

Theorem 2.2. *(Kellerer et al. (2004)) CARDINALITY($2 - KP$) is **NP**-complete.*

2.1.3 Our contribution

As stated before, our main contribution is to propose a PTAS for the MULTISTAGE KNAPSACK problem. Furthermore, we prove that there is no FPTAS for the problem

even in the case where $T = 2$, unless **P=NP**. We also give a pseudopolynomial-time algorithm for the case where the number of steps is bounded by a fixed constant and we show that otherwise the problem remains **NP**-hard even in the case where all the weights, profits and capacities are 0 or 1. The following table summarizes our main result pointing out the impact of the number of time steps on the difficulty of the problem ("no $FPTAS$" means "no $FPTAS$ unless **P=NP**").

$T = 1$	T fixed	any T
pseudopolynomial	pseudopolynomial	strongly **NP**-hard
FPTAS	PTAS	PTAS
-	no FPTAS	no FPTAS

We point out that the negative results (strongly **NP**-hardness and no FPTAS) hold even in the case of *uniform bonus* when $B_{ti} = B$ for all $i \in N$ and all $t = 1, \ldots, T-1$.

2.2 ILP formulation

The MULTISTAGE KNAPSACK problem can be written as an ILP as follows. We define Tn binary variables x_{ti} equal to 1 if i is taken at time t ($i \in S_t$) and 0 otherwise. We also define $(T-1)n$ binary variables z_{ti} corresponding to the transition profit of object i between time t and $t+1$. The profit is 1 if i is taken at both time steps, or taken at none, and 0 otherwise. Hence, $z_{ti} = 1 - |x_{(t+1)i} - x_{ti}|$. Considering that we solve a maximization problem, this can be linearized by the two inequalities: $z_{ti} \leq -x_{(t+1)i} + x_{ti} + 1$ and $z_{ti} \leq x_{(t+1)i} - x_{ti} + 1$ and positive coefficients in the objective function. We end up with the following ILP (called $ILP - MK$):

$$
\begin{cases}
\max \displaystyle\sum_{t=1}^{T} \sum_{i \in N} p_{ti} x_{ti} + \sum_{t=1}^{T-1} \sum_{i \in N} z_{ti} B_{ti} & & & \\
s.t. \left| \begin{array}{lcll}
\displaystyle\sum_{i \in N} w_{ti} x_{ti} & \leq & C_t & \forall t \in \{1, ..., T\} \\
z_{ti} & \leq & -x_{(t+1)i} + x_{ti} + 1 & \forall t \in \{1, ..., T-1\}, \forall i \in N \\
z_{ti} & \leq & x_{(t+1)i} - x_{ti} + 1 & \forall t \in \{1, ..., T-1\}, \forall i \in N \\
x_{ti} \in \{0, 1\} & & & \forall t \in \{1, ..., T\}, \forall i \in N \\
z_{ti} \in \{0, 1\} & & & \forall t \in \{1, ..., T-1\}, \forall i \in N
\end{array} \right.
\end{cases}
$$

In devising the PTAS we will extensively use the linear relaxation $(LP - MK)$ of $(ILP - MK)$ where variables x_{ti} and z_{ti} are in $[0, 1]$.

2.3 A polynomial time approximation scheme

In this section we show that MULTISTAGE KNAPSACK admits a PTAS. The central part of the proof is to derive a PTAS when the number of steps is a fixed constant (Sections 2.3.1 and 2.3.2). The generalization to an arbitrary number of steps is done in Section 2.3.3.

Building upon Caprara et al. (2000), our PTAS for a fixed number of time steps heavily relies on a property of the relaxed LP-formulation of MULTISTAGE KNAPSACK: we show that there are at most T^3 fractional variables in an optimal (basic) solution of the (relaxed) MULTISTAGE KNAPSACK problem. Based on this bound, the PTAS is built from a combination of (1) bruteforce search (to find the most profitable objects), (2) a preprocessing step and (3) a rounding of the fractional solution of the (relaxed) LP-formulation. The preprocessing step associated to the bound on the number of fractional variables allow to bound the global loss of the solution built by the algorithm.

We show how to bound the number of fractional variables in Section 2.3.1. We first illustrate the reasoning on the case of two time-steps, and then present the general result. In Section 2.3.2 we present the PTAS for a constant number of steps. For ease of notation, we will sometimes write a feasible solution as $S = (S_1, \ldots, S_T)$ (subsets of objects taken at each time step), or as $S = (x, z)$ (values of variables in $(ILP - MK)$ or $(LP - MK)$).

2.3.1 Bounding the number of fractional objects in LP-MK

Warm-up: the case of two time-steps

We consider in this section the case of two time-steps ($T = 2$), and focus on the linear relaxation $(LP - MK)$ of $(ILP - MK)$ with the variables x_{ti} and z_i in $[0, 1]$ (we write z_i instead of z_{1i} for readability). We say that an object is *fractional* in a solution S if x_{1i}, x_{2i} or z_i is fractional.

Let us consider a (feasible) solution $\hat{S} = (\hat{x}, \hat{z})$ of $(LP - MK)$, where $\hat{z}_i = 1 - |\hat{x}_{2i} - \hat{x}_{1i}|$ (variables $\hat{z}_i$ are set to their optimal value w.r.t. $\hat{x}$).

We show the following.

Proposition 2.1. *If $\hat{S}$ is an optimal basic solution of $(LP-MK)$, at most 4 objects are fractional.*

Proof. First note that since we assume $\hat{z}_i = 1 - |\hat{x}_{1i} - \hat{x}_{2i}|$, if $\hat{x}_{1i}$ and $\hat{x}_{2i}$ are both integers then $\hat{z}_i$ is an integer. So if an object i is fractional either $\hat{x}_{1i}$ or $\hat{x}_{2i}$ is fractional.

Let us denote:

- L the set of objects i such that $\hat{x}_{1i} = \hat{x}_{2i}$.

- $P = N \setminus L$ the set of objects i such that $\hat{x}_{1i} \neq \hat{x}_{2i}$.

We first show Fact 1.

Fact 1. In P there is at most one object i with $\hat{x}_{1i}$ fractional.

Suppose that there are two such objects i and j. Note that since $0 < |\hat{x}_{1i} - \hat{x}_{2i}| < 1$, $\hat{z}_i$ is fractional, and so is $\hat{z}_j$. Then, for a sufficiently small $\epsilon > 0$, consider the solution S_1 obtained from $\hat{S}$ by transfering at time 1 an amount ϵ of weight from i to j (and adjusting consequently z_i and z_j). Namely, in S_1:

- $x^1_{1i} = \hat{x}_{1i} - \frac{\epsilon}{w_{1i}}$, $z^1_i = \hat{z}_i - d_i \frac{\epsilon}{w_{1i}}$, where $d_i = 1$ if $\hat{x}_{2i} > \hat{x}_{1i}$ and $d_i = -1$ if $\hat{x}_{2i} < \hat{x}_{1i}$ (since i is in P $\hat{x}_{2i} \neq \hat{x}_{1i}$).
- $x^1_{1j} = \hat{x}_{1j} + \frac{\epsilon}{w_{1j}}$, $z^1_j = \hat{z}_i + d_j \frac{\epsilon}{w_{1j}}$, where $d_j = 1$ if $\hat{x}_{2j} > \hat{x}_{1j}$ and $d_j = -1$ otherwise.

Note that (for ϵ sufficiently small) S_1 is feasible. Indeed (1) $\hat{x}_{1i}, \hat{x}_{1j}, \hat{z}_i$ and $\hat{z}_j$ are fractional (2) the weight of the knapsack at time 1 is the same in S_1 and in $\hat{S}$ (3) if $\hat{x}_{1i}$ increases by a small δ, if $\hat{x}_{2i} > \hat{x}_{1i}$ then $|\hat{x}_{2i} - \hat{x}_{i1}|$ decreases by δ so $\hat{z}_i$ can increase by δ (so $d_i = 1$), and if $\hat{x}_{2i} < \hat{x}_{i1}$ then $\hat{z}_i$ has to decrease by δ (so $d_i = -1$), and similarly for $\hat{x}_{1j}$.

Similarly, let us define S_2 obtained from $\hat{S}$ with the reverse transfer (from j to i). In S_2:

- $x^2_{1i} = \hat{x}_{1i} + \frac{\epsilon}{w_{1i}}$, $z^2_i = \hat{z}_i + d_i \frac{\epsilon}{w_{1i}}$
- $x^2_{1j} = \hat{x}_{1j} - \frac{\epsilon}{w_{1j}}$, $z^2_j = \hat{z}_i - d_j \frac{\epsilon}{w_{1j}}$

As previously, S_2 is feasible. Then $\hat{S}$ is clearly a convex combination of S_1 and S_2 (with coefficient 1/2), so not a basic solution, and Fact 1 is proven.

In other words (and this interpretation will be important in the general case), for this case we can focus on variables at time one, and interpret *locally* the problem as a (classical, unidimensional) fractional knapsack problem. By locally, we mean that if $\hat{x}_{1i} < \hat{x}_{2i}$ then x_{1i} must be in $[0, \hat{x}_{2i}]$ (in S^1, x^1_{1i} cannot be larger than $\hat{x}_{2i}$, otherwise the previous value of z^1_i would be erroneous); similarly if $\hat{x}_{1i} > \hat{x}_{2i}$ then x_{1i} must be in $[\hat{x}_{2i}, 1]$. The profit associated to object i is $p_{1i} + d_i B_{1i}$ (if x_{i1} increases/decreases by ϵ, then the knapsack profit increases/decreases by $p_{1i}\epsilon$, and the transition profit increases/decreases by $\epsilon d_i B_{1i}$, as explained above). Then we have at most one fractional variable, as in any fractional knapsack problem.

In P there is at most one object i with $\hat{x}_{1i}$ fractional. Similarly there is at most one object k with $\hat{x}_{2k}$ fractional. In P, for all but at most two objects, both $\hat{x}_{1i}$ and $\hat{x}_{2i}$, and thus $\hat{z}_i$, are integers.

Note that this argument would not hold for variables in L. Indeed if $\hat{x}_{1i} = \hat{x}_{2i}$, then $\hat{z}_i = 1$, and the transition profit decreases in *both* cases: when $\hat{x}_{1i}$ increases by $\delta > 0$ and when it decreases by δ. So, we cannot express $\hat{S}$ as a convex combination of S_1 and S_2 as previously.

However, let us consider the following linear program $2 - DKP$ obtained by fixing variables in P to their values in $\hat{S}$, computing the remaining capacities $C'_t = C_t - \sum_{j \in P} w_{tj}\hat{x}_{tj}$, and "imposing" $x_{1i} = x_{2i}$:

$$\begin{cases} \max \sum_{i \in L} (p_{1i} + p_{2i}) y_i + \sum_{i \in L} B_{1i} & \\ \sum_{i \in L} w_{1i} y_i & \leq \quad C'_1 \\ \sum_{i \in L} w_{2i} y_i & \leq \quad C'_2 \\ y_i \in [0,1] & \qquad \forall i \in L \end{cases}$$

Clearly, the restriction of $\hat{S}$ to variables in L is a solution of $2-DKP$. Formally, let $\hat{S}_L = (\hat{y}_j, j \in L)$ defined as $\hat{y}_j = \hat{x}_{1j}$. $\hat{S}_L$ is feasible for $2 - DKP$. Let us show that it is basic: suppose a contrario that $\hat{S}_L = \frac{S^1_L + S^2_L}{2}$, with $S^1_L = (y^1_i, i \in L) \neq S^2_L$ two feasible solutions of $2 - DKP$. Then consider the solution $S^1 = (x^1, y^1)$ of $(LP - MK)$ defined as:

- If $i \in L$ then $x^1_{1i} = x^1_{2i} = y^1_i$, and $z^1_{1i} = 1 = \hat{z}_{1i}$.
- Otherwise (for i in P) S^1 is the same as $\hat{S}$.

S^1 is clearly a feasible solution of Multistage Knapsack. If we do the same for S^2_L, we get a (different) feasible solution S^2, and $\hat{S} = \frac{S^1 + S^2}{2}$, so $\hat{S}$ is not basic, a contradiction.

By the result of Caprara et al. (2000), $\hat{S}_L$ has at most 2 fractional variables. Then, in L, for all but at most 2 variables both $\hat{x}_{1i}$, $\hat{x}_{2i}$ and $\hat{z}_i$ are integers. □

General case

The case of 2 time steps suggests to bound the number of fractional objects by considering 3 cases:

- Objects with $\hat{x}_{1i}$ fractional and $\hat{x}_{1i} \neq \hat{x}_{2i}$. As explained in the proof of Proposition 2.1, this can be seen locally (as long as x_{1i} does not reach $\hat{x}_{2i}$) as a knapsack problem from which we can conclude that there is at most 1 such fractional object.

- Similarly, objects with $\hat{x}_{2i}$ fractional and $\hat{x}_{1i} \neq \hat{x}_{2i}$.
- Objects with $\hat{x}_{1i} = \hat{x}_{2i}$ fractional. As explained in the proof of Proposition 2.1, this can be seen as a $2 - DKP$ from which we can conclude that there are at most 2 such fractional objects.

For larger T, we may have different situations. Suppose for instance that we have 5 time steps, and a solution (x, z) with an object i such that: $x_{1i} < x_{2i} = x_{3i} = x_{4i} < x_{5i}$. So we have x_{ti} fractional and constant for $t = 2, 3, 4$, and different from x_{1i} and x_{5i}. The idea is to say that we cannot have many objects like this (in a basic solution), by interpreting these objects on time steps $3, 4, 5$ as a basic optimal solution of a $3 - DKP$ (locally, i.e. with a variable y_i such that $x_{1i} \leq y_i \leq x_{5i}$).

Then, roughly speaking, the idea is to show that for any pair of time steps $t_0 \leq t_1$, we can bound the number of objects which are fractional and constant on this time interval $[t_0, t_1]$ (but not at time $t_0 - 1$ and $t_1 + 1$). Then a sum on all the possible choices of (t_0, t_1) gives the global upper bound.

Let us state this rough idea formally. We consider a (feasible) solution $\hat{S} = (\hat{x}, \hat{z})$ of $(LP - MK)$, where $\hat{z}_{ti} = 1 - |\hat{x}_{(t+1)i} - \hat{x}_{ti}|$ (variables $\hat{z}_{ti}$ are set to their optimal value w.r.t. $\hat{x}$).

In such a solution $\hat{S} = (\hat{x}, \hat{z})$, let us define as previously an object as *fractional* if at least one variable $\hat{x}_{ti}$ or $\hat{z}_{ti}$ is fractional. Our goal is to show the following result.

Theorem 2.3. *If $\hat{S} = (\hat{x}, \hat{z})$ is an optimal basic solution of $(LP - MK)$, it has at most T^3 fractional objects.*

Before proving the theorem, let us introduce some definitions and show some lemmas. Let t_0, t_1 be two time steps with $1 \leq t_0 \leq t_1 \leq T$.

Definition 2.3. *The set $F(t_0, t_1)$ associated with $\hat{S} = (\hat{x}, \hat{z})$ is the set of objects i (called fractional w.r.t. (t_0, t_1)) such that*

- $0 < \hat{x}_{t_0 i} = \hat{x}_{(t_0+1)i} = \cdots = \hat{x}_{t_1 i} < 1$;
- *Either $t_0 = 1$ or $\hat{x}_{(t_0-1)i} \neq \hat{x}_{t_0 i}$;*
- *Either $t_1 = T$ or $\hat{x}_{(t_1+1)i} \neq \hat{x}_{t_1 i}$;*

In other words, we have $\hat{x}_{ti}$ fractional and constant on $[t_0, t_1]$, and $[t_0, t_1]$ is maximal w.r.t. this property.

For $t_0 \leq t \leq t_1$, we note C'_t the remaining capacity of knapsack at time t considering that variables outside $F(t_0, t_1)$ are fixed (to their value in $\hat{x}$):

$$C'_t = C_t - \sum_{i \notin F(t_0,t_1)} w_{ti} \hat{x}_{ti}.$$

As previously, we will see $x_{t_0 i}, \ldots, x_{t_1 i}$ as a single variable y_i. We have to express the fact that this variable y_i cannot "cross" the values $\hat{x}_{(t_0-1)i}$ (if $t_0 > 1$) and $\hat{x}_{(t_1+1)i}$ (if $t_1 < T$), so that everything remains locally (in this range) linear. So we define the lower and upper bounds a_i, b_i induced by Definition 2.3 as:

- Initialize $a_i \leftarrow 0$. If $\hat{x}_{(t_0-1)i} < \hat{x}_{t_0 i}$ then do $a_i \leftarrow \hat{x}_{(t_0-1)i}$. If $\hat{x}_{(t_1+1)i} < \hat{x}_{t_1 i}$ then do $a_i \leftarrow \max(a_i, \hat{x}_{(t_1+1)i})$.
- Similarly, initialize $b_i \leftarrow 1$. If $\hat{x}_{(t_0-1)i} > \hat{x}_{t_0 i}$ then do $b_i \leftarrow \hat{x}_{(t_0-1)i}$. If $\hat{x}_{(t_1+1)i} > \hat{x}_{t_1 i}$ then do $b_i \leftarrow \min(b_i, \hat{x}_{(t_1+1)i})$.

Note that with this definition $a_i < \hat{x}_{t_0,i} =< \hat{x}_{t_1,i} < b_i$. This allows us to define the polyhedron $P(t_0, t_1)$ as the set of $y = (y_i : i \in F(t_0, t_1))$ such that

$$\begin{cases} \sum_{i \in F(t_0,t_1)} w_{ti} y_i \leq C'_t & \forall t \in \{t_0, ..., t_1\} \\ a_i \leq y_i \leq b_i & \forall i \in F(t_0, t_1) \end{cases}$$

Definition 2.4. *The solution associated with $\hat{S} = (\hat{x}, \hat{z})$ is $\hat{y}$ defined as $\hat{y}_i = \hat{x}_{t_0 i}$ for $i \in F(t_0, t_1)$.*

Lemma 2.1. *If $\hat{S} = (\hat{x}, \hat{z})$ is an basic solution, then the solution $\hat{y}$ associated with $(\hat{x}, \hat{z})$ is feasible of $P(t_0, t_1)$ and basic.*

Proof. Since $(\hat{x}, \hat{z})$ is feasible, $\hat{y}$ respects the capacity constraints (remaining capacity), and $a_i < \hat{y}_i = \hat{x}_{t_0 i} < b_i$ so $\hat{y}$ is feasible for $P(t_0, t_1)$.

Suppose now that $\hat{y} = \frac{y^1 + y^2}{2}$ for two feasible solutions $y^1 \neq y^2$ of $P(t_0, t_1)$. We associate to y^1 a feasible solution $S^1 = (x^1, z^1)$ for $(LP - MK)$ as follows.

We fix $x^1_{ti} = \hat{x}_{ti}$ for $t \notin [t_0, t_1]$, and $x^1_{ti} = y^1_i$ for $t \in [t_0, t_1]$. We fix variables z^1_{it} to their maximal values, i.e., $z^1_{ti} = 1 - |x^1_{(t+1)i} - x^1_{ti}|$. This way, we get a feasible solution (x^1, z^1) since y^1 and y^2 feasible for $P(t_0, t_1)$. Note that:

- $z^1_{ti} = \hat{z}_{ti}$ for $t \notin [t_0 - 1, t_1]$, since the corresponding variables x are the same in S^1 and $\hat{S}$;
- $z^1_{ti} = 1 = \hat{z}_{ti}$ for $t \in [t_0, t_1 - 1]$, since variables x are constant on the interval $[t_0, t_1]$.

Then, for variables z, the only modifications between z^1 and $\hat{z}$ concerns the "boundary" variables z^1_{ti} for $t = t_0 - 1$ and $t = t_1$.

We build this way two solutions $S^1 = (x^1, z^1)$ and $S^2 = (x^2, z^2)$ of $(LP - MK)$ corresponding to y^1 and y^2. By construction, S^1 and S^2 are feasible. They are also different provided that y^1 and y^2 are different. It remains to prove that $\hat{S}$ is the half sum of S^1 and S^2.

Let us first consider variables x:

- if $t \notin [t_0, t_1]$, $x^1_{ti} = x^2_{ti} = \hat{x}_{ti}$ so $\hat{x}_{ti} = \frac{x^1_{ti}+x^2_{ti}}{2}$.
- if $t \in [t_0, t_1]$, $x^1_{ti} = y^1_i$ and $x^2_{ti} = y^2_t$, so $\frac{x^1_{ti}+x^2_{ti}}{2} = \frac{y^1_i+y^2_i}{2} = \hat{y}_i = \hat{x}_{ti}$.

Now let us look at variables z: first, for $t \notin \{t_0 - 1, t_1\}$, $z^1_{ti} = z^2_{ti} = \hat{z}_{ti}$ so $\hat{z}_{ti} = \frac{z^1_{ti}+z^2_{ti}}{2}$. The last and main part concerns about the last 2 variables $z_{(t_0-1)i}$ (if $t_0 > 1$) and $z_{t_1 i}$ (if $t_1 < T$).

We have $z^1_{(t_0-1)i} = 1-|x^1_{t_0 i}-x^1_{(t_0-1)i}| = 1-|x^1_{t_0 i}-\hat{x}_{(t_0-1)i}|$ and $\hat{z}_{(t_0-1)i} = 1-|\hat{x}_{t_0 i}-\hat{x}_{(t_0-1)i}|$. The crucial point is to observe that thanks to the constraint $a_i \leq y_i \leq b_i$, and by definition of a_i and b_i, $x^1_{t_0,i}$, $x^2_{t_0,i}$ and $\hat{x}_{t_0,i}$ are either all greater than (or equal to) $\hat{x}_{(t_0-1)i}$, or all smaller than (or equal to) $\hat{x}_{(t_0-1)i}$.

Suppose first that they are all greater than (or equal to) $\hat{x}_{(t_0-1)i}$. Then:

$$z^1_{(t_0-1)i} - \hat{z}_{(t_0-1)i} = |\hat{x}_{t_0,i} - \hat{x}_{t_0-1,i}| - |x^1_{t_0,i} - \hat{x}_{t_0-1,i}| = \hat{x}_{t_0 i} - x^1_{t_0 i} = \hat{y}_i - y^1_i$$

Similarly, $z^2_{(t_0-1)i} - \hat{z}_{(t_0-1)i} = \hat{y}_i - y^2_i$. So

$$\frac{z^1_{(t_0-1)i} + z^2_{(t_0-1)i}}{2} = \frac{2\hat{z}_{(t_0-1)i} + 2\hat{y}_i - y^1_i - y^2_i}{2} = \hat{z}_{(t_0-1)i}.$$

Now suppose that they are all smaller than (or equal to) $\hat{x}_{t_0-1,i}$. Then:

$$z^1_{(t_0-1)i} - \hat{z}_{(t_0-1)i} = |\hat{x}_{t_0 i} - \hat{x}_{(t_0-1)i}| - |x^1_{t_0 i} - \hat{x}_{(t_0-1)i}| = x^1_{t_0 i} - \hat{x}_{t_0 i} = y^1_i - \hat{y}_i$$

Similarly, $z^2_{(t_0-1)i} - \hat{z}_{(t_0-1)i} = y^2_i - \hat{y}_i$. So

$$\frac{z^1_{(t_0-1)i} + z^2_{(t_0-1)i}}{2} = \frac{2\hat{z}_{(t_0-1)i} - 2\hat{y}_i + y^1_i + y^2_i}{2} = \hat{z}_{(t_0-1)i}.$$

Then, in both cases, $\hat{z}_{(t_0-1)i} = \frac{z^1_{(t_0-1)i}+z^2_{(t_0-1)i}}{2}$.

With the very same arguments we can show that $\frac{z^1_{t_1 i}+z^2_{t_1 i}}{2} = \hat{z}_{t_1 i}$.

Then, $\hat{S}$ is the half sum of S^1 and S^2, contradicting the fact that $\hat{S}$ is basic. □

Now we can bound the number of fractional objects w.r.t. (t_0, t_1).

Lemma 2.2. $|F(t_0, t_1)| \leq t_1 + 1 - t_0$.

Proof. $P(t_0, t_1)$ is a polyhedron corresponding to a linear relaxation of a $k - DLP$, with $k = t_1 + 1 - t_0$. Since $\hat{y}$ is basic, using Theorem 2.1 (and the note after) there are at most $k = t_1 + 1 - t_0$ variables $\hat{y}_i$ such that $a_i < \hat{y}_i < b_i$. But by definition of $F(t_0, t_1)$, for all $i \in F(t_0, t_1)$ $a_i < \hat{y}_i < b_i$. Then $|F(t_0, t_1)| \leq t_1 + 1 - t_0$. □

Now we can easily prove Theorem 2.3.

Proof. First note that if $\hat{x}_{ti}$ and $\hat{x}_{(t+1)i}$ are integral, then so is $\hat{z}_{ti}$. Then, if an object i is fractional at least one $\hat{x}_{ti}$ is fractional, and so i will appear in (at least) one set $F(t_0, t_1)$.

We consider all pairs (t_0, t_1) with $1 \leq t_0 \leq t_1 \leq T$. Thanks to Lemma 2.2, $|F(t_0, t_1)| \leq t_1 + 1 - t_0$. So, the total number of fractional objects is at most:

$$N_T = \sum_{t_0=1}^{T} \sum_{t_1=t_0}^{T} (t_1 + 1 - t_0) \leq T^3$$

Indeed, there are less than T^2 choices for (t_0, t_1) and at most T fractional objects for each choice. □

Note that with standard calculation we get $N_T = \frac{T^3+3T^2+2T}{6}$, so for $T = 2$ time steps $N_2 = 4$: we have at most 4 fractional objects, the same bound as in Proposition 2.1.

2.3.2 A PTAS for a constant number of time steps

Now we can describe the $PTAS$. Informally, the algorithm first guesses the ℓ objects with the maximum reward in an optimal solution (where ℓ is defined as a function of ϵ and T), and then finds a solution on the remaining instance using the relaxation of $(ILP - MK)$. The fact that the number of fractional objects is small allows to bound the error made by the algorithm.

For a solution S (either fractional or integral) we define $g_i(S)$ as the reward of object i in solution S: $g_i(S) = \sum_{t=1}^{T} p_{ti}x_{ti} + \sum_{t=1}^{T-1} z_{ti}B_{ti}$. The value of a solution S is $g(S) = \sum_{i \in N} g_i(S)$.

Consider the algorithm A^{LP} which, on an instance of $(ILP - MK)$ of Multi-stage Knapsack:

- Finds an optimal (basic) solution $S^r = (x^r, z^r)$ of the continuous relaxation $(LP - MK)$ of $(ILP - MK)$;
- Takes at step t an object i if and only if $x^r_{ti} = 1$.

Clearly, A^{LP} outputs a feasible solution, say $S^{A^{LP}}$, the value of which verifies:

$$g(S^{A^{LP}}) \geq g(S^r) - \sum_{i \in F} g_i(S^r) \tag{2.1}$$

where F is the set of fractional objects in S^r. Indeed, for each integral (i.e., not fractional) object the reward is the same in both solutions.

Now we can describe the algorithm $PTAS_{ConstantMK}$, which takes as input an instance of Multistage Knapsack and an $\epsilon > 0$.

Algorithm $PTAS_{ConstantMK}$

1. Let $\ell := \min\left\{\left\lceil \frac{(T+1)T^3}{\epsilon} \right\rceil, n\right\}$.
2. For all $X \subseteq N$ such that $|X| = \ell$, $\forall X_1 \subseteq X, ..., \forall X_T \subseteq X$:
   If for all $t = 1, \ldots, T$ $w_t(X_t) = \sum_{j \in X_t} w_{tj} \leq C_t$, then:
   - Compute the rewards of object $i \in X$ in the solution $(X_1, \ldots, X_T)$, and find the smallest one, say k, with reward g_k.
   - On the subinstance of objects $Y = N \setminus X$:
     - For all $i \in Y$, for all $t \in \{1, \ldots, T\}$: if $p_{ti} > g_k$ then set $x_{ti} = 0$.
     - Apply A^{LP} on the subinstance of objects Y, with the remaining capacity $C'_t = C_t - w_t(X_t)$ for each t, where some variables x_{ti} are set to 0 as explained in the previous step.
   - Let $(Y_1, ..., Y_T)$ be the sets of objects taken at time $1, \ldots, T$ by A^{LP}. Consider the solution $(X_1 \cup Y_1, ..., X_T \cup Y_T)$.
3. Output the best solution computed.

Theorem 2.4. *The algorithm $PTAS_{ConstantMK}$ is a $(1-\epsilon)$-approximation algorithm running in time $O\left(2^{O(T^5/\epsilon)} n^{O(T^4/\epsilon)}\right)$ for* Multistage Knapsack.

Proof. First, there are $O(n^\ell)$ choices for X; for each X there are 2^ℓ choices for each X_t, so in all there are $O(n^\ell 2^{\ell T})$ choices for $(X_1, \ldots, X_T)$. For each choice $(X_1, \ldots, X_t)$, we compute the reward of elements, and then apply A^{LP}. Since $\ell \leq \lceil T^3(T+1)/\epsilon \rceil$, the running time follows.

Now let us show the claimed approximation ratio. Consider an optimal solution S^*, and suppose wlog that $g_i(S^*)$ are in non increasing order. Consider the iteration of the algorithm where $X = \{1, 2, \ldots, \ell\}$. At this iteration, consider the choice $(X_1, \ldots, X_T)$ where X_t is exactly the subset of objects in X taken by S^* at time t (for $t = 1, \ldots, T$). The solution S computed by the algorithm at this iteration (with X and $(X_1, \ldots, X_T)$) is $S = (X_1 \cup Y_1, \ldots, X_T \cup Y_T)$ where $(Y_1, \ldots, Y_T)$ is the solution output by A^{LP} on the subinstance of objects $Y = N \setminus X$, where x_{ti} is set to 0 if $p_{ti} > g_k$.

Note that we consider $\ell = \left\lceil \frac{(T+1)T^3}{\epsilon} \right\rceil$, otherwise we would get an optimal solution for the problem.

Note that since we consider the iteration where X corresponds to the ℓ objects of largest reward in the optimal solution, we do know that for an object $i > \ell$, if $p_{ti} > g_k$ then the optimal solution does not take object i at time t (it would have a reward greater than g_k), so we can safely fix this variable to 0. The idea behind putting these variables x_{ti} to 0 is to prevent the relaxed solution to take fractional objects with very large profits. These objects could indeed induce a very high loss when rounding the fractional solution to an integral one as done by A^{LP}.
By doing this, the number of fractional objects (i.e., objects in F) does not increase. Indeed, if we put a variable x_{ti} at 0, it is not fractional so nothing changes in the proof of Theorem 2.3.

The value of S^* is $g(S^*) = \sum_{i\in X} g_i(S^*) + \sum_{i\in Y} g_i(S^*)$. Thus, by Equation 2.1, we have:

$$\begin{aligned} g(S) &\geq \sum_{i\in X} g_i(S^*) + \sum_{i\in Y} g_i(S^r) - \sum_{i\in F} g_i(S^r) \\ &\geq \sum_{i\in X} g_i(S^*) + \sum_{i\in Y} g_i(S^*) - \sum_{i\in F} g_i(S^r), \end{aligned}$$

where F is the set of fractional objects in the optimal fractional solution of the relaxation of the LP on Y, where x_{ti} is set to 0 if $p_{ti} > g_k$.

Each object of F has a profit at most g_k at any time steps and a transition profit at most $\sum_{t=1}^{T-1} B_{ti}$, so

$$\forall i \in F \; g_i(S^r) \leq T g_k + \sum_{t=1}^{T-1} B_{ti}$$

Now, note that in an optimal solution each object has a reward at least $\sum_{t=1}^{T-1} B_{ti}$ (otherwise simply never take this object), so for all $i \in F$ $g_k \geq g_i \geq \sum_{t=1}^{T-1} B_{ti}$. So we have:

$$\forall i \in F \; g_i(S^r) \leq (T+1) g_k$$

Since $g_i(S^*)$ are in non increasing order, we have $g_k = g_\ell(S^*) \leq \frac{\sum_{i\in X} g_i(S^*)}{\ell}$. So for all objects i in F, $g_i(S^r) \leq \frac{(T+1)\sum_{i\in X} g_i(S^*)}{\ell}$. By Theorem 2.3, there are at most T^3 of them, thus:

$$\begin{aligned} g(S) &\geq \sum_{i\in X} g_i(S^*) + \sum_{i\in Y} g_i(S^*) - \sum_{i\in F} g_i(S^r) \\ &\geq \sum_{i\in X} g_i(S^*) + \sum_{i\in Y} g_i(S^*) - \frac{(T+1)T^3}{\ell} \sum_{i\in X} g_i(S^*) \\ &\geq (1-\epsilon) \sum_{i\in X} g_i(S^*) + \sum_{i\in Y} g_i(S^*) \geq (1-\epsilon) g(S^*) \end{aligned}$$

□

By Theorem 2.4, for any fixed number of time steps T, MULTISTAGE KNAPSACK admits a PTAS.

2.3.3 Generalization to an arbitrary number of time steps

We now devise a PTAS for the general problem, for an arbitrary (not constant) number of steps. We actually show how to get such a PTAS provided that we have a PTAS for (any) constant number of time steps. Let A_{ϵ,T_0} be an algorithm which, given an instance of MULTISTAGE KNAPSACK with at most T_0 time steps, outputs a $(1-\epsilon)$-approximate solution in time $O(n^{f(\epsilon,T_0)})$ for some function f.

The underlying idea is to compute (nearly) optimal solutions on subinstances of bounded sizes, and then to combine them in such a way that at most a small fraction of the optimal value is lost.

Let us first give a rough idea of our algorithm $PTAS_{MK}$.

Given an $\epsilon > 0$, let $\epsilon' = \epsilon/2$ and $T_0 = \lceil \frac{1}{\epsilon'} \rceil$. We construct a set of solutions $S^1, \ldots, S^{T_0}$ in the following way:

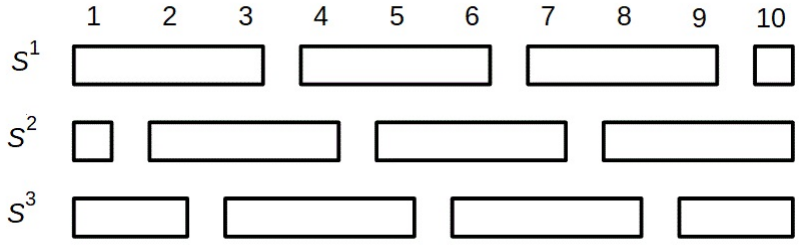
In order to construct S^1, we partition the time horizon $1, \ldots, T$ into $\lceil \frac{T}{T_0} \rceil$ consecutive intervals. Every such interval has length T_0, except possibly the last interval that may have a smaller length. We apply A_{ϵ,T_0} at every interval in this partition. S^1 is then just the concatenation of the partial solutions computed for each interval.

The partition on which is based the construction of the solution S^i, $1 < i \leq T_0$, is made in a similar way. The only difference is that the first interval of the partition of the time horizon $1, \ldots, T$ goes from time 1 to time $i-1$. For the remaining part of the time horizon, i.e. for $i, \ldots T$, the partition is made as previously, i.e. starting at time step i, every interval will have a length of T_0, except possibly the last one, whose length may be smaller. Once the partition is built, we apply A_{ϵ,T_0} to every interval of the partition. S^i, $1 < i \leq T_0$, is then defined as the concatenation of the partial solutions computed on each interval. Among the T_0 solutions $S^1, \ldots, S^{T_0}$, the algorithm chooses the best solution.

The construction is illustrated on Figure 2.1, with 10 time steps and $T_0 = 3$. The first solution S^1 is built by applying 4 times A_{ϵ,T_0}, on the subinstances corresponding to time steps $\{1,2,3\}, \{4,5,6\}, \{7,8,9\}$, and $\{10\}$. The solution S^2 is built by applying 4 times A_{ϵ,T_0}, on the subinstances corresponding to time steps $\{1\}, \{2,3,4\}, \{5,6,7\}$, and $\{8,9,10\}$.

More formally, given an $\epsilon > 0$, the algorithm $PTAS_{MK}$ works as follows.

- Let $\epsilon' = \epsilon/2$ and $T_0 = \left\lceil \frac{1}{\epsilon'} \right\rceil$. Let $\mathcal{I}_t = [t, \ldots, t+T_0-1] \cap [1, \ldots, T]$ be the set of (at most) T_0 consecutive time steps starting at t. We consider $\mathcal{I}_t$ for any t for which it is non empty (so $t \in [-T_0+2, \ldots, T]$).



2.1: The three solutions for $T_0 = 3$ and $T = 10$.

- For $t \in \{1, \ldots, T_0\}$:
  - Apply A_{ϵ',T_0} on all intervals $\mathcal{I}_{t'}$ with $t' \equiv t \pmod{T_0}$. Note that each time step belongs to exactly one of such intervals.
  - Define the solution S^t built from the partial solutions given by the applications of A_{ϵ',T_0}.
- Choose the best solution S among the T_0 solutions $S^1, \ldots, S^{T_0}$.

Theorem 2.5. *The algorithm $PTAS_{MK}$ is a polynomial-time approximation scheme for* Multistage knapsack.

Proof. The algorithm calls the A_{ϵ',T_0} algorithm $\lceil T/T_0 \rceil$ times for each of the T_0 generated solutions. Yet, the running time of A_{ϵ',T_0} is $n^{f(\frac{1}{\epsilon'},T_0)} = n^{f(\frac{2}{\epsilon},\lceil\frac{2}{\epsilon}\rceil)}$, i.e. a polynomial time for any fixed ϵ. So, the running time of the algorithm for any T is $T_0 \times \left\lceil \frac{T}{T_0} \right\rceil n^{f(\frac{2}{\epsilon},\frac{2}{\epsilon})} = O(Tn^{f(\frac{2}{\epsilon},\frac{2}{\epsilon})})$, a polynomial time for any fixed ϵ.

Each solution S^t of the T_0 generated solutions may loose some bonus between the last time step of one of its intervals $\mathcal{I}_{t+kT_0}$ and the first time step of its next interval $\mathcal{I}_{t+(k+1)T_0}$ (in Figure 2.1 for instance, S^1 misses the bonuses between steps 3 and 4, 6 and 7, and 9 and 10). Let $loss(S^t)$ be this loss with respect to some optimal solution S^*. Since we apply A_{ϵ',T_0} to build solutions, we get that the value $g(S^t)$ of S^t is such that:

$$g(S^t) \geq (1 - \epsilon')g(S^*) - loss(S^t)$$

Since the output solution S is the best among the solutions S^t, by summing up the previous inequality for $t = 1, \ldots, T_0$ and dividing by T_0, we get:

$$g(S) \geq (1 - \epsilon')g(S^*) - \frac{\sum_{t=1}^{T} loss(S^t)}{T_0}$$

Now, by construction the bonus between steps j and $j+1$ appears in the loss of exactly one S^t (see Figure 2.1). So the total loss of the T_0 solutions is the global transition bonus of S^*, so at most $g(S^*)$. Hence:

$$\begin{aligned} g(S) &\geq (1-\epsilon')g(S^*) - \frac{g(S^*)}{T_0} \\ &\geq (1-2\epsilon')g(S^*) = (1-\epsilon)g(S^*) \end{aligned}$$

□

2.4 Pseudo-polynomiality and hardness results

We complement the previous result on approximation scheme by showing the following results for MULTISTAGE KNAPSACK:

- First, it does not admit an FPTAS (unless **P=NP**), even if there are only two time steps (Section 2.4.1) and the bonus is uniform ($B_{ti} = B$ for all i, all t);
- Second, the problem is pseudo-polynomial if the number of time steps T is a fixed constant (Section 2.4.2) but is strongly **NP**-hard in the general case even in the case of uniform bonus (Section 2.4.3).

2.4.1 No FPTAS

Theorem 2.6. *There is no FPTAS for* MULTISTAGE KNAPSACK *unless* ***P=NP****, even if there are only two time steps and the bonus is uniform.*

Proof. We prove the result by a reduction from CARDINALITY$(2-KP)$, known to be **NP**-complete (Theorem 2.2)

For an instance I of CARDINALITY$(2-KP)$, we consider the following instance I' of MULTISTAGE KNAPSACK:

- There are $T = 2$ time steps, and the same set of objects $N = \{1, 2, ..., n\}$ as in I.
- The weights w_{1i} and w_{2i} are the same as in I, for all $i \in N$.
- $p_{1i} = p_{2i} = 1$ for all $i \in N$.
- $B_{1i} = 2$ for all $i \in N$.

We show that the answer to I is Yes (we can find K objects fulfilling the 2 knapsack constraints) if and only if there is a solution of value at least $2K + 2n$ for the instance I' of MK.

If we have a set A of K objects for I, then we simply take these objects at both time steps in I'. This is feasible, the knapsack revenue is $2K$ and the transition revenue is $2n$.

Conversely, consider a solution of value at least $2K+2n$ in I'. In such a solution, we necessarily have $x_{1j} = x_{2j}$, i.e., an object is taken at the time 1 if and only if it is taken at the time 2. Indeed, assume that there is one $i \in N$ such that $x_{1i} \neq x_{2i}$, then consider the solution where $x_{1i} = x_{2i} = 0$. This is still feasible; the knapsack revenue reduces by 1 but the transition revenue increases by 2, a contradiction. Thus the same set of objects A is taken at both time steps. Since the value of the solution is at least $2K + 2n$, the size of A is at least K. Hence the answer to I is Yes.

Note that in I' the optimal value is at most $4n$. We produce in this reduction polynomially bounded instances of Multistage Knapsack (with only two time steps), so the problem does not admit an FPTAS. Indeed, suppose that there is an FPTAS producing a $(1-\epsilon)$-approximation in time $p(1/\epsilon, n)$, for some polynomial p. Let $\epsilon = \frac{1}{4n+1}$. If we apply the FPTAS with this value of ϵ on I' we get a solution S of value at least $(1-\epsilon)OPT(I') \geq OPT(I') - \frac{OPT(I')}{4n+1} > OPT(I') - 1$. Yet all the possible values are integers so S is optimal. The running time is polynomial in n, impossible unless **P=NP**. □

2.4.2 Pseudo-polynomiality for a constant number of time steps

We show here that the pseudo-polynomiality of the Knapsack problem generalizes to Multistage Knapsack when the number of time steps is constant. More precisely, with a standard dynamic programming procedure, we have the following.

Theorem 2.7. Multistage Knapsack *is solvable in time* $O(T(2C_{max} + 2)^T n)$ *where* $C_{max} = \max\{C_i, i = 1, \ldots, T\}$.

Proof. For any T-uple $(c_1, \ldots, c_T)$ where $0 \leq c_i \leq C_i$, and any $s \in \{0, \ldots, n\}$, we define $\alpha(c_1, \ldots, c_T, s)$ to be the best value of a solution $S = (S_1, \ldots, S_T)$ such that:

- The weight of knapsack at time t is at most c_t: for any t, $\sum_{i \in S_t} w_{ti} \leq c_i$;
- The solution uses only objects among the first s: for any t, $S_t \subseteq \{1, \ldots, s\}$.

The optimal value of Multistage Knapsack is then $\alpha(C_1, \ldots, C_T, n)$. We compute α by increasing values of s. For $s = 0$, we cannot take any object so $\alpha(c_1, \ldots, c_T, 0) = 0$.

Take now $s \geq 1$. To compute $\alpha(c_1, \ldots, c_T, s)$, we simply consider all the 2^T possibilities for taking or not object s in the T time steps. Let $A \subseteq \{1, \ldots, T\}$ be a subset of time steps. If we take object s at time steps in A (and only there), we first check if A is a *valid* choice, i.e., $w_{ts} \leq c_t$ for any $t \in A$; then we can compute in $O(T)$ the corresponding reward $r_s(A)$ ($\sum_{t \in A} p_{ts}$ plus the transition bonus). We have:

$$\alpha(c_1, \ldots, c_T, s) =$$
$$\max\{r_s(A) + \alpha(c'_1 - w_{1s}, \ldots, c'_T - w_{Ts}, s-1) : A \subseteq \{1, \ldots, T\} \text{ valid}\}$$

with:

$$c'_i = c_i - w_{is} \text{ if } i \in A$$
$$c'_i = c - i \text{ otherwise}$$

The running time to compute one value of α is $O(T2^T)$. There are $O(n\Pi_{t=1}^{T}(C_i + 1)) = O(n(C_{max} + 1)^T)$ values to compute, so the running time follows. A standard backward procedure allows to recovering the solution. □

2.4.3 Strongly NP-hardness

Definition 2.5. Binary Multistage Knapsack *is the sub-problem of the* Multistage Knapsack *where all the weights, profits are all equal to* 0 *or* 1 *and capacities are equal to* 1.

For the usual Knapsack problem, the binary case corresponds to a trivial problem. For the multistage case, we have the following:

Theorem 2.8. Binary Multistage Knapsack *is* ***NP****-hard, even in the case of uniform bonus.*

Proof. We prove the result by a reduction from the Independent Set problem where, given a graph G and an integer K, we are asked if there exists a subset of K pairwise non adjacent vertices (called an independent set). This problem is well known to be **NP**-hard, see Garey and Johnson (1979).

Let (G, K) be an instance of the Independent Set problem, with $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. We build the following instance I' of Binary Multistage Knapsack:

- There are n objects $\{1, 2 \ldots, n\}$, one object per vertex;
- There are $T = m$ time steps: each edge (v_i, v_j) in E corresponds to one time step;

- At the time step corresponding to edge (v_i, v_j): objects i and j have weight 1, the others have weight 0, all objects have profit 1, and the capacity constraint is 1.

- The transition bonus is $b_{ti} = B = 2nm$ for all i, t.

We claim that there is an independent set of size (at least) K if and only if there is a solution for Binary Multistage Knapsack of value (at least) $n(m-1)B+mK$.

Suppose first that there is an independent set V' of size at least K. We take the K objects corresponding to V' at all time steps. This is feasible since we cannot take 2 objects corresponding to one edge. The built solution sequence has knapsack profit mK and transition profit $n(m-1)B$ (no modification).

Conversely, take a solution of Binary Multistage Knapsack of value at least $n(m-1)B + mK$. Since $B = 2nm$, there must be no modification of the knapsack over the time. Indeed, otherwise, the transition profit would be at most $n(m-1)B - B$, while the knapsack profit is at most mn, so the value would be less than $n(m-1)B$. So we consider the set of vertices corresponding to this knapsack. Thanks to the value of the knapsack, it has size at least K. Thanks to the capacity constraints of the knapsacks, this is an independent set. □

Since B is polynomially bounded in the proof, this shows that Multistage Knapsack is strongly **NP**-hard.

2.5 Conclusion

We considered the Multistage Knapsack problem in the offline setting and we studied the impact of the number of time steps in the complexity of the problem. In the next chapter, we present the study in the online setting of the large family of subset maximization problems in the multistage framework, the multistage knapsack problem being part of this family, and measure the importance of the knowledge of the future on the quality of solutions.

Chapter 3

Online Multistage Subset Maximisation Problems

3.1 Introduction

In this chapter, we focus on the online case, where at time t no knowledge is available for instances at times $t+1, \ldots, T$. When it is not possible to handle the online case, we turn our attention to the *k-lookahead* case, where at time t the instances at times $t+1, \ldots, t+k$ are also known. This case is of interest since in some applications like in dynamic capacity planning in data centers, the forecasts of future demands may be very helpful (see Lin et al. (2012); Liu et al. (2014) for examples of application). Our goal is to measure the impact of the lack of knowledge of the future on the quality and the stability of the returned solutions. Indeed, our algorithms are limited in their knowledge of the sequence of instances. Knowing that the number of time steps is given, we compute the competitive ratio of the algorithm after time step T: since we focus on maximization problems, we say that a (deterministic) algorithm is α-competitive (with competitive ratio α) if its value is at least $\frac{1}{\alpha}$ times the optimal value on all instances.

As it is usual in the online setting, we consider no limitations in the computational resources available. This implies that at every time step t, where instance I_t is known, we assume the existence of an oracle able to compute the optimal solution for that time step. Notice also that our lower bounds do not rely on any complexity assumptions. Some recent results are already known for the online multistage model (Bampis et al. (2018b); Gupta et al. (2014)), however all these results are obtained for specific problems. In this chapter, we study multistage variants of a broad family of maximization problems. An extended abstract of this chapter has been presented at ESA (European Symposium on Algorithms) 2019 (Bampis et al. (2019b)).

3.1.1 Problem definition

The family of optimization problems that we consider is the following.

Definition 3.1. (Subset Maximization Problems.) *A Subset Maximization problem $\mathcal{P}$ is a combinatorial optimization problem whose instances $I = (N, p, \mathcal{F})$ each consists of*

- *A ground set N;*
- *A set $\mathcal{F} \subseteq 2^N$ of feasible solutions such that $\emptyset \in \mathcal{F}$;*
- *A non-negative weight $p(S)$ for every $S \in \mathcal{F}$.*

The goal is to find $S^* \in \mathcal{F}$ such that $p(S^*) = \max\{p(S) : S \in \mathcal{F}\}$.

We will consider that the empty set is always feasible, ensuring that the feasible set of solutions is non empty. This is a very general class of problems, including the maximization *Subset Selection* problems studied by Pruhs and Woeginger (2007) (they only considered linear objective functions). It contains for instance graph problems where N is the set of vertices (as in any maximization induced subgraph problem verifying some property) or the set of edges (as in matching problems). It also contains classical set problems (knapsack, maximum 3-dimensional matching,...), and more generally 0-1 linear programs (with non negative profits in the objective function).

Given a problem in the previous class, we are interested in its multistage version (Gupta et al. (2014); Eisenstat et al. (2014)). The stability over time of a solution sequence is classically captured by considering a transition cost when a modification is made in the solution. Here, dealing with maximization problems as in the previous chapter, we will consider again a transition *bonus* B for taking into account the similarity of two consecutive solutions. In what follows, we will use the term object to denote an element of N (so an object can be a vertex of a graph, or an edge,..., depending on the underlying problem).

Definition 3.2. (Multistage Subset Maximization Problems.) *In a Multistage Subset Maximization problem $\mathcal{P}$, we are given*

- *a number of steps $T \in \mathbb{N}$, a set N of n objects;*
- *for any $t \in T$, an instance I_t of the optimization problem. We will denote:*
  - *p_t the objective (profit) function at time t*
  - *$\mathcal{F}_t \in 2^N$ the set of feasible solutions at time t*
- *$B \in \mathbb{R}^+$ a given transition profit.*

- *the value of a solution sequence* $\mathcal{S} = (S_1, \ldots, S_T)$ *is*

$$f(\mathcal{S}) = \sum_{t=1}^{T} p_t(S_t) + \sum_{t=1}^{T-1} b(S_t, S_{t+1})$$

  where $b(S_t, S_{t+1})$ *is the transition bonus for the solution between time steps* t *and* $t+1$ *(a definition of the bonus is given in Definition 3.3). We will use the term profit for* $p_t(S_t)$*, bonus for the transition bonus* $b(S_t, S_{t+1})$*, and value of a solution* $\mathcal{S}$ *for* $f(\mathcal{S})$*;*

- *the goal is to determine a solution sequence of maximum value.*

The fact that T is known may be regarded as rather uncommon the field of online algorithms. At the end of Subsection 3.1.2, we relate it to our results and justify it.

There are two natural ways to define the transition bonus. We will see that these two ways of measuring the stability induce some differences in the competitive ratios one can get.

Definition 3.3. (Types of transition bonus.) *If* S_t *and* S_{t+1} *denote, respectively, the solutions for time steps* t *and* $t+1$*, then we can define the transition bonus as:*

- Intersection Bonus: B *times* $|S_t \cap S_{t+1}|$*: in this case the bonus is proportional to the number of objects in the solution at time* t *that remain in it at time* $t+1$*.*

- Hamming Bonus: B *times* $|S_t \cap S_{t+1}| + |\overline{S_t} \cap \overline{S_{t+1}}|$*. Here we get the bonus for each object for which the decision (to be in the solution or not) is the same between time steps* t *and* $t+1$*. In other words, the bonus is proportional to* $|N|$ *minus the number of modifications (Hamming distance) in the solutions. The* MULTISTAGE KNAPSACK *problem was studied considering* Hamming Bonus *in Chapter 2.*

Note that by scaling profits (dividing them by B), we can arbitrarily fix $B = 1$. So from now on, we assume $B = 1$. Note that this was also the case with the uniform bonus presented in Chapter 2.

In this chapter, we will consider two possible ways for the data to evolve.

Definition 3.4. (Types of data evolution.)

- Static Set of Feasible Solutions (SSFS): *only profits may change over time, so the structure of feasible solutions remains the same (*$\mathcal{F}_t = \mathcal{F}$ *for all* t*).*

- General Evolution (GE): *any modification (but the set of objects) in the input sequence is possible. Both the profits and the set of feasible solutions may change over time. In this latter model, for knapsack, profits and weights of object (and the capacity of the bag) may change over time; for maximum independent set, edges in the graph may change,etc.*

3.1.2 Summary of Results and Overview

The contribution of this chapter is a framework for online multistage maximization problems (comprising different models) and almost tight upper and lower bounds on the best-possible competitive ratio for these models. The focus here is on deterministic algorithms.

We increase the complexity of the considered models over the course of the chapter. We start with the arguably simplest model: Considering a static set of feasible solutions clearly restricts the general model of evolution; while such a straightforward comparison between the Hamming and intersection bonus is not possible, the Hamming bonus seems simpler in that, compared to the intersection model, there are (somewhat comparable) extra terms added on the profit of both the algorithm and the optimum. As we show in Subsection 3.2.1, there is indeed a simple 2-competitive algorithm: At each time t, it greedily chooses the set S_t that either maximizes the transition bonus w.r.t. S_{t-1} (that is, choosing $S_t = S_{t-1}$, which is possible in this model) or maximizes the value $p_t(S_t)$. We complement this observation with a matching lower bound only involving two time steps.

We then toggle the transition-bonus model and the data-evolution model separately and show that constant competitive ratios can still be achieved. First, in Subsection 3.2.2, we consider intersection bonus. We show that, *after* modifying the profits (internally) to make larger solutions more profitable, a $(2 + 1/(T - 1))$-competitive algorithm can be achieved by a greedy approach again. We also give an (almost matching) lower bound of 2 again. Next, we toggle the evolution model. In Subsection 3.3.1, we adapt the greedy algorithm from Subsection 3.2.1 by reweighting to obtain a $(3 + 1/(T - 1))$-competitive algorithm using a more complicated analysis. We complement this result with a lower bound of $1 + \sqrt{2}$.

In Subsection 3.3.2, we finally consider the general-evolution model with intersection bonus, for which we give a simple lower bound showing that a constant-competitive ratio is not achievable. This lower bound relies on forbidding to choose any item in the second step that the algorithm chose in the first step. We circumnavigate such issues by allowing the algorithm a lookahead of one step and present a 4-competitive algorithm for that setting. A similar phase transition has been observed for a related problem (Bampis et al. (2018b)), but our algorithm, based on a doubling approach, is different. We also give a matching lower bound of 4 on the competitive ratio of any algorithm in the same setting. We summarize all results described so far in Table 3.1.

We note that the lower bounds mentioned for the Hamming model are only shown for a specific fixed number of time steps, and that in general there is no trivial way of extending these bounds to a larger number of time steps. One may however argue that the large-T regime is in fact the interesting one for both practical applications and in theory, the latter because the effect of having a first time step without bonus

	static set of feasible solutions	general evolution
Hamming bonus	$2 - o(1) \leq c^\star \leq 2$ Theorems 3.2 and 3.1	$1 + \sqrt{2} \leq c^\star \leq 3 + o(1)$ Theorems 3.8 and 3.6
Intersection bonus	$2 \leq c^\star \leq 2 + o(1)$ Theorems 3.5 and 3.4	$c^\star = \infty$ $c^\star = 4$ for 1-lookahead Theorems 3.10, 3.12, and 3.11

Table 3.1: Our bounds on the best-possible competitive ratio $c^\star$ for the different models. The Landau symbol is with respect to $T \to \infty$.

vanishes. At the end of the respective sections, we therefore give asymptotical lower bounds of 3/2 and roughly 1.696 for the cases of a static set of feasible solutions and general evolutions, respectively. These bounds are non-trivial, but we do not know if they are tight.

It is plausible that the aforementioned upper bounds can be improved if extra assumptions on characteristics of the objective function and the sets of feasible solutions are made. In Subsection 3.3.1, we show that already very natural assumptions suffice: Assuming that at each time the feasible solutions are closed under taking subsets and the objective function is subadditive, we give a $(21/8+o(1))$-competitive algorithm for the model with a general evolution and Hamming bonus, improving the previous $(3 + o(1))$-competitive ratio. Our lower bounds for general evolution and Hamming bonus in fact fulfill the extra assumptions.

We observe that all our algorithms except for the one discussed in Subsection 3.2.1 require that T is known in that their behavior in the last step is different from the behavior in the steps before. This assumption is crucial: In all these models, there are examples in which one can in the first time step choose either a small profit or a potentially large bonus not knowing if there is another timestep to realize the bonus. Such examples imply a superconstant lower bound on the competitive ratio in these models. This justifies our assumption that T is known.

In Section 3.4, we summarize our results and mention directions for future research that we consider interesting.

3.2 Model of a Static Set of Feasible Solutions

We consider here the model of evolution where only profits change over time: $\mathcal{F}_t = \mathcal{F}$ for any t. We first consider the Hamming bonus model and show a simple 2-competitive algorithm. We will then show that a (asymptotic) competitive ratio

of 2 can also be achieved in the intersection bonus model using a more involved algorithm. In both cases, this ratio 2 is shown to be (asymptotically) optimal.

3.2.1 Hamming-Bonus Model

Theorem 3.1. *In the SSFS model with Hamming bonus, there is a 2-competitive algorithm.*

Proof. We consider the very simple following algorithm. At each time step t, the algorithm computes an optimal solution S_t^* with associated profit $p_t(S_t^*)$. At $t = 1$ we fix $S_1 = S_1^*$. For $t > 1$, if $p_t(S_t^*) > n$ then fix $S_t = S_t^*$, otherwise fix $S_t = S_{t-1}^*$ (which is possible thanks to the fact that the set of feasible solutions does not change).

Let f^* be the optimal value. Since any solution sequence gets profit at most $p_t(S_t^*)$ at time t, and bonus at most n between two consecutive time steps, we get $f^* \leq \sum_{t=1}^{T} p(S_t^*) + n(T-1)$.

By construction, at time $t > 1$, either the algorithm gets profit $p_t(S_t^*)$ when $p_t(S_t^*) > n$, or bonus (from $t-1$) n when $n \geq p_t(S_t^*)$. So in any case the algorithm gets profit plus bonus at least $\frac{p_t(S_t^*)+n}{2}$. At time 1 it gets profit exactly $p_1(S_1^*)$. So

$$f(S_1 \ldots, S_T) \geq p_1(S_1^*) + \sum_{t=2}^{T} \frac{p_t(S_t^*)}{2} + \frac{n(T-1)}{2} \geq \frac{f^*}{2}$$

which completes the proof. □

Theorem 3.2. *Consider the SSFS model with Hamming bonus. For any $\epsilon > 0$, there is no $(2-\epsilon)$-competitive algorithm, even if there are only 2 time steps.*

Proof. We consider a set $N = \{1, 2, \ldots, n\}$ of $n = 1 + \lceil \frac{1}{\epsilon} \rceil$ objects, $T = 2$ time steps, and an additive profit function. There are three feasible solutions: $S^0 = \emptyset$, $S^1 = \{1\}$ and $S^2 = \{2, \ldots, n\}$. At $t = 1$, all the profits are 0. Let us consider an online algorithm A. We consider the three possibilities for the algorithm at time 1:

- At time 1, A chooses S^0: at time 2 we give profit 1 to all objects. If A takes no object at time 2, it gets profit 0 and bonus n. If it takes S^1, it gets profit 1 and bonus $n-1$. If it takes S^2, it gets profit $n-1$ and bonus 1, so in any case the computed solution has value n. The solution consisting of taking S^2 at both time steps has profit $n-1$ and bonus n, so value $2n-1$.

- At time 1, A chooses S^1: at time 2 we give profit 0 to object 1, and profit 1 to all other objects. Then, if the algorithm takes S^0 (resp, S^1, S^2), at time 2 its gets value $n-1$ (resp, n, $n-1$) while the solution consisting of taking S^2 at both time steps has value $2n-1$.

- At time 1, A chooses S^2: at time 2 we give profit n to object 1, and 0 to all other objects. Then if the algorithm takes S^0 (resp, S^1, S^2) at time 2 its gets value 1 (resp, n, n), while the solution consisting of taking S^1 at both time steps has value $2n$.

In any case, the ratio is at least $\frac{2n-1}{n} = 2 - \frac{1}{n} > 2 - \epsilon$. □

We complement this lower bound with an asymptotical result for large T.

Theorem 3.3. *Consider the SSFS model with Hamming bonus. For every $\epsilon > 0$, there is a T sufficiently large such that there is no $(3/2 - \epsilon)$-competitive algorithm.*

Proof. Let $N := \{1, 2\}$. The static set of feasible solutions is $\mathcal{F} = \{\emptyset, \{1\}, \{2\}\}$. Initially, $p_1(\{1\}) = 0$ and $p_1(\{2\}) = 1$. As long as the algorithm has not picked item 2 until some time t, we set $p_{t+1}(\{1\}) = 0$ and $p_{t+1}(\{2\}) = 1$ again. Note that, in order to be $(3/2-\epsilon)$-competitive, the algorithm however has to pick item 2 eventually. Further, the ratio between the profit of the optimum and the algorithm during this part is $3/2 - o(1)$ as the length of this part approaches ∞.

The remaining time horizon is partitioned into contiguous *phases*. Consider a phase that starts at time t. The invariant at the beginning of the phase is that both the algorithm and the optimum have picked the same item in the previous time step $t - 1$. Let this item be w.l.o.g. item 2; the other case is symmetric. Then $p_t(\{2\}) = 1$ and $p_t(\{1\}) = 3$. By the same reasoning as above, we can assume the algorithm chooses an item at t. Let $i \in \{1, 2\}$ be that item. Then $p_{t+1}(\{i\}) = 0$ and $p_{t+1}(\{3-i\}) = 1$. As long as the algorithm is still not picking item $3-i$ during the time interval $[t + 1, t']$, $p_{t'+1}(\{i\}) = 0$ and $p_{t'+1}(\{3 - i\}) = 1$. Once the algorithm picks item $3-i$ at some time, the phase ends *regularly*; otherwise it ends *by default*.

Now consider a phase of length ℓ that ends regularly (note $\ell \geq 2$). We claim that the values of the algorithm and the optimum have a ratio of at least 3/2. This is because of the following estimates on the algorithm's and optimum's value:

- In either case for i, the algorithm obtains a value of 3 in time step t. Furthermore, the total bonus in all subsequent time steps is $(\ell - 2) \cdot 2$, because the algorithm has to switch from item i to item $3-i$ at time $t+l-1$. There is an additional profit of 1 at time $t+\ell-1$. Therefore, the total value is $4+(\ell-2)\cdot 2$

- The value of the optimum is at least $6+(\ell-2)\cdot 3$: It chooses item $3-i$ already at time t and keeps it until time $t + \ell - 1$, obtaining a value of 3 in that time step and another 3 in each subsequent time step.

This proves the claim and thereby the theorem as a phase that ends by default can be extended to one that ends regularly by modifying the optimum's and algorithm's values by constants. □

3.2.2 Intersection Bonus Model

In the Intersection Bonus model things get harder since an optimal solution S_t^* may be of small size and then gives very small (potential) bonus for the next step. As a matter of fact, the algorithm of the previous section has unbounded competitive ratio in this case: take a large number n of objects, $\mathcal{F} = 2^N$, and at time 1 all objects have profit 0 up to one which has profit ϵ. The algorithm will take this object (instead of taking $n-1$ objects of profit 0) and then potentially get bonus at most 1 instead of $n-1$.

Thus we shall put an incentive for the algorithm to take solutions of large size, in order to have a chance to get a large bonus. We define the following algorithm called `MP-Algo` (for Modified Profit algorithm). Informally, at each time step t, the algorithm computes an optimal solution with a modified objective function p'_t. These modifications take into account (1) the objects taken at time $t-1$ and (2) an incentive to take a lot of objects. Formally, `MP-Algo` works as follows:

1. At $t = 1$: let $p'_1(S) = p_1(S) + |S|$ for each solution $S \in \mathcal{F}$. Choose S_1 as an optimal solution for the problem with modified profits p'_1.
2. For t from 2 to $T-1$: let $p'_t(S) = p_t(S) + |S \cap S_{t-1}| + |S|$. Choose S_t as an optimal solution for the problem with modified profit function p'_t.
3. At $t = T$: let $p'_T(S) = p_T(S) + |S \cap S_{T-1}|$. Choose S_T as an optimal solution with modified profit function p'_T.

The cases $t = 1$ and $t = T$ are specific since there is no previous solution for $t = 1$, and no future solution for $t = T$.

Theorem 3.4. *In the SSFS model with intersection bonus, `MP-Algo` is* $\left(\frac{2}{1-1/(T-1)}\right)$*-competitive.*

Proof. Let $(\hat{S}_1, \ldots, \hat{S}_T)$ be an optimal sequence. Since S_t is optimal with respect to p'_t, for $t = 2, \ldots, T-1$ we have:

$$p'_t(S_t) = p_t(S_t) + |S_t \cap S_{t-1}| + |S_t| \geq p'_t(\hat{S}_t) \geq p_t(\hat{S}_t) + |\hat{S}_t|. \tag{3.1}$$

Since S_{t-1} is also a feasible solution at time t, we have:

$$p'_t(S_t) = p_t(S_t) + |S_t \cap S_{t-1}| + |S_t| \geq 2|S_{t-1}|. \tag{3.2}$$

Similarly, at $t = T$ $p'_T(S) = p_T(S) + |S \cap S_{t-1}|$ so

$$p_T(S_T) + |S_T \cap S_{T-1}| \geq p_T(\hat{S}_T) and \tag{3.3}$$
$$p_T(S_T) + |S_T \cap S_{T-1}| \geq |S_{T-1}|. \tag{3.4}$$

At $t = 1$, $p'_t(S) = p_t(S) + |S|$, so

$$p_1(S_1) + |S_1| \geq p_1(\hat{S}_1) + |\hat{S}_1|. \tag{3.5}$$

Now, note that $|S_t \cap S_{t-1}|$ is the transition bonus of the computed solution between $t-1$ and t. By summing Equation (3.1) for $t = 2, \ldots, T-1$, Equation (3.3) and Equation (3.5), we deduce:

$$f(S_1, \ldots, S_T) + \sum_{t=1}^{T-1} |S_t| \geq \sum_{t=1}^{T} p_t(\hat{S}_t) + \sum_{t=1}^{T-1} |\hat{S}_t|.$$

Since in the optimal sequence the transition bonus between time t and $t+1$ is at most $|\hat{S}_t|$, we get:

$$f(S_1, \ldots, S_T) + \sum_{t=1}^{T-1} |S_t| \geq f(\hat{S}_1, \ldots, \hat{S}_T). \tag{3.6}$$

Now we sum Equation (3.2) for $t = 2, \ldots, T-1$ and Equation (3.4):

$$f(S_2, \ldots, S_T) + \sum_{t=2}^{T-1} |S_t| \geq 2 \sum_{t=2}^{T-1} |S_{t-1}| + |S_{T-1}| = \sum_{t=1}^{T-2} |S_t| + \sum_{t=1}^{T-1} |S_t|.$$

From this we easily derive:

$$f(S_1, \ldots, S_T) \geq \sum_{t=2}^{T-2} |S_t|. \tag{3.7}$$

By summing Equations (3.6) and (3.7) we have $2f(S_1, \ldots, S_T) \geq f(\hat{S}_1, \ldots, \hat{S}_T) - |S_{T-1}|$. The competitive ratio follows from the fact that $f(\hat{S}_1, \ldots, \hat{S}_T) \geq (T-1)|S_{T-1}|$ (since S_{T-1} is feasible for all time steps). □

We note that competitive ratio 2 can be derived with a similar analysis when the number of time steps is 2 or 3. Let us now show a matching lower bound (which is also valid in the asymptotic setting).

Theorem 3.5. *Consider the SSFS model with intersection bonus. For any $\epsilon > 0$ and number of time steps $T = \lceil 1/\epsilon \rceil$, there is no $(2-\epsilon)$-competitive algorithm.*

Proof. Let $\epsilon > 0$ and $T = \left\lceil \frac{1}{\epsilon} \right\rceil$. We consider T time steps, and a set N of $n = T$ objects. The objective function is linear, and feasible solutions are sets of at most 1 object. At $t = 1$, the profit of each object is 1. Then, at each time step, if the algorithm takes an object, this object will have profit 0 until the end. While an object is not taken by the algorithm, its profit remains 1.

Since the algorithm takes at most one object at each time step, there is an object which is never taken till the last step. The solution of taking this object during all the process has value $2T-1$. But at each time step the algorithm either takes a new object (and gets no bonus) or keeps the previously taken object and gets no profit. So the value of the computed solution is at most T. The ratio is $2-\frac{1}{T} \geq 2-\epsilon$. □

3.3 Model of General Evolution

We consider in this section that the set of feasible solutions may evolve over time. We will show that in the Hamming bonus model, we can still get constant competitive ratios, though ratios slightly worse than in the case where only profits could change over time. Then, we will tackle the intersection bonus model, showing that no constant competitive ratio can be achieved. However, with only 1-lookahead we can get a constant competitive ratio.

3.3.1 Hamming Bonus Model

In this section we consider the Hamming bonus model. We first show in Section 3.3.1 that there exists a $\left(3+\frac{1}{T-1}\right)$-competitive algorithm. Interestingly, we then show in Section 3.3.1 that a slight assumption on the problem structure allows to improve the competitive ratio. More precisely, we achieve a 21/8 (asymptotic) competitive ratio if we assume that the objective function is subadditive (so including the additive case) and that a subset of a feasible solution is feasible. These assumptions are satisfied by all the problems mentioned in introduction. We finally consider lower bounds in Section 3.3.1.

General Case

We adapt the idea of the 2-competitive algorithm working for the Hamming bonus model for a static set of feasible solutions (Section 3.2.1) to the current setting where the set of feasible solutions may change. Let us consider the following algorithm `BestOrNothing`: at each time step t, `BestOrNothing` computes an optimal solution S_t^* with associated profit $p_t(S_t^*)$ and compares it to 2 times the maximum potential bonus, i.e. to $2n$. It chooses S_t^* if the associated profit is at least $2n$, otherwise it chooses $S_t = \emptyset$. A slight modification is applied for the last step T.

Formally, `BestOrNothing` works as follows:

1. For t from 1 to $T-1$:
   (a) Compute an optimal solution S_t^* at time t with associated profit $p_t(S_t^*)$
   (b) If $p_t(S_t^*) \geq 2n$ set $S_t = S_t^*$, otherwise set $S_t = \emptyset$.

2. At time T:
   (a) if $S_{T-1} = S^*_{T-1}$, then $S_T = S^*_T$.
   (b) Otherwise: if $p_t(S^*_t) \geq n$ set $S_T = S^*_T$, otherwise set $S_T = \emptyset$.

We show an upper bound on the competitive ratio achieved by this algorithm.

Theorem 3.6. *In the GE model with Hamming bonus,* `BestOrNothing` *is* $\left(3 + \frac{1}{T-1}\right)$*-competitive.*

Proof. Let us define $J \subseteq \{1, \ldots, T-1\}$ as the set of time steps $t < T$ where $p_t(S^*_t) \geq 2n$.

If $J \neq \emptyset$, let t_1 be the largest element in J. We first upper bound the loss of the algorithm up to time t_1. We will then deal with the time period from $t+1$ up to T.

The global profit of an optimal solution up to time t_1 is at most $2n(t_1 - |J|) + \sum_{t\in J} p_t(S^*_t)$. Its bonus (including the one from time t_1 to $t_1 + 1$) is at most nt_1. So its global value is at most $n\,(3t_1 - 2|J|) + \sum_{t\in J} p_t(S^*_t)$.

The solution computed by `BestOrNothing` gets profit at least $\sum_{t\in J} p_t(S^*_t)$. Note that it chooses the empty set always but $|J|$ times, so it gets transition bonus n at least $t_1 - 2|J|$ times (each step in J may prevent to get the bonus only between $t-1$ and t, and between t and $t+1$). So the global value of the computed solution up to time t_1 is at least $n \max\{0; t_1 - 2|J|\} + \sum_{t\in J} p_t(S^*_t)$.

Up to time t_1, the ratio r between the optimal value and the value of the solution computed by `BestOrNothing` verifies

$$r \leq \frac{n\,(3t_1 - 2|J|) + \sum_{t\in J} p_t(S^*_t)}{n \max\{0; t_1 - 2|J|\} + \sum_{t\in J} p_t(S^*_t)} \leq \frac{3t_1}{\max\{0; t_1 - 2|J|\} + 2|J|},$$

where we used the fact that $\sum_{t\in J} p_t(S^*_t) \geq 2n|J|$, $r \geq 1$ and decreasing in $\sum_{t\in J} p_t(S^*_t)$. Since $\max\{0; t_1 - 2|J|\} + 2|J| \geq t_1$ the ratio is at most 3 up to time t_1.

Now, let us consider the end of the process, from time $t_1 + 1$ (or 1 if J is empty) up to time T. If $t_1 = T - 1$ then we take the best solution at time T and get no extra loss, so the algorithm is 3-competitive in this case.

Now assume $t_1 < T - 1$. We know that `BestOrNothing` chooses the empty set up to $T - 1$. Let us first assume that $p_T(S^*_T) < n$. Then on the subperiod from $t_1 + 1$ to T `BestOrNothing` gets value $n(T - t_1 - 1)$ (bonuses), while the optimum gets bonus at most $n(T - t_1 - 1)$ and profit at most $2n(T - t_1 - 1) + n$. The optimal value is then at most $n\,(3T - 3t_1 - 2) = 3n(T - t_1 - 1) + n$.

Now suppose that $p_T(S^*_T) \geq n$ and $t_1 < T - 1$. On the subperiod from $t_1 + 1$ to T `BestOrNothing` gets value $n(T - t_1 - 2) + p_T(S^*_T)$, while the optimum gets bonus at most $n(T - t_1 - 1)$ and profit at most $2n(T - t_1 - 2) + p_T(S^*_T)$. The worst case

ratio occurs when $p_T(S_T^*) = n \leq \frac{3n(T-t_1-1)+p_T(S_T^*)}{n(T-t_1-2)+p_T(S_T^*)}$ that is decreasing in $p_T(S_T^*)$. In this case, as before, the value of the computed solution is $n(T-t_1-1)$, while the optimal value is at most $n\,(3T-3t_1-1) = 3n(T-t_1-1)+n$.

Then, in all cases we have that the optimal value is at most $3f(S_1,\ldots,S_n)+n$. But $f(S_1,\ldots,S_n) \geq (T-1)n$ (the computed solution has value at least $t_1 n$ up to t_1, and then at least $n(T-t_1-1)$), and the claimed ratio follows. □

Improvement for Sub-additivity and Subset Feasibility

In this section we assume that the problem have the following two properties:

- *subset feasibility*: at any time step, every subset of a feasible solution is feasible.
- *sub-additivity*: for any disjoint S, S', any t, $p_t(S \cup S') \leq p_t(S) + p_t(S')$.

Note that this implies that, if a feasible set X is partitioned into (disjoint) subsets $X_1,\ldots,X_h$, then $X_1,\ldots,X_h$ are feasible and $p_t(X) \leq \sum_i p_t(X_i)$.

We exploit this property to devise algorithms where we partition the set of objects and solve the problems on subinstances. As a first idea, let us partition the set of objects into into 3 sets A, B, C of size (roughly) $n/3$; consider the algorithm which at every time step t computes the best solutions S_t^A, S_t^B, S_t^C on each subinstance on A, B and C, and chooses S_t as the one of maximum profit between these 3 solutions. By sub-additivity and subset feasibility, the algorithm gets profit at least $1/3$ of the optimal profit at each time step. Dealing with bonuses, at each time step the algorithm chooses a solution included either in A, or in B, or in C so, for any $t < T$, at least one set among A, B and C is not chosen neither at time t nor at time $t+1$, and the algorithm gets transition bonus at least $n/3$. Hence, the algorithm is 3-competitive.

We now improve the previous algorithm. The basic idea is to remark that if for two consecutive time steps $t, t+1$ the solution S_t and S_{t+1} are taken in the same subset, say A, then the bonus is (at least) $2n/3$ instead of $n/3$. Roughly speaking, we can hope for a ratio better than $1/3$ for the bonus. Then the algorithm makes a trade-off at every time step: if the profit is very high then it will take a solution maximizing the profit, otherwise it will do (nearly) the same as previously. More formally, let us consider the algorithm `3-Part`. We first assume that n is a multiple of 3. A parameter $x \in \mathbb{R}_+$ will be defined later.

1. Partition N into three subsets A, B, C of size $n/3$.
2. For $t \in \{1,\ldots,T\}$: compute a solution S_t^* maximizing $p_t(S)$
   - Case (1): If $p_t(S_t^*) \geq xn$: define $S_t = S_t^*$

- Otherwise ($p_t(S_t^*) \leq xn$): compute solutions with optimal profit S_t^A, S_t^B, S_t^C included in A, B and C. Let a_t, b_t and c_t be the respective profits.
  - Case (2): if $t \geq 2$ and Case (1) did not occur at $t-1$, do:
    If $S_{t-1} \subseteq A$ (resp. $S_{t-1} \subseteq B$, $S_{t-1} \subseteq C$), compute $\max\{a_t+2n/3, b_t+n/3, c_t+n/3\}$ (resp. $\max\{a_t+n/3, b_t+2n/3, c_t+n/3\}$, $\max\{a_t+n/3, b_t+n/3, c_t+2n/3\}$) and define S_t as S_t^A, S_t^B or S_t^C accordingly.
  - Case (3) ($t=1$ or Case (1) occurred at $t-1$) do:
    * Define S_t as the solution with maximum profit among S_t^A, S_t^B, S_t^C.

If N is not a multiple of 3, we add one or two dummy objects that are in no feasible solutions (at any step). We prove an upper bound on the competitive ratio of this algorithm.

Theorem 3.7. *Consider the GE model with Hamming bonus. Under the assumption of subset feasibility and sub-additivity,* `3-Part` *is* $(21/8+O(1/T+1/n))$*-competitive.*

Proof. We mainly show that in each case (1), (2) or (3) the computed solution achieves the claimed ratio.

- Let us first consider a time step $t \geq 2$ where Case (2) occurs. It means that Case (2) or (3) occurred at the previous step, so S_{t-1} is included in A, B or C. Suppose w.l.o.g that algorithm took $S_{t-1} \subseteq A$. Then S_t^A gives a bonus at least $2n/3$ (between $t-1$ and t), and S_t^B and S_t^C gives a bonus at least $n/3$. By computing $\max\{a_t+2n/3, b_t+n/3, c_t+n/3\}$, we derive:

$$p_t(S_t) + b(S_t, S_{t-1}) \geq \frac{a_t+2n/3+b_t+n/3+c_t+n/3}{3} \geq \frac{p_t(S_t^*)}{3} + \frac{4n}{9},$$

  where S_t^* is a solution maximizing the profit at time t, using the fact that $p_t(S_t^*) \leq a_t+b_t+c_t$ by subset feasibility and submodularity. Since in Case (2) $p_t(S_t^*) \leq xn$, we derive:

$$p_t(S_t) + b(S_t, S_{t-1}) \geq r_2 \left(p_t(S_t^*) + n\right) \tag{3.8}$$

  with $r_2 = \frac{3x+4}{9(1+x)}$.

- Now, consider a time step $t \geq 2$ where Case (3) occurs. Then necessarily Case (1) occurs at step $t-1$. So $S_{t-1} = S_{t-1}^*$. Also, S_t has profit at least $p_t(S_t^*)/3$. So

$$\sum_{\ell=t-1}^{t} p_\ell(S_\ell) + b(S_\ell, S_{\ell-1}) \geq p_{t-1}(S_{t-1}^*) + \frac{p_t(S_t^*)}{3}$$

So $\sum_{\ell=t-1}^{t} p_\ell(S_\ell) + b(S_\ell, S_{\ell-1}) \geq r_3 \left(p_{t-1}(S^*_{t-1}) + n + p_t(S^*_t) + n\right)$ with

$$r_3 = \frac{p_{t-1}(S^*_{t-1}) + \frac{p_t(S^*_t)}{3}}{p_{t-1}(S^*_{t-1}) + 2n + p_t(S^*_t)}.$$

Since $p_{t-1}(S^*_{t-1}) \geq xn$, we get:

$$r_3 \geq \frac{xn + \frac{p_t(S^*_t)}{3}}{(2+x)n + p_t(S^*_t)} = r_4.$$

where we use the fact that $r_3 \leq 1$ and is increasing in $p_{t-1}(S^*_{t-1})$.
Since $p_t(S^*_t) \leq xn$, provided that we choose $x \geq 1$ such that $x/(2+x) \geq 1/3$, we get:

$$r_3 \geq \frac{xn + xn/3}{(2+x)n + xn} = \frac{2x}{3(1+x)}.$$

where we use the fact that $r_4 \geq \frac{1}{3}$ and is decreasing in $p_t(S^*_t)$

- Finally, suppose that Case (1) occurs at some step $t \geq 2$. Then $S_t = S^*_t$ and $p(S^*_t) \geq xn$, so

$$p_t(S_t) + b(S_t, S_{t-1}) \geq p_t(S_t) = p_t(S^*_t) \geq r_1(p_t(S^*_t) + n).$$

with $r_1 = \frac{x}{1+x}$.

By setting $x = \frac{4}{3}$, we get $r_1 \geq r_2 = 8/21$ and $r_1 \geq r_3 \geq \frac{8}{21}$.

It remains to look at step 1. If $p_1(S^*_1) \geq xn$ (Case (1)), then $S_1 = S^*_1$, so there is no profit loss. Otherwise, $p_1(S^*_1) \leq xn$, Case (3) occurs and the loss it at most $2p_1(S^*_1)/3 \leq 2xn/3 \leq n$. Since the optimal value is at least $n(T-1)$, the loss it a fraction at most $1/(T-1)$ of the optimal value.

If n is not a multiple of 3, adding one or two dummy objects add $T-1$ or $2(T-1)$ to solution values, inducing a loss which is a fraction at most $O(1/n)$ of the optimal value. □

Lower Bounds

We complement the algorithmic results with a lower bound for two time steps and an asymptotical one. Interestingly, these bounds are also valid for the latter restricted setting with subset feasibility and sub-additivity.

Theorem 3.8. *Consider the GE model with Hamming bonus and $T = 2$ time steps. For any $\epsilon > 0$, there is no $(1 + \sqrt{2} - \epsilon)$-competitive algorithm.*

Proof. We consider a knapsack problem with n objects ($n = 2$ suffices to show the result, but the proof is valid for any number n of objects), and $T = 2$ time steps. At time 1, all objects have weight 1 and profit $\alpha = \sqrt{2} - 1$; the capacity of the bag is n.

Let S_1 be the set of objects chosen at Step 1 by the algorithm (possibly $S_1 = \emptyset$). At $t = 2$ the algorithm receives the instance $I_2(S_1)$ where:

- the capacity is $c_2 = n - |S_1|$.
- each object not in S_1 receives weight and profit 1.
- each object in S_1 has a weight greater than c_2.

Then at Step 2 the algorithm receives value 1 for each object not in S_1 (either by transition bonus from Step 1, or by taking it at Step 2). The value of its solution is $\alpha|S_1| + n - |S_1|$. Now, the solution consisting of taking $\overline{S_1}$ at both time steps has value $\alpha(n - |S_1|) + n - |S_1| + n = n(2 + \alpha) - |S_1|(1 + \alpha)$. The chosen α is such that $2 + \alpha = \frac{1+\alpha}{1-\alpha}$, so the solution $(\overline{S_1}, \overline{S_1})$ has value $\frac{1+\alpha}{1-\alpha}(n - |S_1|(1 - \alpha))$. The ratio is $\frac{1+\alpha}{1-\alpha} = 2 + \alpha = 1 + \sqrt{2}$. □

Theorem 3.9. *Consider the GE model with Hamming bonus. For every $\epsilon > 0$, there is a T sufficiently large such that, there is no $(\alpha - \epsilon)$-competitive algorithm where* $\alpha = \frac{6 \cdot \sqrt[3]{9+\sqrt{87}}}{\sqrt[3]{6 \cdot (9+\sqrt{87})^2} - \sqrt[3]{36}} \approx 1.696$.

Proof. Consider some $\epsilon > 0$ and some online algorithm A. The ground set only consists of the single item 1, that is, $N = \{1\}$.

At time 1, it is not feasible to pick the item, that is, $\mathcal{F}_1 = \{\emptyset\}$. We partition the remaining time horizon $\{2, 3, \ldots, T\}$ into *phases*. Hence, the first phase starts in time step 2. In any phase, as long as A has not included item 1 in its solution until time $t < T$, both including and not including it is feasible at $t + 1$, that is, $\mathcal{F}_{t+1} = \{\emptyset, \{1\}\}$. Once A includes the item in its solution at time $t < T$ (meaning $S_t = \{1\}$), including it becomes unfeasible at the next time, that is, $\mathcal{F}_{t+1} = \{\emptyset\}$. The current phase also ends at this time. In this case, we say that the phase ends *regularly*. At $t = T$, the current phase ends *by default* in any case. If a phase however ends regularly at time $t + 1 < T$, a new phase starts at time $t + 2$.

There is no profit associated with the empty set, that is, $p_t(\emptyset) = 0$; the profit $p_t(\{1\})$ is β whenever t is the first time step of a phase, and it is γ in all other cases (note that, however, it may be unfeasible to include item 1 in the solution). The remaining part of the proof is concerned with finding β, γ so as to maximize the competitive ratio.

For the analysis, denote by $\mathcal{S}^* = (S_1^*, S_2^*, \ldots, S_T^*)$ the optimal solution, and denote by $\mathcal{S} = (S_1, S_2, \ldots, S_T)$ the solution that A finds. We consider phases separately. First consider a phase of length ℓ starting at time t_0 ending regularly (at time

$t_0+\ell-1$). Note that $\ell \geq 2$ and that the initial situation is independent of t_0 and ℓ in that $1 \notin S_{t_0-1}$. For each time t that is part of the phase, we count $b(S_{t-1}, S_t)+p_t(S_t)$ and $b(S^*_{t-1}, S^*_t) + p_t(S^*_t)$ towards the values of the algorithm and optimum, respectively. If $\ell = 2$, the resulting values of the algorithm and optimum are $\max\{\beta, 2\} \geq 2$ and β, respectively. If $\ell > 2$, the value are $\max\{\beta+(\ell-2)(1+\gamma), \ell\} \geq \beta+(\ell-2)(1+\gamma)$ and $(\ell - 2) + \gamma$, respectively. Hence, in phases of length at most 2, the optimum does not pick the item; in longer phases, it picks the item at all times when it can.

To express the lower bound that we can show, first note that assuming that each phase ends regularly is only with an additive constant loss in both the algorithm's and the optimum's value, so we may make this assumption for the asymptotical competitive ratio considered here. Since the algorithm chooses the phase lengths, the lower bound α that we can show here is equal to the largest lower bound on the ratio between the optimum's and the algorithm's value within any phase, which is lower bounded by

$$\min\left\{\frac{2}{\beta}, \inf_{\ell\in\mathbb{N};\ell\geq 3} \frac{\beta + (\ell - 2)(1 + \gamma)}{(\ell - 2) + \gamma}\right\} \tag{3.9}$$

according to the above considerations.

Note that the infimum in (3.9) is minimized when its argument is identical across all γ. This is the case when

$$\frac{1 + \beta + \gamma}{1 + \gamma} = 1 + \gamma \Leftrightarrow \gamma = \frac{1}{2} \cdot (\sqrt{4\beta + 1} - 1).$$

Furthermore, (3.9) is minimized when both its arguments are identical, meaning

$$1 + \frac{1}{2} \cdot (\sqrt{4\beta + 1} - 1) = \frac{2}{\beta} \Leftrightarrow \beta = \frac{\sqrt[3]{3 \cdot (9 + \sqrt{87})^2} - \sqrt[3]{36}}{3 \cdot \sqrt[3]{9 + \sqrt{87}}}$$

and therefore

$$\alpha = \frac{2}{\beta} = \frac{6 \cdot \sqrt[3]{9 + \sqrt{87}}}{\sqrt[3]{6 \cdot (9 + \sqrt{87})^2} - \sqrt[3]{36}} \approx 1.696.$$

This shows the claim. □

3.3.2 Intersection Bonus Model

We now look at the general evolution model with intersection bonus. This model is different from the ones considered before: We first give a simple lower bound showing that there is no constant-competitive algorithm.

Theorem 3.10. *In the GE model with intersection bonus, there is no c-competitive algorithm for any constant c.*

Proof. We consider an instance with no profit. Let $T = 2$, $N = \{1, 2\}$, and $\mathcal{F}_1 = \{\emptyset, \{1\}, \{2\}\}$, that is, there are two items, and at time 1 it is only forbidden to take both of them. Assume w.l.o.g. that the algorithm does not pick item 2 at time 1. Then picking item 1 becomes infeasible at time 2 while picking item 2 remains feasible. Then the algorithm achieves 0 profit and bonus while the optimum can achieve a bonus of 1. □

Note that in this model, by adding dummy time steps giving no bonus and no profit, the previous lower bound extends to any number of time steps. This lower bound motivates considering the 1-lookahead model: at time t, besides I_t, the algorithm knows the instance I_{t+1}. It shall decide the feasible solution chosen at time t. We consider an algorithm based on the following idea: at some time step t, the algorithm computes an optimal sequence of 2 solutions $(S^*_{t,1}, S^*_{t,2})$ of value z^*_t for the subproblem defined on time steps t and $t+1$. Suppose it fixes $S_t = S^*_{t,1}$. Then, at time $t+1$, it computes $(S^*_{t+1,1}, S^*_{t+1,2})$ of value z^*_{t+1}. Depending on the values z^*_t and z^*_{t+1}, it will either choose to set $S_{t+1} = S^*_{t,2}$, confirming its choice at t (getting in this case value z^*_t for sure between time t and $t+1$), or change its mind and set $S_{t+1} = S^*_{t+1,1}$ (possibly no value got yet, but a value z^*_{t+1} if it confirms this choice at $t+2$). When a choice is confirmed ($S_t = S^*_{t,1}$ and $S_{t+1} = S^*_{t,2}$), then the algorithm starts a new sequence (fix $S_{t+2} = S^*_{t+2,1}, \dots$).

More formally, let $(S^*_{t,1}, S^*_{t,2})$ be an optimal solution of the subproblem defined on time steps t and $t+1$, and denote z^*_t its value (including profits and bonus between time t and $t+1$). To avoid unnecessary subcases, we consider at time T the solution $(S^*_{T,1}, S^*_{T,2})$ where $S^*_{T,2} = \emptyset$ and z^*_T is the profit of the optimal solution for the single time step T, $S^*_{T,1}$. Then consider the algorithm `Balance` which:

1. At time $t = 1$ compute $(S^*_{1,1}, S^*_{1,2})$ and fix $S_1 = S^*_{1,1}$.
2. For $t = 2$ to T: compute $(S^*_{t,1}, S^*_{t,2})$.
   - Case (1): If $t \geq 3$ and if at $t-1$ the algorithm chose S_{t-1} equal to $S^*_{t-2,2}$ (i.e., Case (3) occurred), then fix $S_t = S^*_{t,1}$.
   - Case (2): Otherwise, if $z^*_t > 2z^*_{t-1}$, then fix $S_t = S^*_{t,1}$.
   - Case (3): Otherwise fix $S_t = S^*_{t-1,2}$.

Theorem 3.11. *In the GE model with intersection bonus and 1-lookahead,* `Balance` *is a 4-competitive algorithm.*

Proof. Let V be the set of time steps in which Case (3) occurred. In the proof, intuitively we partition the time period into periods which end at some time $t \in V$, and prove the claimed ratio in each of these sub-periods.

Formally, let u, v $(u < v)$ be two time steps in V such that $w \notin V$ for any $u < w < v$. Note that since Case (3) occurred at time u, Case (1) occurred at $u+1$, so $u \neq v-1$, and Case (2) occurred at time $u+2, \ldots, v-1$. So $z_t^* > 2z_{t-1}^*$ for $t = u+2, \ldots, v-1$. By an easy recurrence, this means that, for all $t \in \{u+1, \ldots, v-1\}$, we have $z_t^* < z_{v-1}^*/2^{v-1-t}$. By taking the sum, we get $\sum_{t=u+1}^{v-1} z_t^* < 2z_{v-1}^*$. Since Case (3) occurred at v, $z_v^* \leq 2z_{v-1}^*$. Finally:

$$\sum_{t=u+1}^{v} z_t^* \leq 4z_{v-1}^*$$

Now, at each time v for which case (3) occurred, we choose $S_v = S_{v-1,2}^*$. As previously said, Case (3) did not occur at $v-1$, so we choose $S_{v-1} = S_{v-1,1}^*$. Then the algorithm gets value at least z_{v-1}^* for these two time steps $v-1$ and v. In other words $f(S_1, \ldots, S_T) \geq \sum_{v \in V} z_{v-1}^*$. Consider first the case where $T \in V$ (Case (3) occurred at time T). Then we get a partition of the time steps into subintervals ending in $v \in V$. So

$$\sum_{t=1}^{T} z_t^* \leq 4 \sum_{v \in V} z_{v-1}^* \leq 4f(S_1, \ldots, S_T)$$

. Let $(\hat{S}_1, \ldots, \hat{S}_T)$ be an optimal solution. We have $\forall t$ $p_t(\hat{S}_t) + p_{t+1}(\hat{S}_{t+1}) + b(\hat{S}_t, \hat{S}_{t+1}) \leq z_t^*$. So $f(\hat{S}_1, \ldots, \hat{S}_T) \leq \sum_{t=1}^{T-1} z_t^*$, and:

$$f(\hat{S}_1, \ldots, \hat{S}_T) \leq \sum_{t=1}^{T} z_t^* \leq 4f(S_1, \ldots, S_T)$$

Note that this is overestimated, as $p_t(\hat{S}_t)$ appears two times in the sum.

Now, if $T \notin V$, then $T-1 \in V$: indeed, Case (2) cannot occur at time T (since $z_T^* \leq z_{T-1}^*$). So we have in this case:

$$\sum_{t=1}^{T-1} z_t^* \leq 4 \sum_{v \in V} z_{v-1}^* \leq 4f(S_1, \ldots, S_T)$$

But again since $f(\hat{S}_1, \ldots, \hat{S}_T) \leq \sum_{t=1}^{T-1} z_t^*$, we have

$$f(\hat{S}_1, \ldots, \hat{S}_T) \leq \sum_{t=1}^{T-1} z_t^* \leq 4f(S_1, \ldots, S_T)$$

This completes the proof. □

We prove a matching lower bound. The idea is as follows: As can be seen from the proof of Theorem 3.11, the estimate on the profit has slack for the 4-competitive algorithm. We give a construction in which there is no profit and in which the bonus when not "committing" to the solution from the previous time step is geometrically increasing over time; otherwise the bonus is 0. As it turns out, however, when the factor is 2 in each time step, we cannot show a lower bound of 4 in case the algorithm does not commit until the last time step. Interestingly, if we use the minimum factor to show a lower bound of $4 - \epsilon$ in case the algorithm commits at any time step but the last, we can find a large enough time horizon such that, in case the algorithm commit only in the last time step, we can also show a lower bound of $4 - \epsilon$.

Theorem 3.12. *Consider the GE model with intersection bonus. For any $\epsilon > 0$, there is a T_ϵ such that, for each number of time steps $T \geq T_\epsilon$, there is no $4 - \epsilon$ competitive algorithm.*

Proof. Consider some 1-lookahead algorithm $\mathtt{A}$ and $\epsilon \in (0, 1)$. We will show that there is some number of time steps T_ϵ such that for all numbers of time steps $T \geq T_\epsilon$ $\mathtt{A}$ is not $(4 - \epsilon)$-competitive. The construction is based on an increasing sequence $a_1, a_2, \ldots, a_{T_\epsilon}$ of natural numbers that we will determine later (along with T_ϵ). In the ground set, there is precisely one item (i, j) for each $i \in \mathbb{N}$ with $1 \leq i \leq T_\epsilon$ and $j \in \mathbb{N}$ with $1 \leq j \leq \max_{1 \leq k \leq T_\epsilon} a_k$. We denote the set $\{(i, j) \mid 1 \leq j \leq a_t\}$ by $R_{i,t}$. In this instance, value can only be obtained from transition bonuses.

Depending on the actions of $\mathtt{A}$, we will define some time $t^\star$ with $2 \leq t^\star \leq T_\epsilon$. At time $t > t^\star$, the empty set will be the only set that can be selected. At time $t = 1, 2$, selecting any set $R_{i,t}$ for some i or the empty set is feasible. If $\mathtt{A}$ selects the empty set in either the first or the second time step, we simply set $t^\star := 2$.

Otherwise, at time t with $2 < t \leq t^\star$, selecting any set $R_{i,t}$ for some i such that $\mathtt{A}$ has not selected $R_{i,t'}$ at any time $t' \leq t - 2$ or the empty set is feasible. If $\mathtt{A}$ selects $R_{i,t-1}$ *and* $R_{i,t}$ for some i and $t \geq 2$, we say that $\mathtt{A}$ *confirms* at time t. Then,$\mathtt{A}$ *confirms* if $\mathtt{A}$ chooses the empty set at time t and set $t^\star := t + 1$; if $\mathtt{A}$ never confirms, then $t^\star := T_\epsilon$. Note that this is a feasible construction for the 1-lookahead model.

We consider the competitive ratio in different cases:

- If $\mathtt{A}$ chooses the empty set at some time $t \leq t^\star$, $\mathtt{A}$ does not obtain value at all while the optimum can obtain positive value (at least a_1), so $\mathtt{A}$ is not competitive.

- If $\mathtt{A}$ confirms at time $t \geq 2$, it obtains value a_{t-1}. Note that there exists some $i^\star$ so that $\mathtt{A}$ never chooses $R_{i^\star,t'}$ for any t'. The optimum chooses $R_{i^\star,t'}$ for all time steps $t' = 1, \ldots, \min\{t + 1, T_\epsilon\}$. We distinguish two cases.

  - We have $t + 1 \leq T_\epsilon$. Then the total value of the optimum is $\sum_{j=1}^{t} a_j$, leading to competitive ratio $\sum_{j=1}^{t} a_j / a_{t-1}$.

  - We have $t+1 > T_\epsilon$, implying $t = T_\epsilon$ (otherwise A could not have confirmed at t). Then the total value of the optimum is $\sum_{j=1}^{T_\epsilon-1} a_j$, leading to competitive ratio $\sum_{j=1}^{T_\epsilon-1} a_j / a_{T_\epsilon-1}$.

- If A never confirms, A does not obtain value either while the optimum can obtain value $\sum_{j=1}^{T_\epsilon-1} a_j > 0$. So in this case A is not competitive either.

For convenience, we will now describe a sequence $a'_1, a'_2, \ldots, a'_{T_\epsilon}$ of rational numbers; $a_1, a_2, \ldots, a_{T_\epsilon}$ can then be obtained by multiplying all numbers in the former sequence with a suitable natural number. Let $\epsilon' \in (0, \epsilon)$ be rational. Now the goal can be reformulated to choose $a'_1, a'_2, \ldots, a'_{T_\epsilon}$ such that the ratios

$$\frac{\sum_{j=1}^{t} a_j}{a_{t-1}} = \frac{\sum_{j=1}^{t} a'_j}{a'_{t-1}} \text{ for all } t < T_\epsilon \tag{3.10}$$

and

$$\frac{\sum_{j=1}^{T_\epsilon-1} a_j}{a_{T_\epsilon-1}} = \frac{\sum_{j=1}^{T_\epsilon-1} a'_j}{a'_{T_\epsilon-1}} \tag{3.11}$$

(corresponding to the above ones) are all at least $4-\epsilon'$. To do so, we start by setting $a'_1 := 1$. Now we inductively define a'_t for $t \leq T_\epsilon - 2$ (note that T_ϵ is yet to be defined). Assuming all $a'_1, \ldots, a'_{t-1}$ are defined, we set a'_t to be such that (3.10) for t is precisely $4-\epsilon'$. Equivalently, set $a'_t := (4-\epsilon') \cdot a'_{t-1} - \sum_{j=1}^{t-1} a'_j$.

We claim there exists a (first) t_0 such that

$$(4-\epsilon') \cdot a'_{t_0} - \sum_{j=1}^{t_0} a'_j \leq a'_{t_0} \tag{3.12}$$

(meaning the (t_0+1)-st element of the sequence $a_1, a_2, \ldots$ would become smaller than a'_{t_0}). Then we set $T_\epsilon := t_0 + 2$ and $a'_{t_0+2} = a'_{t_0+1} = a'_{t_0}$. Note that then indeed, by (3.12) and $a'_{t_0+1} = a'_{t_0}$, (3.10) is at least $4-\epsilon'$ for $t = T_\epsilon - 1$ (and therefore, by the previous argument, for all). Furthermore, since $a'_{t_0+2} = a'_{t_0+1}$, (3.11) is identical to (3.10) for $t = T_\epsilon - 1$ and therefore also at least $4-\epsilon'$.

So it remains to show the claim. Define

$$b_t := \frac{(4-\epsilon') \cdot a'_t - \sum_{j=1}^{t} a'_j}{a'_t}.$$

for all t including the first element where the fraction becomes at most 1 (if it exists; otherwise the sequence is infinite). Further note that for all such $t \geq 3$ we have $\sum_{j=1}^{t} a'_j = (4-\epsilon') \cdot a'_{t-1}$ (by using the definition for a_{t-1}). Therefore $b_t = (4-\epsilon') \cdot (1 - a_{t-1}/a_t) = (4-\epsilon') \cdot (1 - 1/b_{t-1})$.

We now show two properties of the sequence $b_1, b_2, \ldots$:

- If $b_t \geq 1$ for $t \geq 3$, then $b_{t+1} < b_t$. Note that this expression simplifies to to $b_t^2 - (4 - \epsilon') \cdot b_t + (4 - \epsilon') > 0$, which is true for all $b_t \geq 1$.
- The sequence $b_1, b_2, \ldots$ does not converge to a value at least 1. Suppose it did. This would imply there exists $x \geq 1$ with $x = (4 - \epsilon') \cdot (1 - 1/x)$, which however does not have a real solution.

By basic calculus, this proves that there exists a t with $b_t \leq 1$, implying the claim, which in turn implies the theorem. □

3.4 Conclusion

In this chapter, we have developed techniques for online multistage subset maximization problems and thereby settled the achievable competitive ratios in the various settings almost exactly. Disregarding asymptotically vanishing terms in the upper bounds, what remains open is the exact ratio in the general-evolution setting with Hamming bonus (shown to be between $1 + \sqrt{2}$ and 3 in this chapter) and exact bounds for the models with Hamming bonus when $T \to \infty$. Furthermore, it is plausible that the ratios can be improved for (classes of) more specific problems.

In the next chapter is presented a direct application of the multistage framework in a musical context.

Chapter 4

Target-based computer-assisted orchestration problem

We previously presented some theoretical analysis of the multistage framework, both in the offline setting studying the MULTISTAGE KNAPSACK problem in Chapter 2 and in the online setting with the study of the large family of the subset maximization problems in the multistage framework in Chapter 3. In this chapter, the TARGET-BASED COMPUTER-ASSISTED ORCHESTRATION problem is addressed, being a direct application of the multistage framework in a musical environment. In Sections 4.1 and 4.2, a global introduction of the musical application is given (this introduction is mostly based on the YouTube tutorial by C. E. Cella (Cella (2020a) - link accessed on Sept. 2020). Note that some of the figures of these sections are taken from this video. Then, Section 4.3 are presented the problem we focus on and our results.

An extended abstract of this chapter will be put on arXiv and submitted to a journal by the time of the thesis defense.

4.1 Introduction

Among the different aspects of musical writing, musical orchestration is probably the most empirical one and it has been traditionally taught in an intuitive and non formalized way. Treatises in orchestration are often made of a sequence of recipes on instruments and do not take into account the theoretical studies made on the orchestration problem directly.

More recently, however, musical composers started imagining highly complex timbres made of extended instrumental techniques and needed a more systematic approach to orchestration and to composition in general.

This motivated the development of computational tools able to assist the pro-

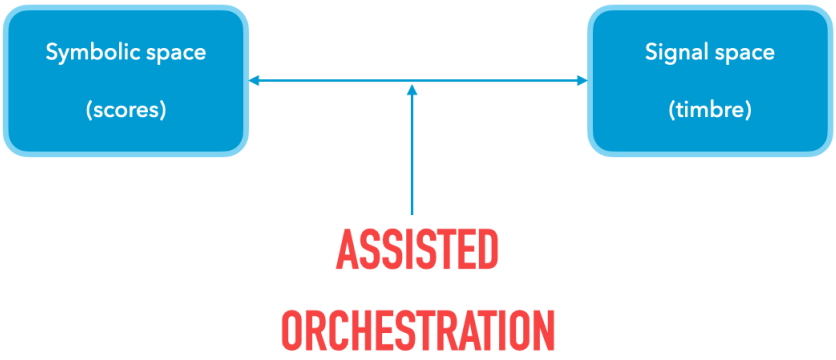
cess of musical composition and consolidated in an import research area known as *Computer-Assisted Composition (CAC)* (Fernández and Vico (2013),Ariza (2005)). Within CAC, target-based computer-assisted orchestration is an important example of how computers can be used for assisting musical orchestration (Maresz (2013)).

Target-based computer-assisted orchestration can be defined as follows.

Definition 4.1. (Assisted orchestration) *Assisted orchestration can be seen as the process of searching for the best combinations of orchestral sounds to match a target sound under specified metric constraints.*

The notions presented in this definition will be discussed and addressed later in this section.

In a nutshell, target-based computer-assisted orchestration is the fact of making a connection between the symbolic space of music which is the score of a music (note that the score of a music in this section will refer to its music sheet), seeing music as a combination of symbols describing it with notes, pitches, instruments and so on, and the signal space of music which is the timbre of a sound, the sound itself.



4.1: Assisted orchestration as the connection between symbolic space and signal space

While at the origin of computer-assisted composition, contemporary composer focused more on the symbolic space of musical writing, in the last thirty years the attention shifted on the acoustic space of music. Composers started thinking music more in term of sound spectra than simple chords and this converged in the definition of spectralism, an important musical aesthetics that was developed in France since the early 90's.

4.1.1 Target-based music orchestration

This section will introduce some basic definitions about sounds and signal that will be used later in the chapter.

Preliminaries

A **sound** is a periodic wave. It is possible to decompose this periodic wave, seen as a periodic function, using Fourier series which are weighted sums of sinusoids.

The different sinusoids, being distinct from one another in terms of their frequencies, compose the harmonic series of a sound. In this series, the fundamental harmonic, i.e., the harmonic with the lowest frequency and the one that is the most noticeable to a human ear, is called the first **partial** and the other frequencies, i.e. the other **partials** of a sound, create the rest of the sound.

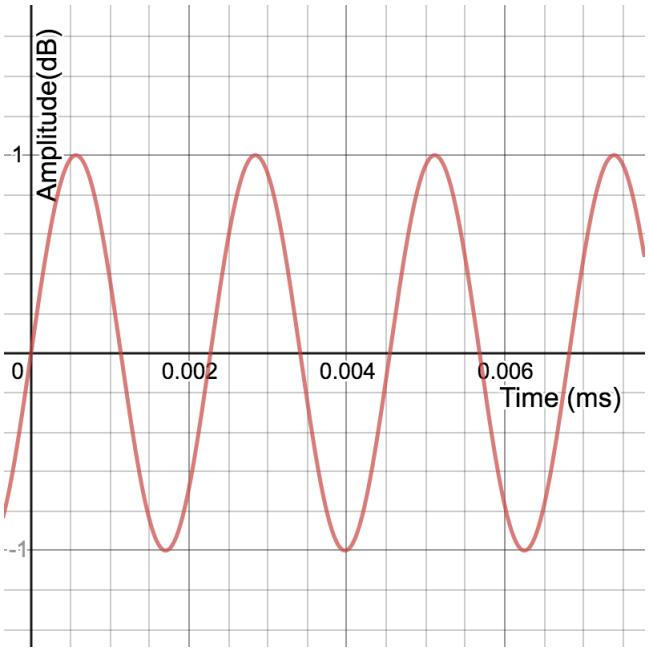
The first partial usually define the **pitch** of a sound, represented by notes letter from A to G#.

The **timbre** of a sound can be seen as the quality of a sound, each instrument like a cello or a violin producing its own quality of sound. It is the timbre that gives the possibility to a human ear to identify and recognize one instrument from another. Formally, it is directly created by the harmonic series, i.e. the set of partials of a sound.

The **amplitude** of a sound is its volume, measured in dB.

Brief history The first well known application of assisted orchestration is a piece done by the English composer J. Harvey called *Mortuos Plango, Vivos Voco (1981)*. Harvey tried in this piece to transcribe the sound of the bell. The transcription was done manually by looking at the partials of the recorded bell sound.
Later in 2008, in a piece called *Speakings*, J. Harvey used *Orchidee*, a system for static target-based orchestration discussed later in the chapter, in order to replicate the sound of a human voice.

To write a music score, a composer writes symbol on a music sheet. Then, the symbols are "played" to generate a sound.
Assisted orchestration can be seen as the inverse process. Given a sound, assisted orchestration retrieves its corresponding score. This differs from music transcription. In assisted orchestration the sound can be anything like bells, noise, voices and so on, for which there is no corresponding score. In a general sense, assisted orchestration can be thought of as a generalisation of musical transcription.



4.2: Representation of the note A

The assisted orchestration problem can also be seen as the problem corresponding to, given an orchestration, reproducing a target timbre within a compositional context. Namely, given specific musical constraints by the composer like an orchestra with a set of chosen instruments, playing styles, dynamics and so on, is it possible to be as close as possible to the target sound.

Let us present an example in order to show the hardness of the problem.

Example 4.1. *Given a target sound, one is asked to reconstruct the sound using instruments of a restricted orchestra composed of only two instruments being able to play only two notes at a fixed single dynamic.*
To solve this problem, one has to find the best combination of sounds, being the closest one to the target sound, among the set of possible combinations. Indeed, one can choose to take the first note on the first instrument and on the other instrument, the second note on the first instrument and on the other instrument, the first note on the first instrument and the second note on the other instrument, the second note on the first instrument and the first note on the other instrument. For each of the four combinations, one has to compute a distance between the combination and the target sound. Under this specific set of very restricted constraints it is easy to find the best possible solution in a small amount of time.

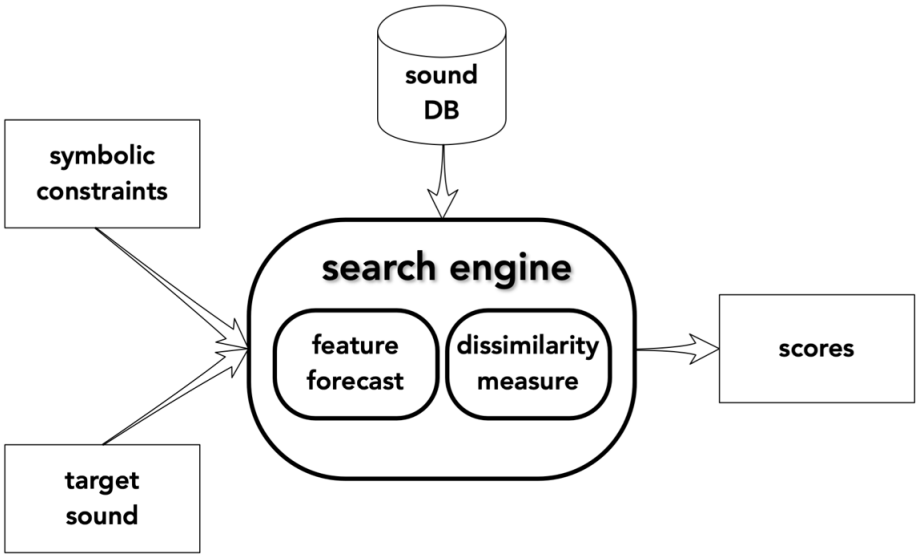
However, in practice the set of feasible solutions is way larger: each instrument of an orchestra has the possibility to play around forty notes, several dynamics, articulations, playing styles and so on. The number of different combinations in a basic setting is around 2^{30000} making the problem intractable. An analysis of the problem complexity will be presented later in this chapter.
Typical solutions to this problem are orchestral scores that include specific instruments, notes, dynamics and playing styles such that, when played, they sound perceptually similar to the target sound. In principle, to find the combination of orchestral sounds that is the closest to the target, it would be enough to compute all possible combinations of the sounds in the database and rank them by a distance measure with the target sound. This approach, while conceptually justified, is normally not computable given the large amount of possible combinations.

The intractability of the problem leads to the introduction of heuristics to navigate the combinatorial space. The general structure of the algorithm (see Figure 4.3) consists of,

- Given:
  - a **target sound**
  - a set of **symbolic constraints** i.e. musical constraints
  - a set of **sounds** in the database
- a **search engine** computing:
  - the **feature forecast**: for each combination its feature are forecast using specific strategies
  - the **dissimilarity measure**: measure evaluating the distance between two sounds
- the search engine then optimizes the instance and gives a **score** as an output

The outcome of an assisted orchestration algorithm is a score, i.e. a set of symbols, and not sounds. One can play the score using some sound database containing the instruments of the solution in order to be able to directly compare the solution sound to the target sound, but the aim of the algorithm is to help a composer reproduce a given sound with a set of instruments.

Note that the choice of the dissimilarity measure and its distance is still an open question in the literature. The hardness of this choice comes from the fact that the two compared sounds have to be close for example regarding their respective spectrum (this notion will be defined later), but this does not necessarily imply that the two sounds will be close perceptually.



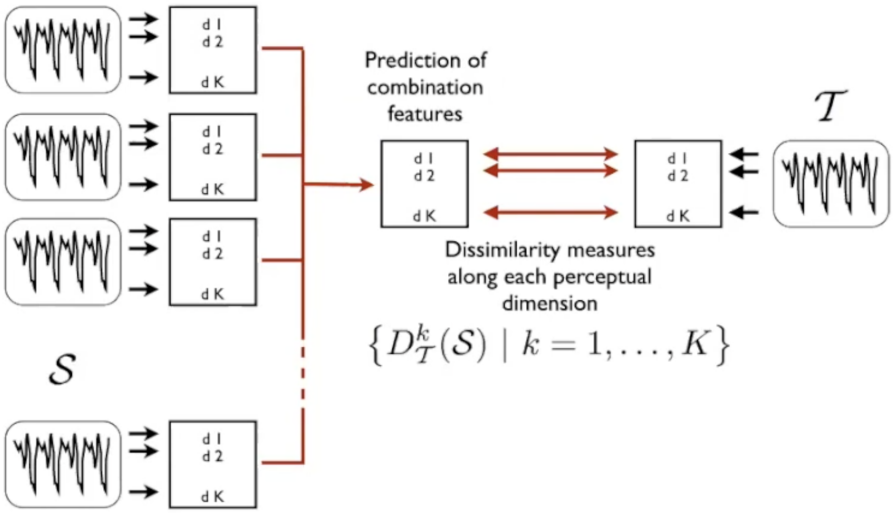
4.3: General structure of the assisted orchestration algorithm

4.1.2 The search engine: features forecast and dissimilarity measure

Let us now develop the search engine and the notions behind the feature forecast and the dissimilarity measure.

The sounds of the database are described by features. Each feature (a few of them exist in the literature) describes the sound with a specific metric and has a fixed number of dimensions. For example, in Figure 4.4, each sound is described by a feature of dimension K, i.e., to each sound is associated a vector value of dimension K: $d_1, \ldots, d_K$ (note that, in our analysis presented later in the chapter, we say that to each sound i is associated a vector value of dimension M: $p_{i1}, \ldots, p_{iM}$). In the first assisted orchestration algorithm that we will present, one makes a random combination between the sounds (the techniques used by the algorithm will be developed later in the chapter). Then, with the combination obtained, one needs to find the features of the combination, i.e. the features of the sounds selected by the algorithm. However, in practice, it is not possible to compute the exact features of the combination, being too large and taking too much time to compute. This is where the notion of **features forecast** is introduced. Indeed, features are only predicted and their calculation themselves is a difficult problem, the features being, for most of the cases non linear features (two ways of computing the features will

be presented in the chapter).
Once the features forecast of the combination is predicted, one needs to describe the target sound with the same features (note that here, the features of the target sound are its "real" features, the complexity of computing them being small because the calculation is being applied only to one sound). Only then, a distance is computed, the **dissimilarity measure**, between the features of the target sound and the features of combination found by the algorithm.



4.4: The search engine

We see in Figure 4.4 (taken from Carpentier (2008)) an illustration of the search engine, the set $\mathcal{S}$ of sounds on the left of the figure being the sounds of the database described with K dimension features. A random combination of sounds is then generated and its features description predicted. We see on the right of the figure the target sound $\mathcal{T}$ described also with K dimension features. The two of them are then compared using a distance, the dissimilarity measure $D_{\mathcal{T}}^k(\mathcal{S})|k = 1, \ldots, K$, applied on the features. The measure will be developed later in the document (note that the choice of the problem parameters notations will be different, for the sake of clarity, in our analysis presented later in this chapter with a different notation for the distance, a number of dimension M and p_{ij} the value of a sound i in dimension j).

4.2 An overview of target-based music orchestration

In this section we will present the Target-based computer-assisted orchestration and the software called *Orchidea* doing it.

The main reference page is `www.orch-idea.org` (last accessed on September 2020) where are presented the software, the system and the information.
The *Orchidea* software is an assisting tool that does not orchestrate but helps the composer to orchestrate. It was developed by a few researchers including Carmine Cella, currently teaching at University of California, Berkeley and that is the author of the latest version of the software. He worked with IRCAM (Institut de recherche et coordination acoustique/musique), UC Berkeley and HEM (Haute Ecole de Musique de Geneve).

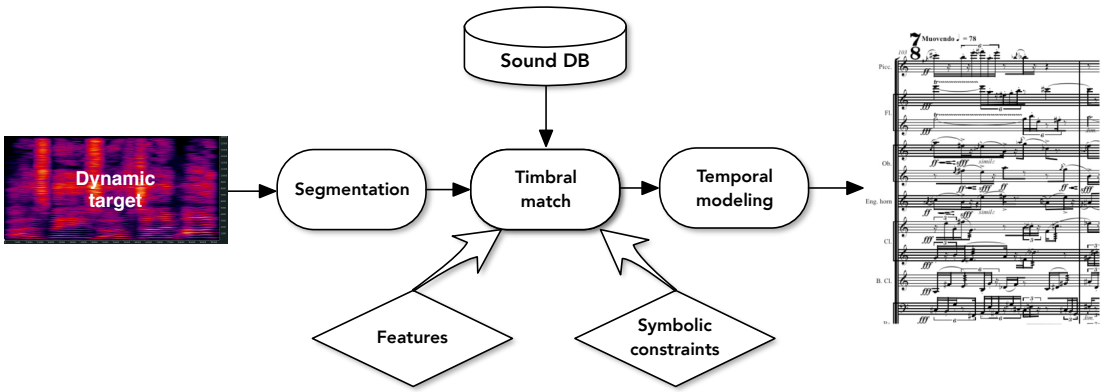
The approach implemented in *Orchidea* works as follows, and will be developed more into details in this chapter:

- a target sound and a database of orchestral sounds are embedded in a high-dimensional feature space;
- the target sound is cut into segments that represent the temporal variations;
- each segment is timbrally matched against a large number of combinations of sounds of the database by means of a multi-stage algorithm that optimise both the features and the symbolic constraints;
- the best match for each segment is then connected temporally to the other best matches in order to produce a musically meaningful score.

The approach discussed above is depicted in Figure 4.5 in the subsection presenting the *Orchidea* software.

4.2.1 The original software: *Orchidee*

Let us now present the idea behind the assisted orchestration software called *Orchidee*.
Indeed, a solution to the target-based music orchestration problem proposed in Carpentier et al. (2007), consists in using a multi-objective genetic heuristic and a constraint solver that are jointly optimized. The target sound and the sounds in the database are embedded in a low-dimensional feature space (in contrasts with the high-dimensional one for *Orchidea*) and each generated combination of sounds



4.5: An overview of *Orchidea*, the state-of-the-art system for dynamic target-based orchestration. The three main stages of the algorithm implemented in *Orchidea* are the segmentation of the target, the timbral match and the temporal modelling of the solutions.

is evaluated with a distance metric. This approach is solely focused on the so-called *static* aspect of the problem: the target sound is considered as time-invariant and the algorithm does not consider any timbral change over time.
Using only a few features, the sound is embedded in a low dimension space. The selected features describing the sound of the database are called perceptual features i.e. relevant for the perception features such as the pitch, the energy, the brightness, the bandwith,....

Then a multiobjective optimization algorithm is applied to the instance, minimizing jointly all the features describing the sounds with value $D_{\mathcal{T}}^{k}(\mathcal{S})|k = 1, \ldots, K$. The algorithm then generates a set of solutions that can be represented as a Pareto front.

Indeed, each solution on the Pareto front would be better according to a given feature, for example, one solution would have the best pitch, another would have the best brightness and so on.
This approach was very useful for the composers that could choose between the set of all the Pareto dominating solutions according to their personal listening preferences between the features. The assisted orchestration problem was implemented this way around 2007 in the *Orchidee* software.
This approach was developed and presented in the thesis by Carpentier in 2008 (Carpentier (2008)).

In order to address a risk of combinatorial explosion, a genetic algorithm was developed to thwart the impossibility of computing all the combinations of sounds.

The genetic algorithm indeed generates a random combination of sounds "good enough", i.e. not optimal, but close to it.
Let us now present informally the genetic algorithm used in the *Orchidee* software. It starts with a random combinations of sounds, for example 500 random combinations of sounds, and then the *fitness* of the combination is calculated, ranking it in comparison to the target. Then, genetic transformations called crossovers and mutations are applied to the best found combination of sounds. The crossover transformation takes, given two combinations, part of one combination and part of the other combination. Then, the algorithm generates a new combination of sounds being a mix of the two combinations. Next, the algorithm applies a mutation on the produced combination of sounds, i.e. altering the combination of sounds by a random factor. Finally, the fitness of this last combination is computed again, in order to be able to rank again the solutions found. Then, the mutation and the crossover operations are applied again on the best found solution, generating a new solution on which the *fitness* is computed again. The algorithm does these series of operations over and over again until the solutions found converge, i.e., the value of the solution does not change too much between two series of genetic operations. The solution found at the end is "good enough". To be more precise, the genetic algorithm of *Orchidee* works as follows:

Genetic assisted orchestration algorithm

1. Initialization:
    - Generate randomly an initial combination of sounds,
    - Compute the fitness of the combination of sounds.
2. While the fitness of the solution has not converged do:
    - Rank the solution found and find the best combination of sounds,
    - Apply the crossover operation to the best solution found,
    - Apply the mutation operation to the best solution found,
    - Compute the fitness of the solution.
3. Output the solution

This algorithm was developed in 2008 and is the foundation of the current assisted orchestration tool that we are going to present now.

4.2.2 *Orchidea*

While the problem adressed by the *Orchidee* software is an interesting problem, in practice no real sounds exhibit a time-invariant nature. As such, the computer-assisted orchestration problem can be formulated in a *dynamic* manner: the parameters of the algorithm should change over time, to account to the timbral changes of the target sound. This approach, that requires many more parameters and has a greater computational complexity, has been discussed in Cella and Esling (2018), Cella (2020b) and is implemented in the *Orchidea* toolbox for assisted orchestration (www.orch-idea.org), currently considered the state-of-the-art system for target-based computer-assisted dynamic and static orchestration.

Orchidea is the assisted orchestration tool on which our theoretical studies presented later in the document were based on. *Orchidea* focuses on both static and dynamic/temporal target sounds. Indeed, the target sound can now be dynamic, like a whole piece of music, a series of sounds, ... To deal with the temporal aspect of assisted orchestration, *Orchidea* introduces two new approaches in the assisted orchestration framework, a joint time-target optimization and connection models. These two approaches will be developed later in the section.
The implementation of *Orchidea* began in 2017 and is still on going, a first public and usable version with a modular interface was presented in 2019 on the visual programming language MAX/MSP (see 4.6), as well as a standalone application.

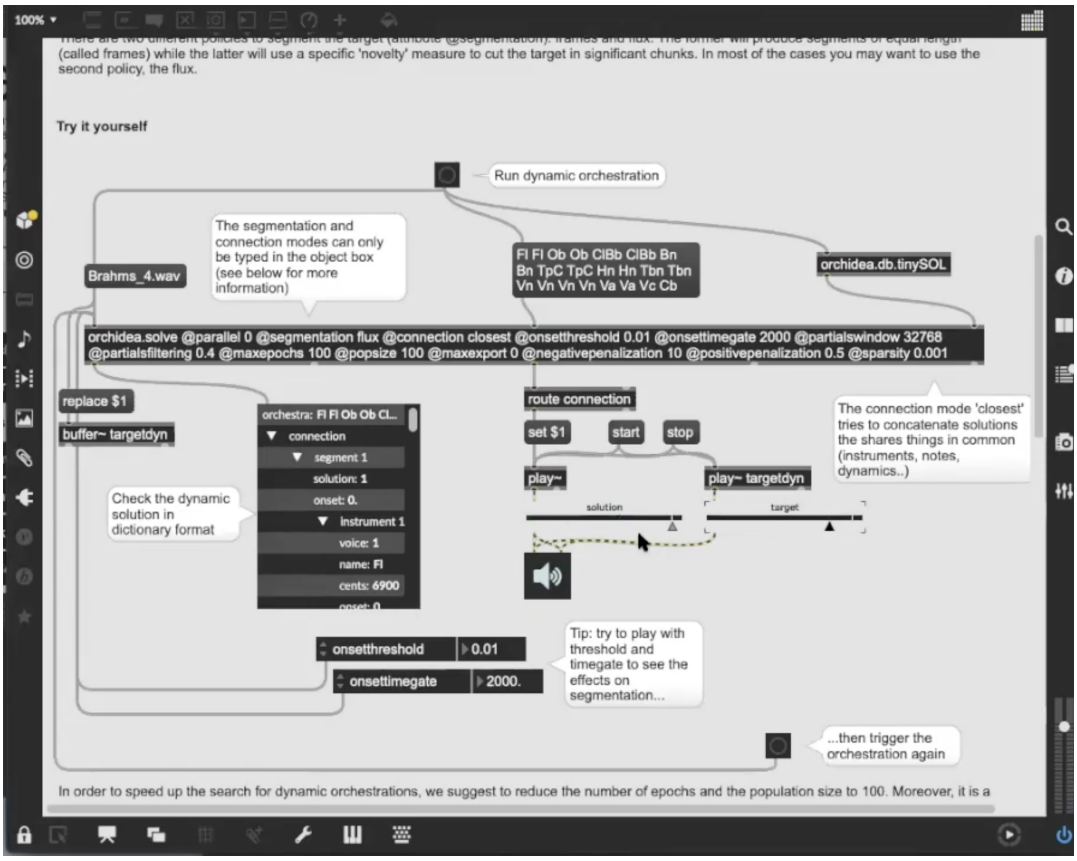
The current architecture of the *Orchidea* software can be described as:

- A high-dimensional mono-objective optimization function with high-dimensional embedding and dynamic symbolic constraints. This contrasts with the previous approach that was multi-objective with low-dimensional description. The choice of this new approach, increasing the abstraction level, was motivated by being closer to composers, to what they are looking for when they are using the assisted orchestration tool and less focused on the sound analysis itself;

- A greedy strategy is used to generate the initial population of sounds. This contrasts also with the existing approach where random combinations were used. The greedy strategy, called stochastic pursuit strategy, finds the closest best solution and generates the initial combination of sounds;

- A neural network to predict the features of the sound combinations contrasting with the previous genetic approach developed in *Orchidee*;

- An asymmetric distance for the evaluation of the solutions (the dissimilarity measure). This new distance uses two new variables called positive and negative penalizations. This will be developed later in the document;



4.6: The *Orchidea* software in MAX/MSP

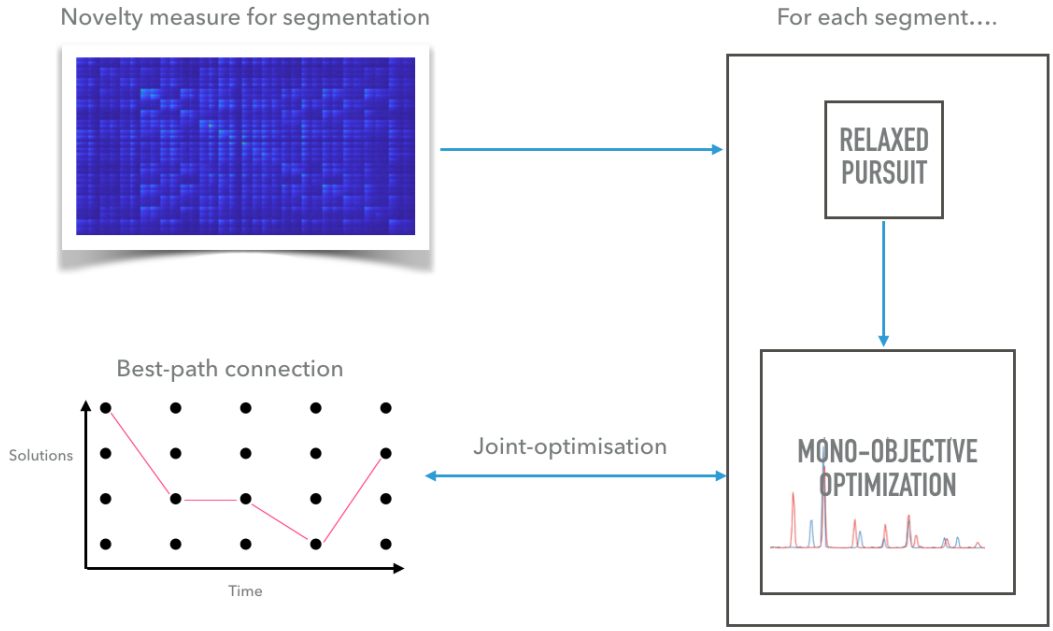
- A joint time-frequency optimization called hysteresis of the orchestration where the solutions found on previous time steps are weighted in order to connect the solutions of the current time step to the previous ones. This will also be developed later in the document;

- Temporal modeling graph: a representation of the dynamic orchestration problem as a graph and the introduction of the continuity model (note that this model will be presented further in the chapter).

A dynamic sound target is decomposed into slices respecting the temporal variations and generates a segmentation of the target with a fixed number of time steps. To be more precise, to each time step is associated a static target sound on which calculations are applied.

First, a dynamic target is segmented and decomposed into static target sounds, i.e. the temporal target is cut into slices. Then, the features of sound combinations are forecast using a neural network. Next, the forecast is matched with the target sounds using the stochastic matching pursuit and the evolutionary optimization

algorithms subject to a set of symbolic constraints. Finally, temporal modeling is calculated with a joint time-frequency optimization algorithm taking into account the continuity of the solutions, i.e., the continuity model.

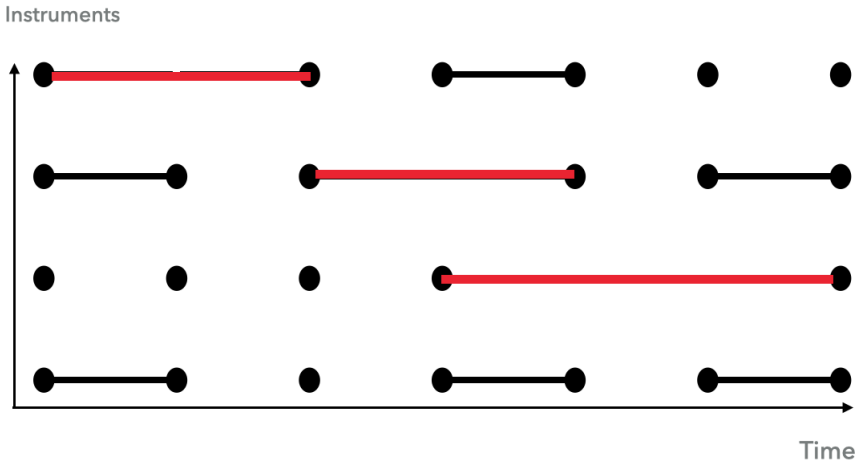


4.7: Temporal modeling of *Orchidea*

Figure 4.7 presents the way temporality is addressed in the *Orchidea* software. Given a temporal target sound, a segmentation is generated according to a novelty measure, i.e. each time the sound is "new" a new slice/time step is generated. On each time step/segment generated is applied relaxed pursuit and high-dimensional mono-objective optimization. All solutions are represented as a graph (we use the same technique later in the chapter, the construction of the graph will be detailed then) and joint-optimization is applied on both paths of the graph and the target of the next time step in the instance of dynamic orchestration. Indeed, a best-path connection is done connecting the different segment/time steps of the dynamic orchestration, in order to minimize both the distance to the target sound and the changes in the solution between two consecutive time temps, i.e. for the solution to be as stable as possible across the time horizon. This stability echoes the multistage framework.

The stability of a solution in a dynamic orchestration problem is evaluated using a so-called continuity model, limiting as much as possible the movements in the

solution. This means that when a sound is taken it is best to keep it at the next time step. Indeed, if a sound is kept for two consecutive time steps, the solution will "connect" the different time steps together. This connection implies better results acoustically as the sound generated by the instrument will be continuous. In Figure 4.8 is presented a representation of the model where a red line connecting several dots means that a same instrument is selected in the solution for consecutive time steps.



4.8: Illustration of the continuity model

4.3 Target-based computer-assisted orchestration: a theoretical analysis

We presented in the previous section the musical problem Target-based computer-assisted orchestration and its applications. Now we will address the problem from a theoretical point of view, as it is closely related to the multistage framework, and compare results obtained with an ILP formulation to the *Orchidea* ones.

4.3.1 Problems definition and results

We now formally define the target-based orchestration problems we focus on in this chapter.

Sounds. We have a set $N = \{s_1, s_2, \ldots, s_n\}$ of n sounds, corresponding to the database of sound samples. Each sound is produced by an instrument. Formally, N is partitioned into $N_1, \ldots, N_Z$ where Z is the number of instruments. Besides its

instrument, each sound s_i is characterized by its values on several dimensions (feature space): formally, to s_i are associated M nonnegative values $p_{ij}, j = 1, \ldots, M$, where M is the number of dimensions.

We also have a target sound G, typically not in the database N. As any sound, G has a description in the feature space, i.e., it has a value G_j on each dimension $j = 1, \ldots, M$.

Subset of sounds. The goal of the problem is to choose a subset of sounds to be played by the orchestra to reproduce in the best possible way a given target. To evaluate the quality of such a subset of sounds, we first give its value in the feature space (dimensions), and then define how to evaluate the quality of this subset with respect to a target.

Let $S \subset N$ denotes a non empty subset of sounds. The value of S on dimension j is the average value of each sound in S on this dimension. Formally, writing $p_j(S) = \sum_{s_i \in S} p_{ij}$, the value of S on dimension j is $\frac{p_j(S)}{|S|}$.

Distances. As explained before the value of (a subset of) sounds on the dimensions allows to define a distance between them, measuring how different they are (also called before the dissimilarity measure). Formally, let S, S' be two non empty subset of sounds. The *distance* between these two subsets is defined as

$$d(S, S') = \sum_{j=1}^{M} \left| \frac{p_j(S)}{|S|} - \frac{p_j(S')}{|S'|} \right|$$

Note that $d(S, S') = 0$ if and only if S and S' have the same value on each dimension.

Orchestration constraints. As explained in Section 4.1, in the target-based orchestration problem, we want to reproduce with an orchestra a given target sound. Then, the sounds we select in the database (to be played by the orchestra) must respect some constraints based on the orchestra composition (if there are two transverse flute, then we cannot select more than two sounds played by a transverse flute). Formally, we can have:

- For each instrument z, a constraint L_z on the number of selected sounds played by instrument z that can be selected. The constraint is hence $|S \cap N_z| \leq L_z$.
- A constraint on the total number of chosen sounds: $|S| \leq L$ (this could be the total number of instruments, or a more restrictive constraint imposed by a composer).

Static target-based orchestration problem

Now we are able to define the static target-based orchestration problem S-TOP.

Definition 4.2. *In the problem* S-TOP*, we are given a set N of sounds, a target sound G, and a set of orchestration constraints. The goal is to find a nonempty set S of sounds which minimizes $d(S, G)$, while fulfilling the orchestration constraints.*

In the theoretical analysis of S-TOP, we will first focus on a simple version of the problem, where there is one instrument, no orchestration constraints, and only one dimension. We will denote this problem as S-TOP-1D.

Dynamic target-based orchestration problem.

Now let us look at the Dynamic Target-based Orchestration Problem D-TOP. Think for instance that we want to reproduce a human pronouncing a sentence (the target). As explained in Section 4.2, the target is first cut into segments, say T segments, and each segment t corresponds to a target sound G^t. The sentence can be for instance cut into syllables, giving a sequence of syllables/targets G^t.

Then, we shall select at each time step (segment) t a subset of sounds $S_t \subset N$ to reproduce G^t. The distance $d(S_t, G^t)$, measuring how far S_t is from G^t, should be minimized. Besides, from a musical perspective, it is also interesting to select subset of sounds that do not differ too much in consecutive time steps, i.e., $d(S_t, S_{t+1})$ should also be small. Typically, if selected subsets are similar, use the same instruments and/or notes, the solution sounds more melodious (see Section 4.2).

In the model we consider, we linearly aggregate these distances, and get the following problem.

Definition 4.3. *In the problem* D-TOP *we are given a set N of sounds, a set of orchestration constraints, a time horizon $T \in \mathbb{N}^*$, a target G^t for each $t \in \{1, \ldots, T\}$, and a penalty $C_t \geq 0$ for each $t \in \{1, \ldots, T\}$.*

We are asked to select T subsets of sounds $S_1, \ldots, S_T$, such as each S_t fulfills the orchestration constraints. The goal is to minimize

$$f(S_1, \ldots, S_T) = \sum_{t=1}^{T} d(S_t, G^t) + \sum_{t=1}^{T-1} C_t d(S_t, S_{t+1})$$

Note that the set of sounds N, the value p_{ij} of sounds on dimensions, and the orchestration constraints are static (do not evolve during the time horizon).

In this objective function f, the first part measures the (average) distance of subset S_t with respect to target G^t, we will call it *orchestration cost*. The second part measures the cost from moving from subset S_t to subset S_{t+1}, we will refer to it as *transition cost*. Coefficients C_t allow to balance between these two costs.

4.3.2 Contributions and organization of the chapter

The main contribution of this chapter is a theoretical analysis of algorithmic properties of S-TOP and D-TOP. We first focus on the computational complexity of the problems, both in the static case and in the dynamic case. We show that the static problem S-TOP is **NP**-hard, even in the very restricted case of S-TOP-1D (with

Problem	Complexity	Pseudo-polynomial exact algorithm	Polynomial time approximation algorithm
S-TOP-1D	NP-hard	$O(n^3 p_{max})$	$OPT + \epsilon p_{max}$ in time $O(n^3/\epsilon)$
S-TOP	NP-hard	$O(L^2 n^{M+1} p_{max}^M)$	$OPT + \epsilon p_{max}$ in time $O(L^2 n (nM/\epsilon)^M)$
D-TOP-1D	NP-hard	$O(n^3 p_{max} + T n^2 p_{max}^2)$	$OPT + \epsilon p_{max}(1 + C_{max})$ in time $O(Tn^3/\epsilon + T^3 n^2/\epsilon^2)$
D-TOP	NP-hard	$O(TL^2 (n p_{max})^{2M})$	$OPT + \epsilon p_{max}(1 + C_{max})$ in time $O(TL^2 (2nMT/\epsilon)^{2M})$

4.9: Summary of results. $p_{max} = \max_{i,j} p_{ij}$ and $C_{max} = \max_t C_t$. OPT denotes the optimal value (so error are additive in the last column).

only one dimension, one instrument and no constraint). We then show that S-TOP is solvable in pseudo-polynomial time (when the number of dimensions is fixed), by a dynamic programming (DP) procedure. Combining this procedure with a shortest path representation capturing temporality, we show that the pseudo-polynomiality result extends to the dynamic case. We also propose approximation algorithms for both problems based on rounding techniques from the DP procedures. Table 4.9 sums up the results we obtain.

We also perform some experiments to generate orchestrations on specific targets. We evaluate our results both in a qualitative and a quantitative manner. The qualitative evaluation is performed by acoustic inspection of the solution and direct comparison with the solutions of the same target generated by *Orchidea*. The quantitative evaluation uses the objective function of the considered model to compare solutions.

The remainder of the chapter is organized as follows. In Section 4.4 we focus on the restricted static problem S-TOP-1D. We tackle the general static problem S-TOP in Section 4.5, and the dynamic problems D-TOP in Section 4.6. Experimental results are given in Section 4.7.

4.3.3 Related works in the MULTISTAGE framework

As explained in Section 4.3.1, in the problem with temporality D-TOP, we build a sequence of subset sounds $S_1, \ldots, S_T$, which induces two costs: the orchestration cost $\sum_t d(S_t, G^t)$, which measures how close we are from the target, and the transition cost $\sum_t d(S_t, S_{t+1})$ which should be minimize from an acoustic viewpoint to get solution sequences that can be connected in a melodious way.

This situation, where we have a time horizon, a solution sequence to build, and an objective function mixing quality of individual solutions and transition cost

corresponds to the multistage framework with a static set of solutions studied in the thesis.

4.4 S-TOP-1D: complexity and approximation

In this section, we focus extensively on the problem S-TOP-1D, where there is one instrument, no orchestration constraint, and only one dimension. Basically, each sound s_i is characterized by one value $p_i \geq 0$.

We first show that even this very restricted case of the orchestration problem is already **NP**-hard. Let us consider the decision version of this problem, DECISION S-TOP-1D, where we are given $K \geq 0$ and we are asked if there is a feasible solution of value at most K.

We will prove the **NP**-hardness by a reduction from the EQUIPARTITION problem, known to be **NP**-complete Garey and Johnson (1979):

Definition 4.4. EQUIPARTITION *:*
INSTANCE: Finite set A of q elements and a size $v(a) \in \mathbb{N}_+^*$ for each $a \in A$.
QUESTION: Is there a subset $A' \subseteq A$ of size $q/2$ such that $\sum_{a \in A'} v(a) = \frac{B}{2}$, with $B = \sum_{a \in A} v(a)$.

Note that q and B are assumed to be even.

Theorem 4.1. DECISION S-TOP-1D *is **NP**-Complete, even with $K = 0$.*

Proof. The problem is obviously in **NP**.

Given an instance of EQUIPARTITION on the set $A = \{a_1, \ldots, a_q\}$ with $q = 2p$ elements with $B = \sum_{i=1}^{q} v(a_i)$, we build the following instance I' of S-TOP-1D:

- There is a set $N = \{s_0, \ldots, s_q\}$ of $q + 1$ sounds: each element a_i $(i \geq 1)$ corresponds to one sound s_i, with value $p_i = v(a_i)$. s_0 has a large value $\Omega = B(q + 2)$.
- The target is $G = \frac{\Omega + B/2}{p+1}$, and $K = 0$.

Note that if G is not integer, we can multiply the value of each sound by $p + 1$ and get an equivalent instance with an integer target.

With $K = 0$, the problem is equivalent to finding a subset $S \subseteq N$ s.t $\frac{p(S)}{|S|} = G$.

Suppose that there is a set A' of size $p = q/2$ of sum $B/2$. Then the corresponding set of p sounds, plus s_0, is a set S of $p + 1$ sounds, with $p(S) = \Omega + B/2$, so $\frac{p(S)}{|S|} = \frac{\Omega + B/2}{p+1} = G$.

Conversely, suppose that there is a nonempty set S of sounds of average value G. First, notice that S necessarily contains s_0, otherwise $p(S) \leq B$, and $p(S)/|S| \leq B$, while $B < G$ (as $\Omega > B(p + 1)$).

Then, $S = \{s_0\} \cup S'$. We show that $|S'| = p$:

- Suppose first that $|S'| \geq p+1$. Then $\frac{p(S)}{|S|} \leq \frac{\Omega+B}{p+2}$. But with the choice of Ω, $\frac{\Omega+B}{p+2} < \frac{\Omega+B/2}{p+1} = G$, a contradiction.
- If $|S'| \leq p-1$, then $\frac{p(S)}{|S|} \geq \frac{\Omega}{p}$. But with the choice of Ω, $\frac{\Omega}{p} > \frac{\Omega+B/2}{p+1} = G$, a contradiction.

Then $|S'| = p$, and $\frac{\Omega+p(S')}{1+p} = G = \frac{\Omega+B/2}{1+p}$, so $p(S') = B/2$. The corresponding set A' of p elements shows that the answer to the equipartition instance is yes. □

We now show that S-TOP-1D is solvable in pseudo-polynomial time, using dynamic programming.

Theorem 4.2. S-TOP-1D *is solvable in time $O(n^3 p_{max})$ with n the number of sounds and p_{max} the maximum value of a sound in N.*

Proof. To solve the problem, we build a truth table TAB such that $\text{TAB}(i,j,k) = true$ if and only if there exists a subset of the first i sounds that sums to j with exactly k sounds (i.e. of size k). TAB is defined for $i = 1, \ldots, n$, $j = 0, \ldots, \sum_{i=1}^{n} p_i$, and $k = 0, \ldots, n$.

Note that as $\sum_{i=1}^{n} p_i \leq np_{max}$, the size of the table is $O(n^3 p_{max})$.

We initialize TAB when $i = 1$ or $j = 0$ or $k = 0$, using the following four different cases:

- $TAB(i,0,0) = true$ for all $i \in \{1, \ldots, n\}$
- $TAB(i,0,k) = false$ for all $i \in \{1, \ldots, n\}$, all $k \in \{1, \ldots, n\}$.
- $TAB(i,j,0) = false$ for all $i \in \{1, \ldots, n\}$, all $j > 0$.
- If $j \geq 1$ and $k \geq 1$, $TAB(1,j,k) = true$ if $j = p_1$ and $k = 1$, $false$ otherwise.

Then, for $i \geq 2$, $j \geq 1$ and $k \geq 1$ we compute $TAB(i,j,k)$ using the recursion:

$$\begin{cases} TAB(i,j,k) = & TAB(i-1,j,k) \;||\; TAB(i-1,j-p_i,k-1) & \text{if } p_i \leq j \\ & TAB(i-1,j,k) & \text{otherwise} \end{cases}$$

Indeed, to get a sum of j with exactly k sounds, we can either take the sound s_i ($TAB(i-1,j-p_i,k-1)$) if $p_i \leq j$, or not take s_i ($TAB(i-1,j,k)$).

Once all the TAB is filled with respect to the previous rules, by simply browsing through the TAB, and finding the cell where $|\frac{j}{k} - G|$ is minimal, we get the value of an optimal solution. Then a standard backward procedure allows to recover the solution. As each cell is filled in constant time, the overall complexity of $O(n^3 p_{max})$. □

Now, we consider approximation algorithms. The problem consisting of knowing whether the optimal value is 0 or not is **NP**-hard (Theorem 4.1). Thus it is not possible to get any c-approximation polynomial time algorithm, as any such algorithm allows to detect when the optimal value is 0 or not. Thus, an additive term is mandatory in the approximation result. Here, we show that by standard rounding technique, one can use the previous dynamic algorithm to get a polytime algorithm with only *additive* error - and no multiplicative error. More precisely, we have the following result (where OPT is the optimal value and $p_{max} = \max p_i$).

Theorem 4.3. *For any $\epsilon > 0$,* S-TOP-1D *admits an algorithm which outputs a solution of value at most $OPT + \epsilon p_{max}$ in time $O(\frac{n^3}{\epsilon})$.*

Proof. Let $\epsilon > 0$. Given an instance I of S-TOP-1D, we construct an instance I' of the same problem by rounding the p_i: for all sound s_i in N, $p'_i = \lfloor \frac{p_i}{\lambda} \rfloor$, where $\lambda = \frac{p_{max}\epsilon}{2}$. We also round the target: $G' = \frac{G}{\lambda}$. The algorithm simply computes an optimal solution S' on I' using the dynamic programming algorithm and outputs it. Note that the complexity of the algorithm is $O(n^3 p'_{max})$, i.e., $O(n^3 p_{max}/\lambda) = n^3/\epsilon)$ as claimed.

Let S^* be an optimal solution for the original instance I. The remainder of the proof is to show that

$$d(S', G) \leq d(S^*, G) + 2\lambda = d(S^*, G) + \epsilon p_{max} \tag{4.1}$$

For any subset of sounds S, let $m(S) = \frac{p(S)}{|S|}$ and $m'(S) = \frac{\sum_{i \in S} p'_i}{|S|}$. From the inequality $\frac{p_i}{\lambda} \geq \lfloor \frac{p_i}{\lambda} \rfloor \geq \frac{p_i}{\lambda} - 1$, we get $\frac{m(S)}{\lambda} \geq m'(S) \geq \frac{m(S)}{\lambda} - 1$. From this, we immediately deduce:

$$m(S) \geq \lambda m'(S) \geq m(S) - \lambda \tag{4.2}$$

$$\lambda m'(S) \leq m(S) \leq \lambda m'(S) + \lambda \tag{4.3}$$

Now, let us look at $d(S, G) = |m(S) - G|$:

- if $m(S) - G \geq 0$ then $|m(S) - G| = m(S) - G \leq \lambda m'(S) + \lambda - G = \lambda(m'(S) - \frac{G}{\lambda}) + \lambda$
- if $m(S) - G < 0$ then $|m(S) - G| = G - m(S) \leq G - \lambda m'(S) = \lambda(\frac{G}{\lambda} - m'(S))$.

Then, in both cases we have:

$$|m(S) - G| \leq \lambda \left| m'(S) - \frac{G}{\lambda} \right| + \lambda \tag{4.4}$$

Regarding $|m'(S) - G'|$, we get:

- if $m'(S) - \frac{G}{\lambda} \geq 0$ then $|m'(S) - \frac{G}{\lambda}| = m'(S) - \frac{G}{\lambda} = \frac{\lambda m'(S) - G}{\lambda} \leq \frac{m(S) - G}{\lambda}$.
- if $m'(S) - \frac{G}{\lambda} < 0$ then $|m'(S) - \frac{G}{\lambda}| = \frac{G}{\lambda} - m'(S) \leq \frac{G}{\lambda} - (\frac{m(S)}{\lambda} - 1) = \frac{G - m(S)}{\lambda} + 1$

Then, we have:

$$|m'(S) - \frac{G}{\lambda}| \leq \frac{|m(S) - G|}{\lambda} + 1 \tag{4.5}$$

Thus, using Equations (4.4) and (4.5) we can bound the loss incurred by the scaling technique when considering S' as a solution of the initial distance:

$$\begin{aligned} |m(S') - G| - |m(S^*) - G| &\leq \lambda \left| m'(S') - \frac{G}{\lambda} \right| - \lambda \left| m'(S^*) - \frac{G}{\lambda} \right| + 2\lambda \quad (4.6) \\ &= \lambda(\left| m'(S') - \frac{G}{\lambda} \right| - \left| m'(S^*) - \frac{G}{\lambda} \right|) + 2\lambda \quad (4.7) \end{aligned}$$

By optimality of S' on I', we have $|m'(S') - \frac{G}{\lambda}| - |m'(S^*) - \frac{G}{\lambda}| \leq 0$.
Thus we get $|m(S') - G| - |m(S^*) - G| \leq 2\lambda$, ie, Equation (4.1). □

4.5 The static target-based orchestration problem

We now consider the general static orchestration problem S-TOP. We recall that, with respect to S-TOP-1D, in S-TOP:

- Each sound s_i has M evaluations $p_{ij}, j = 1, \ldots, M$. We will denote $\vec{p_i} = (p_{i1}, \ldots, p_{iM})$ the vector of evaluations of sound s_i.
- The set N of sounds is partitioned into Z sets $N_z, z = 1, \ldots, Z$. N_z, of size n_z, is the set of sounds played with a given instrument. The number of sounds chosen in N_z cannot exceed L_z, while the global number of sounds cannot exceed L.

S-TOP is of course **NP**-hard, as it generalize S-TOP-1D. In this section, we show how the dynamic programming algorithm and the approximation algorithm generalizes to S-TOP. We first start by the dynamic programming algorithm, showing that S-TOP remains pseudo-polynomial when the dimension M is fixed.

Theorem 4.4. S-TOP *is solvable in time* $O(L^2 n^{M+1} p_{max}^M)$.

Proof. The algorithm proceeds instrument by instrument.

Let $\vec{v}$ be a vector of M positive integers. For each instrument z, we define the table $\textsc{Tab}_z(i, \vec{v}, k, \ell)$ which equals *true* if there is a subset S of sounds such that:

- $|S| = k$;
- $\vec{p}(S) = \vec{v}$ (ie, on each dimension j, $\sum_{s_i \in S} p_{ij} = v_j$);
- S takes only sounds among the instruments up to $z-1$, or in the first i sounds of N_z.
- $|S \cap N_z| = \ell$.

We define this value for $i = 0, \ldots, n_z$, for any vector in $\{0, np_{max}\}^M$ (as no value can exceed np_{max} on a dimension), $k = 0, \ldots, L$ and $\ell = 0, \ldots, L_z$.

Note that the size of TAB_z is $O(n_z(np_{max})^M L^2)$ (as $L_z \leq L$), so the total size of these tabs is $O(n(np_{max})^M L^2)$.

For $i \geq 1$, we use a recurrence relation similar to the one in the restricted case of S-TOP-1D. Let s_i^z be the i^{st} sound of N_z, with evaluations $\vec{p_i^z}$. Then:

- If $k = 0$ or $\ell = 0$, or if there exists $j \in \{1, \ldots, M\}$ such that $p_{ij}^z > \vec{v}_j$, then we cannot take s_i^z, so $\textsc{Tab}_z(i, \vec{v}, k, l_z) = \textsc{Tab}_z(i-1, \vec{v}, k, l_z)$.
- Otherwise, we can either take s_i^z or not, meaning that

$$\textsc{Tab}_z(i, \vec{v}, k, \ell) = \textsc{Tab}_z(i-1, \vec{v}, k, \ell) \vee \textsc{Tab}_z(i-1, \vec{v} - \vec{p_i^z}, k-1, \ell-1)$$

Now we consider initialization, when $i = 0$. When $z = 1$ (first instrument), we have $\textsc{Tab}_1(0, \vec{v}, k, \ell) = true$ iff $k = \ell = 0$ and $\vec{v}_j = 0$ for all j.

For $z \geq 2$, the initialization of $\textsc{Tab}_z$ uses the values of $\textsc{Tab}_{z-1}$. If $\ell > 0$ then $\textsc{Tab}_z(0, \vec{v}, k, \ell)$ is clearly false. Otherwise, it is true iff we have one true value in the table $\textsc{Tab}_{z-1}$ for one number of sounds in N_{z-1}:

$$\textsc{Tab}_z(0, \vec{v}, k, 0) = \vee_{\ell=0}^{L_{z-1}} \textsc{Tab}_{z-1}(n_{z-1}, \vec{v}, k, \ell)$$

Once the tables $\textsc{Tab}_z$ have been filled, we look at the last table $\textsc{Tab}_z$ with $i = n_Z$. We look at all the couples $(\vec{v}, k)$, with $k \in \{1, \ldots, L\}$, for which there exists $\ell \leq L_z$ such that $\textsc{Tab}_z(n_Z, \vec{v}, k, \ell) = true$. This corresponds to a solution S of value $d(S, G) = \sum_{j=1}^{M} \left| \frac{\vec{v}_j}{k} - G_j \right|$. This way we can find the optimal value, and an optimal solution with standard backward procedure. □

Now, let us consider approximation algorithms, and see how Theorem 4.3 generalizes. Note that the algorithm in Theorem 4.5 runs in polynomial time when M is a fixed constant (as $L \leq n$).

Theorem 4.5. *For any $\epsilon > 0$,* S-TOP *admits an algorithm which outputs a solution of value at most $OPT + \epsilon p_{max}$ in time $O\left(L^2 n \left(\frac{2nM}{\epsilon}\right)^M\right)$.*

Proof. Let $\epsilon > 0$. Given an instance I, as for S-TOP-1D we create an instance I' by rounding the valuations: $p'_{ij} = \lfloor \frac{p_{ij}}{\lambda} \rfloor$, where $\lambda > 0$. We also round the target: the new target G' is defined as $G'_j = \frac{G_j}{\lambda}$. We compute an optimal solution on I' using the dynamic programming algorithm, and output it.

Let S' be an optimal solution on I', and S^* be an optimal solution on I. Following the proof on Theorem 4.3, we get that on each dimension j:

$$\left|\frac{p_j(S')}{|S'|} - G_j\right| - \left|\frac{p_j(S^*)}{|S^*|} - G_j\right| \leq \lambda \left(\left|\frac{p'_j(S')}{|S'|} - \frac{G_j}{\lambda}\right| - \left|\frac{p'_j(S^*)}{|S^*|} - \frac{G_j}{\lambda}\right|\right) + 2\lambda \quad (4.8)$$

Thus, by summing on the dimensions, we have:

$$d(S', G) - d(S^*, G) \leq \lambda \left(d'(S', G') - d'(S^*, G')\right) + 2M\lambda$$

As S' is optimal on I', we deduce $d(S', G) \leq d(S^*, G) + 2M\lambda$. By choosing $\lambda = \frac{\epsilon p_{max}}{2M}$ we get the claimed bound on the value of the solution.

As $p'_{max} = \max_{i,j} p'_{i,j} \leq p_{max}/\lambda = 2M/\epsilon$, the algorithm has complexity $O\left(L^2 n \left(\frac{2nM}{\epsilon}\right)^M\right)$. □

4.6 The dynamic target-based orchestration problem

We now consider the problem D-TOP, where we have a sequence of targets, and both orchestration cost and transition cost. We show that the static DP procedures combined with a shortest path formulation of the problem capturing temporality, allow to solve the problem in pseudo-polynomial time (when the number of dimensions is fixed).

As for the static case, we first focus on the problem D-TOP-1D which is the restriction of D-TOP when there is only one instrument, no orchestration constraint, and one dimension. We tackle then the general problem. Of course, as generalizations of S-TOP-1D, D-TOP-1D and D-TOP are **NP**-hard.

4.6.1 D-TOP-1D

Now, consider T time steps, a sequence $G^1, \ldots, G^T$ of targets, and we seek a sequence $S_1, \ldots, S_T$ minimizing $\sum_{t=1}^{T} d(S_t, G^t) + \sum_{t=2}^{T} C_t \; d(S_t, S_{t-1})$. The problem is pseudo-polynomial, as stated in the following theorem.
Note that we consider for the theorems and proofs that C_t is fixed to 1.

Theorem 4.6. D-TOP-1D *is solvable in time* $O(n^3 p_{max} + T n^2 p_{max}^2)$.

Proof. We first apply the dynamic procedure devised for the static case to get for each time step t a table TAB such that $\textsc{Tab}_t(i,j,k) = true$ if and only if there exists a subset of the first i sounds that sums to j with exactly k sounds (i.e. of size k). This is done in time $O(n^3 p_{max})$. Using the backtrack procedure, we get a set of sounds for each cell with value true. Note that the distance $d(S,S')$ between two sounds S and S' are fully determined by $p(S)$, $p(S')$ and $|S|$, $|S'|$. Thus we can keep only one solution per cell.

Let $\Omega = (S^1, \ldots, S^\ell)$ be the set of solutions corresponding to true cells for $i = n$. Each such solution corresponds to a sum j and a size k. Note that there are at most np_{max} of them. Now, the goal is to select a sequence $(S_1, \ldots, S_T)$, where $S_t \in \Omega$, in order to minimize the global cost. This can be done by computing a shortest path in a directed acyclic graph.

We define a graph G composed with T layers: in each layer t we have a copy of Ω. There is an arc between any solution S_t^q of layer t to any solution S_{t+1}^m of layer $t+1$. This arc has weight $d(S_{t+1}^m, G^{t+1}) + d(S_t^q, S_{t+1}^m)$, corresponding to the orchestral cost of S_{t+1}^m and its transition cost from S_t^q.

We also add a source s, with an arc to any solution S_1^q of the first layer, with weight $d(S_1^q, G^1)$, and a sink t with an arc with weight 0 from any vertex of the last layer to t.

From the construction, it is clear that $s-t$-path correspond to (feasible) sequence of solutions, and the weight of each such path is the cost of the corresponding sequence.

As there are $T|\Omega|+2$ vertices, and each vertex has at most $|\Omega|$ predecessors, the shortest path can be computed in time $O(T|\Omega|^2)$. As $|\Omega| \leq np_{max}$, the complexity of computing a shortest path is $O(Tn^2 p_{max}^2)$. □

We now show that the scaling technique applies also to the dynamic setting. We get a polynomial time algorithm with additive error $\epsilon p_{max}(1 + C_{max})$, where $C_{max} = \max_{t=1}^{T-1} C_t$.

Theorem 4.7. *For any $\epsilon > 0$,* D-TOP-1D *admits an algorithm which outputs a solution of value at most $OPT + \epsilon p_{max}(1 + C_{max})$ in time $O\left(n^3 \frac{T}{\epsilon} + n^2 \frac{T^3}{\epsilon^2}\right)$.*

Proof. Let $\epsilon > 0$. Given an instance I of D-TOP-1D, we construct an instance I' of the same problem by rounding the p_i and the targets G_t in the same way as in the static version of the problem: $p'_i = \left\lfloor \frac{p_i}{\lambda} \right\rfloor$ and $G'_t = \frac{G_t}{\lambda}$. We fix $\lambda = \frac{\epsilon p_{max}}{2T}$.

The algorithm outputs an optimal solution $S' = (S'_1, \ldots, S'_T)$ on I', using the DP algorithm.

Let $S^* = (S_1^*, \ldots, S_T^*)$ be an optimal solution on I. We bound the loss incurred by the scaling, ie we upper bound $f(S') - f(S^*)$, both in the orchestration cost and in the transition cost.

For the orchestration cost, we can use the same inequality as in the static case (see Equation 4.4) and get that, for any t:

$$|m(S'_t) - G_t| - |m(S^*_t) - G_t| \leq \lambda(|m'(S'_t) - G'_t| - |m'(S^*) - G'_t|) + 2\lambda \tag{4.9}$$

Let us now look at the transition costs. Let S_1 and S_2 be two subsets of sounds, and let us bound $|m(S_2) - m(S_1)|$.

- if $m(S_1) - m(S_2) > 0$, then $|m(S_1) - m(S_2)| = m(S_1) - m(S_2) \leq \lambda m'(S_1) + \lambda - \lambda m(S'_2)$.
- if not, we have: $|m(S_1) - m(S_2)| = -m(S_1) + m(S_2) \leq -\lambda m'(S_1) + \lambda + \lambda m(S'_2)$.

Thus we have:

$$|m(S_1) - m(S_2)| \leq \lambda(|m'(S_1) - m'(S_2)|) + \lambda \tag{4.10}$$

With a similar argument, we get that:

$$|m'(S_1) - m'(S_2)| \leq \frac{|m(S_1) - m(S_2)|}{\lambda} + 1 \tag{4.11}$$

Combining Equations (4.10) and (4.11), we get that for any $t < T$:

$$|m(S'_{t+1}) - m(S'_t)| - |m(S^*_{t+1}) - m(S^*_t)| \leq \lambda(|m'(S'_{t+1}) - m'(S'_t)| - |m'(S^*_{t+1}) - m'(S^*_t)|) + 2\lambda \tag{4.12}$$

By summing over t equations (4.9) and (4.12), we obtain:

$$\begin{aligned} f(S') - f(S^*) &\leq \lambda(f'(S') - f'(S^*)) + 2\lambda T + 2\lambda(T-1)C_{max} \\ &\leq \lambda(f'(S') - f'(S^*)) + 2\lambda T(1 + C_{max}) \end{aligned} \tag{4.13}$$

where $C_{max} = \max_{t=1}^{T-1} C_t$, and f' is the objective function on I'. As S' is optimal on I', with $\lambda = \frac{\epsilon p_{max}}{2T}$ we have that $f(S) \leq f(S^*) + \epsilon p_{max}(1 + C_{max})$.

In I', $p'_{max} \leq p_{max}/\lambda = 2T/\epsilon$, so the running time follows. □

4.6.2 D-TOP

In this section we go back to the general problem D-TOP. We can use similar arguments as the ones for D-TOP-1D and get the following results, showing pseudo-polynomiality when M is a fixed constant (Theorem 4.8), and a polynomial time (when M is a fixed constant) approximation algorithm with additive error (Theorem 4.9).

Theorem 4.8. D-TOP *is solvable in time* $O(TL^2(2nMT/\epsilon)^{2M})$.

Proof. As for S-TOP-1D, we first use the static dynamic programming algorithm to fill truth tables $\text{TAB}_z(i, \vec{v}, k, \ell)$. This takes times $O(L^2 n^{M+1} p_{max}^M)$. We are interested in table TAB_Z, ($z = Z$, all instruments are considered), for $i = n_Z$ (all sounds are considered). For each $(\vec{v}, k)$ we know if there exists a feasible (static) solution of size k and with evaluations equal to $\vec{v}$ (by looking at all possible values of ℓ).

Let Ω be a set of such feasible solutions, one for each $(\vec{v}, k)$ with such a solution. Ω has size at most $O((np_{max})^M L)$. The shortest path procedure takes time $O(T\Omega^2)$, ie $O(TL^2(np_{max})^{2M})$.

The global complexity is $O(L^2 n^{M+1} p_{max}^M) + O(TL^2(np_{max})^{2M})$, which is $O(TL^2(np_{max})^{2M})$. □

Theorem 4.9. *For any $\epsilon > 0$,* D-TOP *admits an algorithm which outputs a solution of value at most $OPT + \epsilon p_{max}(1 + C_{max})$ in time $O(TL^2(np_{max})^{2M})$.*

Proof. As for the case of 1 dimension, we scale the profits and target to get an instance I', and use the DP algorithm to get an optimal solution S' on I'. Then we get a loss of at most $2\lambda T(1 + C_{max})$ on each dimension (see Equation (4.13). As the global cost is the sum of cost on each dimension, the global loss is upper bounded by $2M\lambda T(1 + C_{max})$. By fixing $\lambda = \frac{\epsilon p_{max}}{2MT}$ we get an additive error $\epsilon p_{max}(1 + C_{max})$. As in I' $p'_{max} \leq p_{max}/\lambda = 2MT/\epsilon$, the running time follows. □

4.7 Towards practical solutions: some experimental results

We conduct some experiments to solve the orchestration problems on some specific target sounds, both in the static and in the dynamic frameworks. Our experiments use the database *TinySOL* used in the software *Orchidea* (see Section 4.7.1), containing around 1500 sounds. We compare our results with the solutions output by the evolutionary algorithm used in the software *Orchidea*.

We first consider the static case (Section 4.7.2). While a pseudo-polynomial time algorithm has been obtained (Theorem 4.4), the number of dimensions of sounds in the database is $M = 1024$, which makes DP inefficient as its time complexity is exponential in M. Rather, we propose an ILP formulation of the problem, that allows to solve the problem efficiently on the *TinySol* database.

For the dynamic case (Section 4.7.3), an ILP formulation turns out to be way too slow, even for a few time steps. To solve the problem, we propose a heuristic based on the shortest path formulation used in Theorems 4.6 and 4.8.

4.7.1 Dataset

Database. The database of orchestral sounds used in our experiments is called

TinySOL and is distributed with the software *Orchidea*. TinySOL is a streamlined version of the Studio On Line (SOL) database created by IRCAM, that contains over 117,000 instrument samples, including extended techniques and contemporary playing styles. TinySOL, on the other hand, contains only 1,529 samples from 12 instruments. The instruments come from different orchestral families: strings, woodwinds, and brass. Each sample is one instrument playing a single note in the *ordinario* playing style, with one of three dynamics: *pp*, *mf*, or *ff* (for example `Flute-C4-pp` or `Clarinet-D5-mf`).

Each sample (sound) in the database is described by its value on $M = 1024$ dimensions, called its *spectrum* (related to the spectrum analysis of the sound).

Orchestra. In the experiments, we always use the same orchestra: it contains 23 instruments: 2 Flutes (Fl), 2 Oboes (Ob), 2 Clarinets in Bb (ClBb), 2 Bassoon (Bn), 2 French Horns (Hn), 2 Trumpets in C (TpC), 2 Trombones (Tbn), 1 Bass Tuba (BTb), 2 Violins (Vn), 2 Violas (Va), 2 Cellos (Vc) and 2 Contrabasses (Cb). This specifies the orchestration constraints of our problems.

Targets. There are two types of target sounds:

- Static targets: we conducted experiments with real instrument sounds such as the note of a clarinet or a wind harp, with extract of orchestral sounds, and with some real life sounds such as a boat docking, a girl screaming, or a car horning.
- Dynamic targets: we conducted experiments with real instruments sounds, and real life sounds such as the sound of drops falling.

Note that the target sounds are listenable at http://www.orch-idea.org/.

4.7.2 Static case

We first focus on the static case. As explained above, with $M > 1000$ dimensions, DP is impossible to use, even with rounding, as there is a factor n^M in the time complexity of the algorithms. We give an ILP formulation of the problem, and use it to solve the static orchestration problems on some real target sounds.

ILP formulation

For each sound s_i we consider a binary variable x_i equal to 1 iff s_i is taken in the solution S. We have mainly to linearize the objective function $d(S, G) = \sum_{j=1}^{M} \left| \frac{p_j(S)}{|S|} - G_j \right|$. To do so, we introduce a variable δ_j which will correspond to

the cost $|\frac{p_j(S)}{|S|} - G_j|$ on dimension j. We have the following formulation with $n + M$ variables (n binary and M continuous) and a number of constraints of the same order (we remind that L_z is the number of available instruments of type z in the orchestra, and L an upper bound on the number of selected sounds).

$$
\left\{
\begin{array}{l}
\min \sum_{j=1}^{M} \delta_j \\
\quad s.t. \left|
\begin{array}{lcll}
\delta_j \sum_{i=1}^{n} x_i & \geq & \sum_{i=1}^{n} p_{ij} x_i - G_j \sum_{i=1}^{n} x_i & \forall j \in \{1, ..., M\} \\
\delta_j \sum_{i=1}^{n} x_i & \geq & G_j \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} p_{ij} x_i & \forall j \in \{1, ..., M\} \\
\sum_{i : s_i \in N_z} x_i & \leq & L_z & \forall z = 1, \ldots, Z \\
\sum_{i=1}^{n} x_i & \leq & L & \\
x_i \in \{0, 1\} & & & , \forall i = 1, \ldots, n
\end{array}
\right.
\end{array}
\right.
$$

However, this is not yet a linear formulation as the left hand side of the first two families of constraints is quadratic. A first way to deal with it is to introduce nM variables y_{ij}, and force them to be equal to $\delta_j x_i$ by putting the following set of constraints (where R is a sufficiently large positive number):

- $y_{ij} \leq \delta_j$
- $y_{ij} \geq \delta_j - R(1 - x_i)$
- $y_{ij} \leq R x_i$
- $y_{ij} \geq 0$

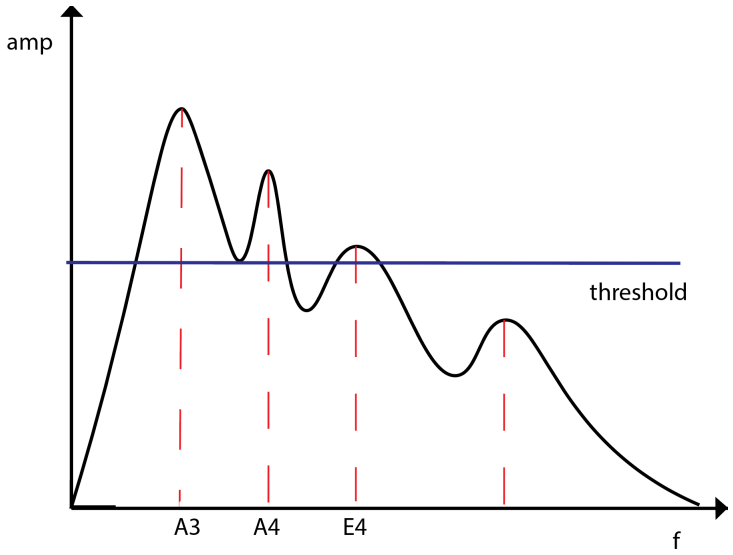
This adds nM variables and $\Theta(nM)$ constraints.

A second way to linearize is to fix the number of sounds $\sum_{i=1}^{n} x_i$ selected, by adding a constraint $\sum_{i=1}^{n} x_i = C$ (and use C in the first two constraints), and to solve several ILPs, one for each value of C, from 1 to the total number L of instruments in the orchestra, i.e. $L = 23$ in our case. This does not increase the size of the formulation but we have to solve several ILPs (in fact, L of them).

We use this latter solution in our experiments as it turned out to be much more efficient, avoiding the introduction of $nM > 10^6$ variables y_{ij}.

Results on some target sounds

The aim of these tests were (1) to test whether the ILP formulation is efficient to solve real-world instances or not, and (2) to compare and evaluate the ILP model with the current *Orchidea* software which uses an evolutionary algorithm.
In the software two specific ingredients are used:



4.10: Representation of the harmonic filter

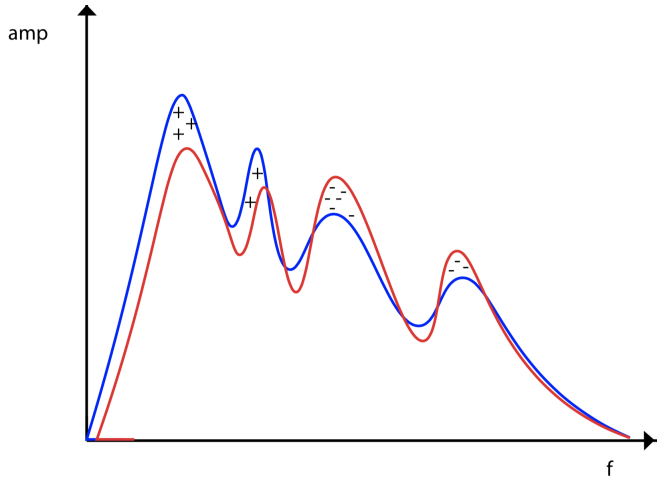
(1) An **harmonic filter**. It is a filter that reduces the set of sounds in order to apply algorithms on smaller database and running faster. Looking at the spectral envelope of the static target sound, it is possible to extract the fundamental harmonic (the first partial) and its corresponding note as well as the other present harmonics and notes. The partials and their specific notes are represented by peaks in the spectral envelope. The partial filtering, i.e. the harmonic filter, uses then a threshold applied on the spectral envelop and only keep in the database the notes corresponding to the peaks above the threshold.

Figure 4.10 illustrates the harmonic filter. In the example, the peaks above the threshold have corresponding notes/partials A3, A4 and E4. This implies that in the database will be present only sounds with the notes A3, A4 and E4, removing all the other notes for all instrument and reducing considerably the size of the database to use.

(2) **Upper and lower penalties** The distance measure $d(S, G)$ used in the software is actually not symmetric. For musical purpose, it is less problematic on a

dimension to be lower than the target than higher than it regarding the amplitude of the spectral envelopes of both sounds. This induces a slight modification in the objective function.

Figure 4.11 is a representation of the penalties. The blue curve represents the spectral envelop of the target sound and the red one the solution envelop. Positive penalties (+) are applied when the solution curve peaks are below the target ones and negative penalties (-) when they are above the target spectral envelop.

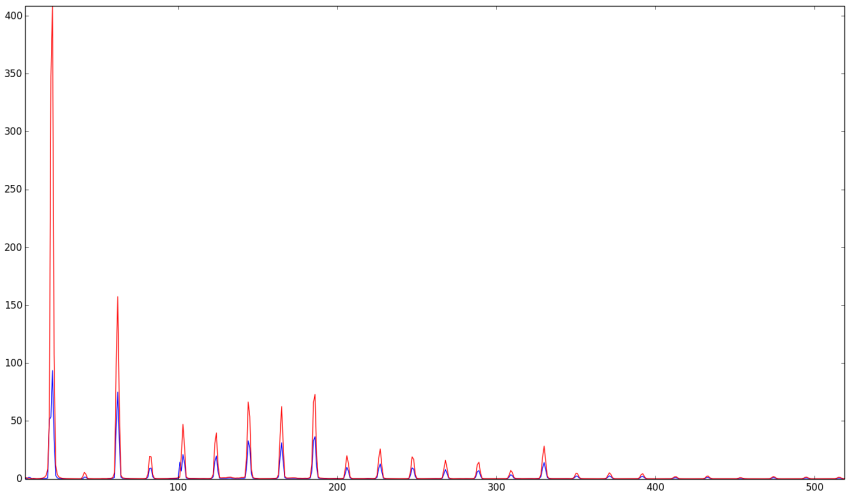


4.11: Representation of the penalties with the blue curve the spectral envelop of the target sound and the red one the solution envelop

To make a fair comparison between with the solutions computed by the *Orchidea* software and by the ILP, we chose to include these two ingredients in ILP experiments, i.e., the calculations were applied on the same databases and with the same penalty factors.

For illustrative purpose Figure 4.12 represents the target Clarinet A3 (in red) and its optimal solution (in blue). The horizontal axis represents frequencies, and the vertical axis the value in the spectral analysis. We see that the solution produces a sound which essentially uses the same frequencies.

In Table 4.13 are presented our results on some static targets sounds. The ILP was able to solve optimally all instances. Concerning the solutions themselves, it is noticeable that for some instances the two algorithms (note that **Evolutionnary** refers to the algorithm used in *Orchidea*) chose disjoint sets of sounds. Concerning the values, ILP gives optimal solutions so the values are of course better, but we note that the improvements with respect to the solutions computed by the evolutionary algorithm are significant. These elements indicate that an LP approach is certainly interesting for the static orchestration problem. Even if the solution were better



4.12: Representation of the solution with the static target Clarinet A3

according to their values (calculated on the objective function of the ILP), some solutions of the evolutionary algorithm seemed to "sound" better. This is due to the hardness of the dissimilarity measure choice and to the ILP formulation being a simpler modelization than the evolutionary one.

target sound	ILP	Evolutionary
Archeos Bell	508	2803
Bass Clarinet	1114	6074
Beethoven Chord2	490	2401
Boat Docking	463	2363
Car Horn	1017	3968
Girl Scream	2651	3581
Winchester Bell	897	2302
Wind Harp	736	1656
YaBn mul PG ST12	757	5660

4.13: Results of the ILP and the Evolutionary algorithms on some static target sounds.

In practice, there is a significant difference between the model and the reality. To overcome this difference and have results closer to real world applications, the script **dbgen** was developed in order to analyze the results. **dbgen** is a script that, given a sound or a set of sounds generates its feature description, i.e. it creates a

value vector of dimension the size of the chosen feature type (here 1024 as it is the spectrum feature that was used).
In the following results (see 4.14), a sound was first generated based on the spectral description of the solution and then **dbgen** was applied to it, generating again a spectral description of the sound on which the distances were computed.

target sound	ILP post dbgen	Evolutionary post dbgen
Archeos Bell	3323	6436
Bass Clarinet	4785	14153
Beethoven Chord2	1145	2513
Boat Docking	1470	3922
Car Horn	1674	9414
Girl Scream	4096	4555
Winchester Bell	3097	3434
Wind Harp	3101	6288
YaBn mul PG ST12	1304	8255

4.14: Results of the ILP and the Evolutionary algorithms on some static target sounds using **dbgen**.

Concerning the values presented in 4.14, ILP post dbgen still gives solutions with better values but it is interesting to notice that for most of the target sounds the solutions of the evolutionary algorithm and the ILP ones are much closer than in the previous tab because of the fact that the evolutionary algorithm was calibrated using dgben.

4.7.3 Dynamic Case

We give an ILP formulation of the assisted orchestration problem in its dynamic version. The idea behind the formulation is to emphasize the similarities and links with the multistage framework.
For the dynamic case, we now need to minimize $\sum_{t=1}^{T} d(S_t, G^t) + \sum_{t=1}^{T-1} C_t d(S_t, S_{t+1})$.

As said before, the function can be seen as a multistage framework function.
Indeed, one needs to seek a trade-off between:

- the quality of the solution: on each of the T time steps of the time horizon is computed a distance between the solution and the target sound
- the stability of the solution for consecutive time steps: for all consecutive time steps is computed the distance between the consecutive time steps solutions.

Note that in our formulation of the problem, the distance to the target sound and the distance between consecutive time steps solutions is the same. The fact of using the same distance for the two ingredients of the objective function differs with the approach used in *Orchidea* software. Indeed, a lot of different functions could be considered in order to evaluate the stability of a given solution to the problem. It could be interesting for example to seek a solution that minimizes: the instruments modifications (if an instrument is played at a time step t, a bonus is obtained for taking the same instrument at the next time step), the note modifications (keeping the same note on any instrument) or a combination of the two. This is interesting if a composer is giving importance to the continuity of its solutions (see Figure 4.8) which is represented in the continuity model. Our choice of using the same distance measure gives us the possibility to model the problem as an ILP problem and not as a multi-objective problem, the two elements of our function being comparable. Note also that in some cases, if the target pitch, i.e. its notes, change a lot between two consecutive steps, it is not relevant to keep the same note as the result would be certainly far from the target sound.

Let us now focus on the ILP formulation. We need now to consider nT binary variables x_{it}, indicating whether s_i is taken in S_t or not. To linearize the objective function, we can introduce $\Theta(nT)$ variables: δ_{jt} which will be equal to $|p_j(S_t)/|S_t| - G_j|$, and α_{jt} which will be equal to $|p_j(S_t)/|S_t| - p_j(S_{t+1})/|S_{t+1}||$. Then the objective is to minimize $\sum_{t=1}^{T}\sum_{j=1}^{M}\delta_{jt} + \sum_{t=1}^{T-1}\sum_{j=1}^{M}\alpha_{jt}$, and we put the constraints:

- $\delta_{jt} \geq \frac{p_j(S_t)}{|S_t|} - G_{jt}$ and $\delta_{jt} \geq G_{jt} - \frac{p_j(S_t)}{|S_t|}$;
- $\alpha_{jt} \geq \frac{p_j(S_t)}{|S_t|} - \frac{p_j(S_{t+1})}{|S_{t+1}|}$ and $\alpha_{jt} \geq \frac{p_j(S_{t+1})}{|S_{t+1}|} - \frac{p_j(S_t)}{|S_t|}$.

In order to linearize these constraints, as in the static case we can:

- Either add some extra variables, to avoid quadratic terms in the constraints;
- Or fix the number of selected sounds at *each* time step, and solve the ILP for all the possible combinations of sizes of solutions S_t.

In the first case we need to introduce $\Omega(nMT)$ variables to linearize the constraints. In the second case we need to solve L^T ILP, where L is the maximum number of sounds selected at one time step, $L = 23$ in our orchestra.

Both approaches were not able to solve our instances (recall that n and M are of order of 1000), as soon as $T > 2$.

The number of variables of the linear program is very large, in fact there are: NT variables x,N^2T variables y,MT variables δ,$M(T-1)$ variables γ, NMT variables α, $N(T-1)$ variables β, $N(T-1)$ variables λ, $N^2(T-1)M$ variables θ.

In practice, we apply the harmonic filter on the TinySol dataset and apply the algorithm on around 1400 sounds, i.e. $N = 1400$, of dimension M equal to 1024 and it is possible to have around 20 time steps, i.e. $T = 20$. The number of variables is thus extremely large. For exemple, only for the θ variable we have $1400^2 \times 20 \times 1024 = 40140800000 = 40$ billion θ variables. So it is impossible to run the program, even on powerful engines.

However, to verify it, we run the algorithm on a very restrictive instance with only $T = 2$ time steps, a hard filter on the dataset, allowing only one note to be picken and thus around $N = 250$ sounds, a small metric called moments with $M = 4$. Even is this special case, the number of variables is quite large with around $2 * 250^2 * 4 = 500000$ θ variables.

To overcome this difficulty, we rather propose in the following a heuristic, called *SPH*, the idea of which is to combine (1) the fact that we can solve the static problems (as explained in the previous section) (2) a shortest path formulation that allows to efficiently combine solutions found in different time steps.

SPH: Shortest Path Heuristic

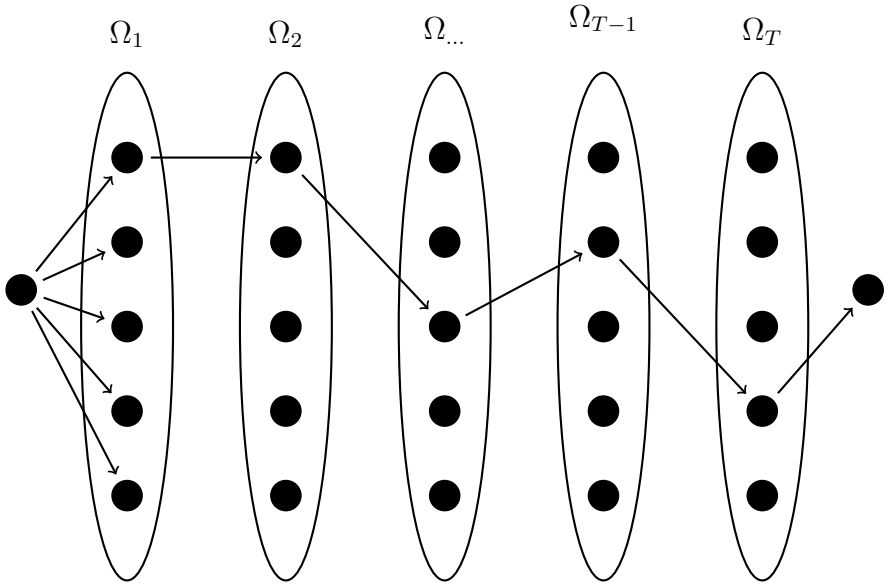
In Theorems 4.6 and 4.8, the principle was to use DP to produce a set of solutions at each time step, and then to solve a shortest path problem to find the best combinations of solutions. As explained previously, with $M = 1024$ dimension we cannot use DP to produce solutions at each time step, but the shortest path idea is still interesting. Then, the principle of SPH (shortest path heuristic) is (see Figure 4.15):

- To compute, for each time step t, a set Ω_t of feasible solutions;
- Then to find the best tuple $(S_1, \ldots, S_T) \in \Omega_1 \times \cdots \times \Omega_T$, by solving a shortest path problem on a DAG on vertex set $\cup\Omega_t$ (and two vertices s and t).

The second step can be done very efficiently, and the problem boils down to finding interesting sets of solutions (independently) for each time step.
Recall that, for the static case, we solve the instances using an ILP for each number of sounds i, from 1 to the maximum number of sounds L. This produces an interesting bunch of L static solutions. Our heuristic SPH is precisely to use for Ω_t this set A_t of L feasible solutions. Once obtained these static solution, the shortest path problem is on LT nodes (plus s and t).
Note that overall the computation time is (nearly) linear in T.

Results on some dynamic target sounds

As previously, we compare SPH with the solution output by the current software *Orchidea*.



4.15: Representation of the SPH algorithm for the D-TOP

Also, we were interested in evaluating the choice of sets $\Omega_t = A_t$ in SPH. To do so, we also consider the set E_t of solutions output by the evolutionary algorithm of *Orchidea* on target at time t. We consider in the experiments 3 possible choices:

- $\Omega_t = A_t$ (SPH);
- $\Omega_t = A_t \cup E_t$;
- $\Omega_t = E_t$.

target sound	$\Omega_t = A_t$	$\Omega_t = A_t \cup E_t$	$\Omega_t = E_t$	*Orchidea*	T
A Minor	5354	5354	7236	8316	16
Brahms	3446	3446	4235	4845	8
Drops	5902	5902	8270	8925	12
Jarret Vienna	1727	1727	2304	2531	6

4.16: Summary of results of the LP and the Evolutionary algorithms on some dynamic target sounds

Figure 4.16 shows the results we obtained for the dynamic instances, where T is the number of time steps which go from 6 to 16. Here again SPH significantly improves the result on the computed solutions with respect to the current used heuristic. As previously, we point out that this comparison is valid only in the

considered abstract model and in the considered settings (database, constraints,...), which is clearly different from the acoustically perceived quality of solutions.

We also note that the fact of adding E_t in Ω_t never lead to an improvement of the result. The solutions output by the ILP at each time step seem to make a good set of individual solutions in SPH.

4.8 Conclusion

We provide in this chapter the first, to our knowledge, theoretical analysis of the static and dynamic target-based orchestration problems S-TOP and D-TOP, showing both **NP**-hardness, pseudo-polynomial and approximation results. We also note the dynamic case naturally falls into the multistage optimization framework (Gupta et al. (2014)). Finally, we propose an experimental study of our model, comparing it to the *Orchidea* software.

Conclusion and outlook

In this thesis we presented a wide variety of results for maximization optimization problems in the multistage framework and study a direct application of the multistage setting.

In Chapter 2, we presented a $PTAS$ for the MULTISTAGE KNAPSACK problem in the offline setting. As an outlook, it would be interesting to address other maximization optimization problems in the offline setting, as the presence of an approximation scheme for the **NP**-hard KNAPSACK problem contrasts with inapproximability results for polynomially solvable problems in their classical version. Furthermore, we could try to understand the origin of the contrasts between optimization problems in their classical versions and in the multistage setting in order to be able to give a global characterization of multistage problems in the offline setting.

In Chapter 3, we presented a framework for multistage subset maximization problems in the online setting looking at different models with different types of data evolution and transition bonus. An outlook would be to look at the family of multistage subset minimization problems and see if similar results are findable or not.
We also emphasize that we have focused on deterministic algorithms in this chapter. Indeed, some of our bounds can be improved by randomization (assuming an oblivious adversary):

- In the general-evolution model with Hamming bonus assuming sub-additivity and subset feasibility, there is a simple randomized $(2 + o(1))$-competitive algorithm (along the lines of the algorithms in subsection 3.3.1): Initially partition N uniformly at random into two equal-sized sets (up to possibly one item) A and B. At each time, select the optimal solution restricted to A. Again, the algorithm is $(2 + o(1))$-competitive separately on both profit and bonus.

- While the strong lower bound without lookahead in the general-evolution model with intersection bonus still holds, we can get a simple 2-competitive algorithm for lookahead 1: Initially flip a coin to interpret the instance as a sequence of length-2 instances either starting at time 1 or 2. Thanks to lookahead 1, the length-2 instances can all be solved optimally. The total value of all these length-2 instances adds up to at least the optimal value, and the expected value obtained by the algorithm is half of that.

While we believe that we have treated several of the most natural ways of defining transition bonus and data evolution in multistage subset maximization problems, other ways can be thought of, to some of which our results extend. For instance, Theorem 3.1 also works for time-dependent or object-dependent bonus without major modifications (whereas, e.g., Theorem 3.4 does not).

We have not considered computational complexity ; indeed, often we use an oracle providing the optimal solution to instances of a potentially hard problem. However, we mention that, if only an approximation algorithm to the problem at hand was known, we would be able to obtain similar online algorithms whose competitive ratio would depend on the approximation guarantee of the approximation algorithm.

Finally in Chapter 4, we gave the first theoretical analysis of the static and dynamic target-based orchestration problems. The experiments we conducted give hope for possible practical improvements using ILP and shortest path formulations. As said previously, this is only a first step in this direction - there is probably a gap between a good solution for the abstract model we consider and what a composer would consider as a harmonious solution - but a promising step. Also, if one manages to understand better the problem itself and especially to find a unique best dissimilarity measure between different sounds, it would be possible to decrease the difference between the abstract solution given by our model and the best solution from a composer standpoint.

BIBLIOGRAPHY

Bibliography

Akrida, E. C., Mertzios, G. B., Spirakis, P. G., and Zamaraev, V. (2020). Temporal vertex cover with a sliding time window. *J. Comput. Syst. Sci.*, 107:108–123.

Albers, S. (1997). Competitive online algorithm. *OPTIMA*, (54):1–8.

An, H., Norouzi-Fard, A., and Svensson, O. (2017). Dynamic facility location via exponential clocks. *ACM Trans. Algorithms*, 13(2):21:1–21:20.

Archetti, C., Bertazzi, L., and Speranza, M. G. (2003). Reoptimizing the traveling salesman problem. *Networks: An International Journal*, 42(3):154–159.

Ariza, C. (2005). Navigating the landscape of computer aided algorithmic composition systems: a definition, seven descriptors, and a lexicon of systems and research. In *Proceedings of the 2005 International Computer Music Conference, ICMC 2005, Barcelona, Spain, September 4-10, 2005*. Michigan Publishing.

Ausiello, G. and Paschos, V. T. (2018). Reductions that preserve approximability. In Gonzalez, T. F., editor, *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*, pages 243–258. Chapman and Hall/CRC.

Awerbuch, B. and Kleinberg, R. (2008). Online linear optimization and adaptive routing. *J. Comput. Syst. Sci.*, 74(1):97–114.

Bampis, E., Christou, D., Escoffier, B., and Thang, N. K. (2019a). Online-learning for min-max discrete problems. *CoRR*, abs/1907.05944.

Bampis, E., Escoffier, B., and Kononov, A. (2020). Lp-based algorithms for multi-stage minimization problems. *WAOA, to appear*.

Bampis, E., Escoffier, B., Lampis, M., and Paschos, V. T. (2018a). Multistage matchings. In Eppstein, D., editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2018, June 18-20, 2018, Malmö, Sweden*, volume 101 of *LIPIcs*, pages 7:1–7:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Bampis, E., Escoffier, B., and Mladenovic, S. (2018b). Fair resource allocation over time. In André, E., Koenig, S., Dastani, M., and Sukthankar, G., editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 766–773. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Bampis, E., Escoffier, B., Schewior, K., and Teiller, A. (2019b). Online multistage subset maximization problems. In Bender, M. A., Svensson, O., and Herman, G., editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Bampis, E., Escoffier, B., and Teiller, A. (2019c). Multistage knapsack. In Rossmanith, P., Heggernes, P., and Katoen, J., editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Beardwood, J., Halton, J. H., and Hammersley, J. M. (1959). The shortest path through many points. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 55, pages 299–327. Cambridge University Press.

Berman, K. A. (1996). Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks*, 28(3):125–134.

Bertsimas, D. (1988). *Probabilistic combinatorial optimization problems*. PhD thesis, Massachusetts Institute of Technology.

Bertsimas, D. J., Jaillet, P., and Odoni, A. R. (1990). A priori optimization. *Oper. Res.*, 38(6):1019–1033.

Boria, N. and Paschos, V. T. (2011). A survey on combinatorial optimization in dynamic environments. *RAIRO-Operations Research*, 45(3):241–294.

Bredereck, R., Fluschnik, T., and Kaczmarczyk, A. (2020). Multistage committee election. *CoRR*, abs/2005.02300.

Buchbinder, N., Chen, S., and Naor, J. (2014). Competitive analysis via regularization. In Chekuri, C., editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 436–444. SIAM.

Buchbinder, N., Chen, S., Naor, J., and Shamir, O. (2016). Unified algorithms for online learning and competitive analysis. *Math. Oper. Res.*, 41(2):612–625.

Caprara, A., Kellerer, H., Pferschy, U., and Pisinger, D. (2000). Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123(2):333–345.

Carpentier, G. (2008). *Approche computationnelle de l'orchestration musciale-Optimisation multicritère sous contraintes de combinaisons instrumentales dans de grandes banques de sons*. PhD thesis, Université Pierre et Marie Curie-Paris VI.

Carpentier, G., Tardieu, D., Assayag, G., Rodet, X., and Saint-James, E. (2007). An evolutionary approach to computer-aided orchestration. In *Workshops on Applications of Evolutionary Computation*, pages 488–497. Springer.

Cella, C.-E. (2020a). Orchidea livestream hands on tutorial on youtube https://www.youtube.com/watch?v=ltups7urth4feature.

Cella, C. E. (2020b). Target-based assisted musical orchestration: comprehensive overview and state-of-the-art. *in preparation*.

Cella, C.-E. and Esling, P. (2018). Open-source modular toolbox for computer-aided orchestration. In *Timbre conference, Montreal, Canada, 2018*.

Chimani, M., Troost, N., and Wiedera, T. (2020). Approximating multistage matching problems. *arXiv preprint arXiv:2002.06887*.

Cohen, E., Cormode, G., Duffield, N. G., and Lund, C. (2016). On the tradeoff between stability and fit. *ACM Trans. Algorithms*, 13(1):7:1–7:24.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In Harrison, M. A., Banerji, R. B., and Ullman, J. D., editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM.

Downey, R. G. and Fellows, M. R. (2012). *Parameterized complexity*. Springer Science & Business Media.

Eisenstat, D., Mathieu, C., and Schabanel, N. (2014). Facility location in evolving metrics. In Esparza, J., Fraigniaud, P., Husfeldt, T., and Koutsoupias, E., editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 459–470. Springer.

Eppstein, D., Galil, Z., and Italiano, G. F. (1999). Dynamic graph algorithms. In Atallah, M. J., editor, *Algorithms and Theory of Computation Handbook*, Chapman & Hall/CRC Applied Algorithms and Data Structures series. CRC Press.

Escoffier, B. (2005). Approximation polynomiale de problemes d'optimisation: Aspects structurels et operationnels. *These de doctorat universite Paris IX*.

Even, S. and Gazit, H. (1985). Updating distances in dynamic graphs.

Fernández, J. D. and Vico, F. (2013). Ai methods in algorithmic composition: A comprehensive survey. *Journal of Artificial Intelligence Research*, 48:513–582.

Fluschnik, T., Niedermeier, R., Rohm, V., and Zschoche, P. (2019). Multistage vertex cover. In Jansen, B. M. P. and Telle, J. A., editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Fluschnik, T., Niedermeier, R., Schubert, C., and Zschoche, P. (2020). Multistage s-t path: Confronting similarity with dissimilarity. *CoRR*, abs/2002.07569.

Frederickson, G. N. (1985). Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798.

Frieze, A. M. (1985). On the value of a random minimum spanning tree problem. *Discret. Appl. Math.*, 10(1):47–56.

Frieze, A. M., Clarke, M., et al. (1984). Approximation algorithms for the m-dimensional 0-1 knapsack problem: Worst-case and probabilistic analyses. *European Journal of Operational Research*, 15:100–109.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gupta, A., Talwar, K., and Wieder, U. (2014). Changing bases: Multistage optimization for matroids and matchings. In Esparza, J., Fraigniaud, P., Husfeldt, T., and Koutsoupias, E., editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014,*

Proceedings, Part I, volume 8572 of *Lecture Notes in Computer Science*, pages 563–575. Springer.

Heeger, K., Himmel, A.-S., Kammer, F., Niedermeier, R., Renken, M., and Sajenko, A. (2019). Multistage graph problems on a global budget. *arXiv*.

Himmel, A.-S., Molter, H., Niedermeier, R., and Sorge, M. (2017). Adapting the bron–kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35.

Ibarra, O. H. and Kim, C. E. (1975). Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468.

Ivkovic, Z. and Lloyd, E. L. (1993a). Fully dynamic algorithms for bin packing: Being (mostly) myopic helps. In Lengauer, T., editor, *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings*, volume 726 of *Lecture Notes in Computer Science*, pages 224–235. Springer.

Ivkovic, Z. and Lloyd, E. L. (1993b). Fully dynamic maintenance of vertex cover. In van Leeuwen, J., editor, *Graph-Theoretic Concepts in Computer Science, 19th International Workshop, WG '93, Utrecht, The Netherlands, June 16-18, 1993, Proceedings*, volume 790 of *Lecture Notes in Computer Science*, pages 99–111. Springer.

Jaillet, P. (1985). *Probabilistic traveling salesman problems*. PhD thesis, Massachusetts Institute of Technology.

Karlin, A. R., Manasse, M. S., McGeoch, L. A., and Owicki, S. S. (1990). Competitive randomized algorithms for non-uniform problems. In Johnson, D. S., editor, *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA*, pages 301–309. SIAM.

Karlin, A. R., Manasse, M. S., Rudolph, L., and Sleator, D. D. (1988). Competitive snoopy caching. *Algorithmica*, 3:77–119.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York.

Kellerer, H. and Pferschy, U. (1999). A new fully polynomial time approximation scheme for the knapsack problem. *J. Comb. Optim.*, 3(1):59–71.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer, Berlin, Germany.

Kempe, D., Kleinberg, J. M., and Kumar, A. (2000). Connectivity and inference problems for temporal networks. In Yao, F. F. and Luks, E. M., editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 504–513. ACM.

Korte, B. and Schrader, R. (1981). On the existence of fast approximation schemes. In Magasarian, O., Meyer, R., and Robinson, S., editors, *Nonlinear Programming 4*, pages 415–437. Academic Press.

Lawler, E. L. (1979). Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356.

Lin, M., Wierman, A., Andrew, L. L., and Thereska, E. (2012). Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking*, 21(5):1378–1391.

Liu, Z., Liu, I., Low, S., and Wierman, A. (2014). Pricing data center demand response. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):111–123.

Magazine, M. J. and Oguz, O. (1981). A fully polynomial approximation scheme for the 0-1 knapsack problem. *European Journal of Operational Research*, 8:270–273.

Maresz, Y. (2013). On computer-assisted orchestration. *Contemporary Music Review*, 32(1):99–109.

Mertzios, G. B., Michail, O., and Spirakis, P. G. (2019). Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449.

Michail, O. (2016). An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280.

Oguz, O. and Magazine, M. J. (1980). A polynomial time approximation algorithm for the multidimensional knapsack problem. *Working paper, University of Waterloo*.

Olver, N., Pruhs, K., Schewior, K., Sitters, R., and Stougie, L. (2018). The itinerant list update problem. In *International Workshop on Approximation and Online Algorithms*, pages 310–326. Springer.

Pruhs, K. and Woeginger, G. J. (2007). Approximation schemes for a class of subset selection problems. *Theor. Comput. Sci.*, 382(2):151–156.

Ravi, R. and Sinha, A. (2006). Hedging uncertainty: Approximation algorithms for stochastic optimization problems. *Math. Program.*, 108(1):97–114.

Rottner, C. (2018). *Combinatorial Aspects of the Unit Commitment Problem. (Aspects combinatoires du Unit Commitment Problem).* PhD thesis, Sorbonne University, Paris, France.

Schäffter, M. W. (1997). Scheduling with forbidden sets. *Discret. Appl. Math.*, 72(1-2):155–166.

Scheideler, C. and Hajiaghayi, M. T., editors (2017). *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*. ACM.

Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *J. ACM*, 32(3):652–686.

Vazirani, V. V. (2013). *Approximation algorithms*. Springer Science & Business Media.