Amsaa: an Anticipatory Algorithm for Online Stochastic Combinatorial Optimization

By
Luc Mercier
BS, École Polytechnique, 2004
MS, Université Pierre et Marie Curie, 2005

A dissertation submitted in partial fulfillment of the
requirements for the degree of doctor of philosophy
in the department of Computer Science at Brown University

Providence, Rhode Island
May 2009

Vita

2005–2008	PhD, Brown University, Providence, RI, United States
2004–2005	MS, Université Pierre et Marie Curie, Paris, France
2002–2005	BS, École Polytechnique, Paris, France

Acknowledgements

A PhD dissertation might have only one author, but is certainty not the production of one person alone, and many deserve to be thanked today for their help, inspiration, or support.

I discovered Operations Research thanks to Benoit Rottembourg, at the time director of the Bouygues e-lab. Philippe Baptiste, from CNRS/École Polytechnique, advised me to come to Brown University. Here I started working with Pascal Van Hentenryck, who has awarded a great deal of trust and responsabilities; with whom I had many heated and interesting research discussions; and who always supported me, both when research went well and when it did not. Pascal is the person who taught me how to do research, and I am very grateful for that. Later I also had the chance to collaborate with Grégoire Dooms, at the time a post-doc in the Brown Optimization Lab, and to have fruitful exchanges with my thesis committee members, Claire Mathieu from Brown University and Shabbir Ahmed from Georgia Tech. Research cannot be conducted without equipment and assistance, and the technical and administrative staff at Brown did a great job at providing good working conditions to us PhD students at Brown.

I would not have accomplished this work without any of the people above, and I wish to thank sincerely each and every one of them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction: the Case for Online Stochastic Combinatorial Optimization

Throughout this dissertation, we will be concerned with a class of decision-making problems called *online stochastic combinatorial optimization* (OSCO) problems. *Optimization* problems are problems consisting of finding the decision that optimize an objective: typically, maximizing revenues, minimizing costs, or maximizing some service quality metric. Such problems are *online* when the decision-maker has to make a sequence of decisions, while receiving more and more information about her environment, like the weather, stock prices, or inventory levels. An online optimization problem is *stochastic* when, in addition, the decision-maker has some probabilistic knowledge of the future, and *combinatorial* if decisions are discrete (i.e., do this *or* do that) rather than continuous (like turning a knob).

Examples of such problems abound. In inventory management for example, a continuous flow of information about sales and supply is available to the decision-maker, stochastic models of the demands are available, and placing an order is a discrete decision. In chemical engineering, starting a chemical reaction is a discrete decision, and there are uncertainties about the demand and the outcomes of chemical reactions. In revenue management, requests for a good or a service arrive, and the decision-maker has to decide whether to accept or reject the request at a given price, with uncertainty about future requests.

A very thorough reference on OSCO is [50], in which the term *online stochastic combinatorial optimization* was introduced, and in which a class of algorithms for decision-making called *anticipatory algorithms* was studied. This thesis will be in the direct continuation of that line of work.

Yet, today most such problems are treated in ways that ignore some aspects of the problem. In inventory management, it is still common to use simple rules such as "order when the inventory hits a threshold". In manufacturing, the scheduling process is getting more and more complicated by to the "mass-customization" model. Scheduling decisions in such factories are often made ignoring future demands, so only the combinatorics is taken into account. This is the case for example at Dell's Morton L. Topfer Manufacturing Center in Texas [33]: to accommodate Dell's build-to-order model, assembly lines can produced many different computers, but there are set up cost when switching from one model to another. Yet, switching decisions are based with respect to existing orders only, without forecasting. In revenue management, it is traditionally the opposite: uncertainty is at the heart of models used by airlines, but the discrete nature of seats is not considered.

There are two premises to this doctoral work:

1. The Information Technology (IT) infrastructure in medium-sized and large companies provide all what is needed to implement OSCO algorithms;

2. Making decisions using sophisticated algorithms that consider the three aspects (online, stochastic, and combinatorial) alltogether can yield productivity gains.

In the rest of this chapter, I will back the these two claims, arguing that the time is right to do research on OSCO algorithms, and that the market for such solutions will be large.

1.1 The IT Infrastructure Is Here

Developed in the late 80s, Enterprise Resource Planning (ERP) softwares have become ubiquitous. These software provide an integrated IT platform for monitoring and controlling all the resources in a company. Since the mid-90s at Colgate-Palmolive, an early adopter of SAP R/3, the leading ERP software, low inventory level in warehouses automatically triggers production orders, and if necessary orders of raw materials to achieve this production.

What was revolutionnary at that time is today incredibly common: in 2005, the ERP world market was worth $16B (source: ARC Advisory Group, Inc). Virtually all large companies have implemented an ERP, and thanks to the software-as-a-service model, which drastically reduces fixed costs, ERP vendors are now putting much effort into converting midsized and small companies as well.

While ERPs gained penetration, sensors technology also improved. It has come a long way, from printed numbers on items that operators had to type, to bar codes and now to these so fashionable RFID tags. An excellent example of the progresses made in system monitoring are the package tracking tools provided on the Web by all major carriers such as the United States Postal Services and FedEx, which provides updates every few hours or so about the location of a package.

While RFID enabled monitoring items in factories and warehouses, GPSs and wireless data transmission enabled monitoring the location of vehicles. Today, both sensor technology and ERPs are mature, so that softwares have real-time access to a very accurate description of the state of the system being controlled and of its environment.

The high penetration of ERPs and advances in sensors technology provide a fantastic opportunity for Operations Research (OR) practitioners. OR, the science of using resource in the most efficient way, was traditionally used for strategic planning (i.e., long term and often capital-intensive decision such as “should I build a new warehouse?”). Such questions were studied by OR *analysts*, which spent most of their time using spreadsheet and statistics software, and whose end product was a recommendation. This has been very successful in some cases, but as of 2007, there were only 60,000 OR anylists working in the US, according to the Bureau of Labor Statistics of the US Department of Labor. That’s a tiny number given that in 2002 there were 6.9m companies registered in the US that have at least one employee (`http://www.census.gov/econ/census02/data/us/US000.HTM`). Such a small number of OR analysts reflect that for many decisions to be made, asking an analyst is not practical:

- it can be that the response time needed is too short for an analyst;
- or it can be that the monetary returns of a decision made scientifically would be exceeded by the cost of the analysis.

Therefore, there is a clear need for automated decision-making tools, with a short response time, and an operating cost (per decision made) that is much smaller than the one of an OR analyst. Yet, this is exactly what ERPs and sensor technology permits to do. There is therefore a shift from OR analysts to OR *software engineers* whose end products are not recommendations, but *decision support tools* (that produce recommendations, validated by a human operator) or *decision making tools* that directly control some actuators, without human intervention. Which one is best depend on the required response time. Example of decision-making tools in which no human intervention is possible is sheet routing in modular printers currently developed by Xerox [23](in which there are exponentially many ways to route the paper from the supply bin to the output bin, so that the system is resilient to paper jams), or embedded radar control systems under development by Thales [54], a French defense company, in which local search and linear programming are used to schedule several hundreds tasks per seconds.

So far, we have provided arguments that the IT infrastructure enables the use of online decision making software. But we are concerned with, more specifically, online stochastic combinatorial optimization (OSCO) software. To implement such software, being able to monitor the state of the system and to commit decisions or recommendations is not enough: you also need to have access to a stochastic model of the future, and to compute good decisions. Yet, again thanks to ERPs, massive amount of historical data are available, which can be used in various way to provide stochastic information about the future: this data can be used to provide *non-parametric* models, or to *fit* a parametric model, or to *validate* a stochastic model. The maturity of statistics and data-mining, and the abundance of data, provides accurate stochastic models of the future.

The last ingredient needed is the ability to *compute* a good decision, given the state of the system and a stochastic model of the future. That questions was studied for the first time by Bellman in the 1950s, but, despite fifty years of continuous effort, many problems remain computationally hard. The main purpose of this thesis is to advance the state of the art in that respect.

1.2 Talking Into Account All Aspects of OSCO Problems Can Yield Productivity Gains

The previous section argued that the IT infrastructure in modern companies make it *possible* to implement OSCO software, but does not answer the question: is it *worth* the effort? I believe the answer is yes.

Indeed, faced by a OSCO problem, it is usually possible to write software that ignores one or several aspect of the problem, either its online aspect, its stochastic aspect, or its combinatorial aspect. Certainly, there are problems in which one aspect is less important than the others. A premise of this dissertation is that there are problems in which none of the aspects can be ignored without suffering from a substantial decrease in decision quality.

The two main problems we will focus on in this work fall in that category. The first one is a stochastic project scheduling in the pharmaceutical industry. A company has a number of drugs that are potential cures for different diseases. Releasing such a drug on the market requires to pass a series of tests. The problem is *online*, because there is a continuous flow of information about that status of the ongoing tests; it is *stochastic* because tests can fail, preventing the drug from being released; and it is *combinatorial*, because limited resources (labs and workforce) are available to conduct the tests. On such a problem, the goal is to maximize the expected profit, which is the difference between the expected revenues, that depend of the release date of drugs, and expected costs, that depend of the tests that have been conducted.

On this problem, the three aspects need to be considered together to produce high quality decisions:

- Ignoring uncertainty is a very common simplification that can takes several forms. One of them is to make decision assuming the most likely future is the one that will happen. In other words, a point-forecast is used for the future instead of a probability distribution. Such a strategy is a bad idea on this project scheduling problem, as projects with less than 50% chances of success will never be tried, even if they have huge potential returns. Such a strategy obtains expected profit that is 13 % below the one achieved by our best OSCO algorithm in the median case, and up to 39% below on some instance.

- Ignoring the online aspect consists of ignoring that there is an alternation between receiving information about the tests and making scheduling decisions. This has proved to be a very successful on many problems, such as online reservations systems to assign spots to commercial breaks on TV [3, 26]. However, here such a simplification yields an expected profit that is 9% below the one achieved by our best OSCO algorithm in the median case, and up to 26% below on some instance.

- Ignoring the combinatorial aspect of the problems would be ignoring how the availability of the resources creates interactions between the decisions for all the potential drugs to tests. One way of doing that is to compute the expected profit achieved by testing and releasing each possible each drug independently, so that the resource constraint is irrelevant; then ranking drugs by independent expected profit and testing the ones that are on top of the list. That approach provides an expected profit that is 5.5% below the one achieved by our best OSCO algorithm in the median case, and up to 16% below on some instance.

Much of the art of operations research consists of knowing what aspects of a problem can be ignored. This example shows that there are OSCO problems on which high-quality decisions can only be achieved be considering the three aspects altogether.

1.3 The Goal of this Dissertation

We have identified that the infrastructure is in place to plug in OSCO software, and that doing so can provide productivity gains, which are the two premises of this work. Yet, to that date very few companies use OSCO softwares. This is because of two reasons:

- There are still computational barriers: OSCO problems are hard to solve;
- and there are psychological barriers: OSCO is perceived as too complicated, or not worth it.

My goal, in writing this thesis, is to make progress on these two issues. For the computational aspect, I will present a new and efficient algorithm that produce substantial increase in decision qualities over the previous state-of-the-art. This algorithm has theoretical guarantees, such as the convergence of the decision to the optimal one when the available runtime increases, but is also very good at making decisions within a given runtime.

For the psychological aspect, which is probably the most important barrier today to the adoption of OSCO softwares, we will show examples on applications that are as close as possible to the real world problem. The first one comes from the pharmaceutical industry, and features uncertainty about revenues, task durations, tasks outcomes, and costs. The second one is a fairly accurate model of a polystyrene production factory run by ABB, a Swiss engineering conglomerate. These examples should convey the idea that this work is not a mere academic exercise, and that OSCO algorithms can be applied in real-world situations.

1.4 Plan

This thesis is organized as follows. Chapter 2 will describe the few OSCO problems that we will consider in this dissertation. These are the three above-mentioned problems (TV commercial breaks packing, pharmaceutical project scheduling, polystyrene production scheduling), as well as a missile interception problem. Chapter 3 will then review the state of the art of optimization under uncertainty, focusing on the applicability of the described methods to the considered OSCO problems.

Chapter 4 will describe a new model, exogenous Markov decision processes, to model the type of problems we consider. This model will provide the framework for the two next chapters.

Chapter 5 will study widely used algorithms called *one-step anticipatory algorithms* that ignores the alternation of decisions and observations. We investigate why this suboptimal algorithm performs sometimes surprisingly well. However, as mentioned earlier, there are other problems on which this approach is not satisfying, and this will lead us to introduce *Amsaa*, the *Anytime Multistep Anticipatory Algorithm*, in chapter 6.

Empirical results on *Amsaa* will be presented in chapter 7. Chapter 8 will investigate the design choices of a subroutine used in *Amsaa* for which many variants are possible. Finally, chapter 9 will give some concluding remarks.

Chapter 2

Motivation: Several Problems and their Shared Charateristics

To start with some concrete material, here are three examples of online stochastic optimization problem: the Stochastic Multiknapsack problem (SMK), originating from TV commercial break allocation; the stochastic resource constraint project scheuling problem (S-RCPSP), originating from the pharmaceutical industry; the weapon resource management (WRM) problem, a very prospective problem in the field of missile interception; and the batch scheduling (BS) problem, a production problem in the chemical industry.

These problems are all Online Stochastic Combinatorial Optimization (OSCO) problems, and they will serve as motivation for this work, as well as benchmarks.

2.1 The Stochastic Multiknapsack Problem

2.1.1 Description

The Stochastic Multiknapsack Problem (SMK) is an abstraction of an online reservation systems for TV commercial breaks that was introduced in [3]. Customers place requests in real time for some service at a fixed date (e.g., having a given commercial spot broadcasted during a commercial break). The resources are modeled by a multiknapsack constraint: there are several breaks, equivalent from the advertiser point of view, each with a given capacity (its duration). The decision-maker must choose whether to accept or deny the requested service. Customers must be immediately notified of this decision, and all accepted requests must be satisfied. Moreover, accepted requests must be assigned to a specific resource at reservation time and this choice cannot be reconsidered. The goal is to maximize the expected profit of the served requests which come from different types with different characteristics and arrival frequencies.

The input is specified as follow. The number of bins and their respective capacities are given. A collection of item types is given. Each item belongs to a given type, which specifies its value (the price the advertiser will have to pay) and its weight (the duration of the ad). For each type, the sequence of arrival of items of this type is modelled as a Poisson process of given parameter.

2.1.2 Scenario and Offline Problem

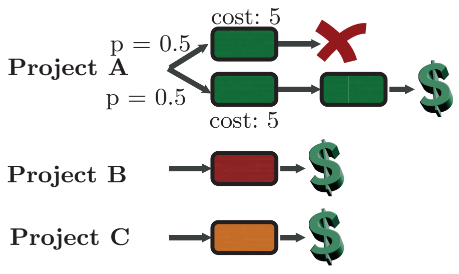
On the SMK problem, one can define a concept of *scenario*. A scenario is one possible realization of each of the Poisson processes associated with the item types. The *offline problem* is the problem of making decisions given a scenario. This is the problem faced by a decision maker that could perfectly foresee the future. Here, this problem is a multiknapsack problem. Indeed, if a decision maker knows the set of items that will arrive from the current time to the date of the service, their respective arrival date are irrelevant: all that makers if to choose how to allocate these items to the resources — and that's a multiknapsack problem.

2.2 The Stochastic Resource-Constained Project Scheduling Problem

2.2.1 Description

Another problem is the *stochastic resource-constrained project scheduling problem* (S-RCPSP), originating from the pharmaceutical industry [17]. A pharmaceutical company has a number of candidate molecules that can be commercialized if shown successful, and a number of laboratories to test them. Each molecule is associated with a project consisting of a sequence of tasks. Each task has a duration, a cost, and a result (i.e., a failure, which ends the project, or different degrees of success, which allow the project to continue). Tasks are not preemptive, and cannot be aborted once started. A project is successful if all its tasks are successful. A successful project generates a revenue which is a known non-increasing function of the completion date of the project. The goal is to schedule the tasks in the laboratories subject to the precedence and resource constraints to maximize the expected profit, the profit being the difference between the project revenues and their costs. The resource constraints impose that the number of tasks executing at any time t does not exceed the number of labs.

Each molecule's project has its own stochastic model. The realizations of a task are triplets of the form (duration, cost, result) and the durations, costs, and outcomes of the tasks in a given project are not independent. The more successful a task is, the higher the probability that the next task in the project will be successful too. Formally, the stochastic model for a project is a heterogeneous first-order Markov chain: For each task i, a transition matrix gives the probability of the realization of task $i + 1$ given the realizations of task i. The task realizations, transition probabilities, and revenue functions are all given. Observe that it may be optimal to stop a project even if it has not failed so far. It is also possible that, at a given time, not scheduling any task in an available lab is



- All tasks have duration 2
- All costs are zero, except task A1, cost 5
- There are 2 labs, one available at time 0, one available at time 1.
- The revenues, as function of completion time, are:

Compl	**<= 2**	**3**	**4**	**5**	**>=6**
rev. A	45	45	45	22	0
rev. B	18	9	0	0	0
rev. C	15	8	1	0	0

Figure 2.1: An Instance of the Stochastic Project Scheduling Problem.

optimal. Indeed, waiting may reveal uncertain information and allow for more informed decisions, as already demonstrated in dynamic fleet management [8].

2.2.2 Example

Figure 2.1 depicts a small instance which we will use to illustrate soem algorithms. In the instance, there are 3 projects and 4 tasks. There are two laboratories, one immediately available, the other that will become available at time 1.

We will benchmark algorithms on a collection of instances introduced in [21], and itself based on the instance studied in [17]. Figure 2.2 depicts the stochastic models of the regular instance, that we will denote *Reg*. In this instance, two labs are available. The figure shows the five projects. Tickness of arrows represent transition probability: the thicker, the more likely. Boxes represents tasks, with their length represented by their length (the scales show the time units), and their costs written inside the boxes. The color depicts whether the test succeeds. Revenue functions are not represented here.

2.2.3 Scenario and Offline Problem

Like for SMK, it is possible to define *scenarios* on this problem. Here, a scenario specifies, for each molecule, a path in the Markov model associated with the project for that molecule. On the depicted instance, there are only two different scenarios, corresponding to the duration of task A.2.

Given a scenario, the *offline problem* is the problem faced by a decision maker who can foresee that this scenario is the actual one. For the depicted instance, the offline optimal schedules for the two possible scenarios are shown in Figure 2.3. Note that they differ at the first decision when the uncertainty is not yet resolved. Hence the optimal online policy necessarily acheives an expected profit inferior to those of a perfect clairvoyant decision maker.

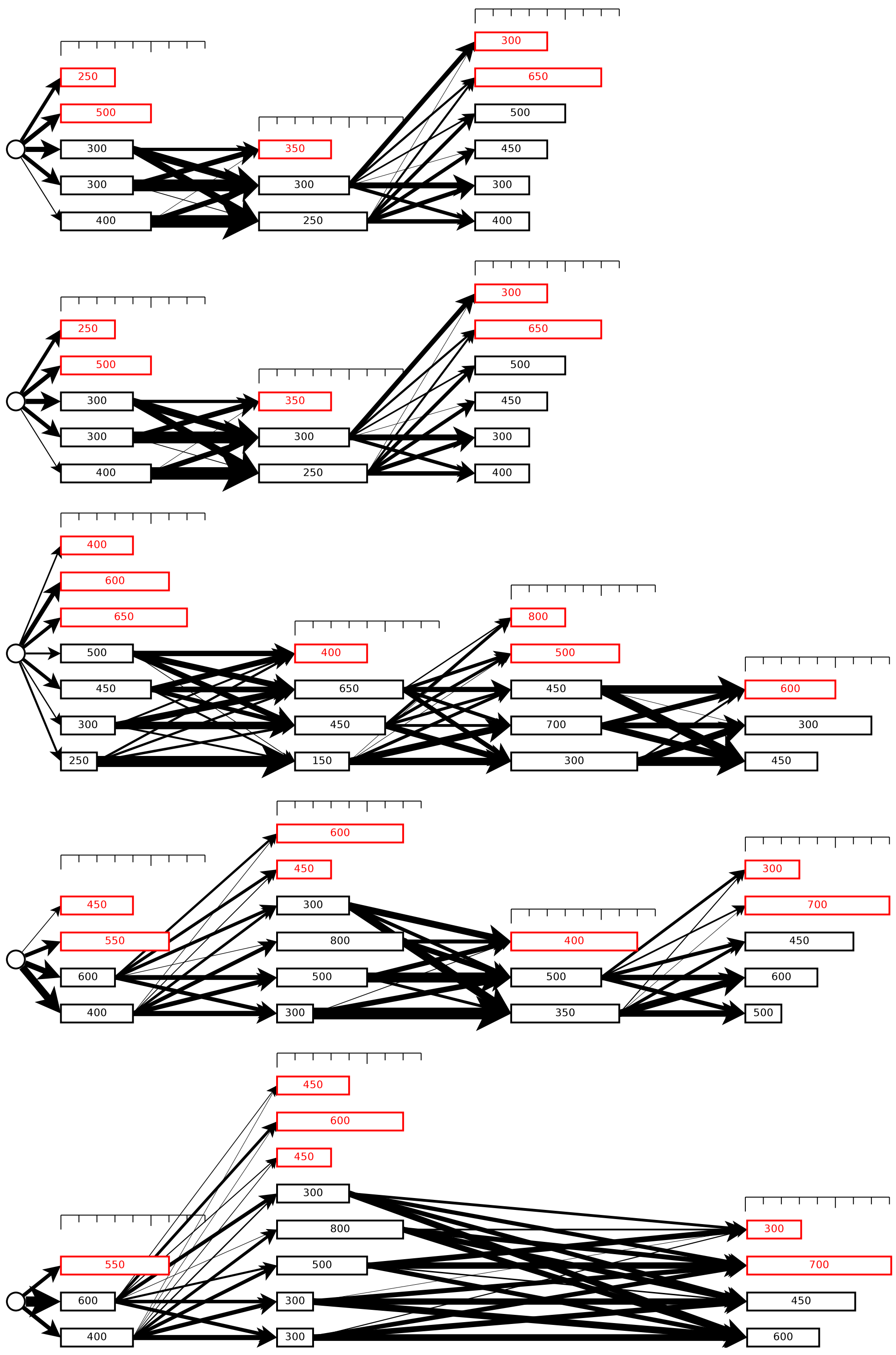


Figure 2.2: Stochastic model for instance *Reg* of the S-RCPSP.

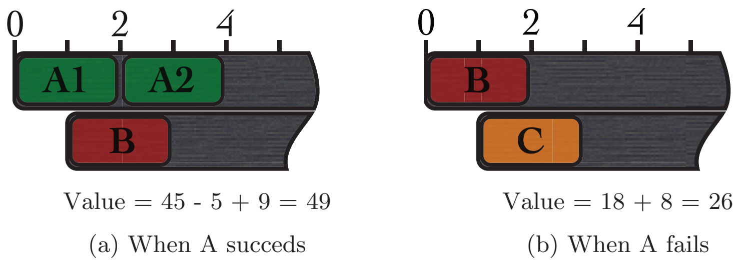


Figure 2.3: The Offline Optimal Schedules for the Stochastic Project Scheduling Instance.

2.3 The Weapon Resource Management Problem

2.3.1 Description

There are many variants of the Weapon Resource Management Problem (WRM), all concerned with the optimal utilization of a limited resources (ammunitions, missiles...) against a set of strategic targets (mobile or not). The variant we present here was introduced by David Grabiner from the MIT Lincoln Laboratory, and is inspired by [28].

In this problem, the decision maker is facing a salve of missiles coming toward his country. If the targeted country is the United States, it is a reasonable assumption that all the enemy missiles will be launched at once, because such a strike will undoubtedly result in a reaction of total elimination of the attacker — and so lauching all the missiles at once is optimal.

Radars can already detect and track incoming missiles; in the near future, the current research will also provide software tools to predict their target with great accuracy. Hence, we assume that the decision maker can fully observe the whole set of incoming missiles, $e_1, \ldots, e_E$, and their respective target $\mathrm{trg}(e_1), \ldots, \mathrm{trg}(e_E)$. We also assume knowledge of the probability $p(e_i)$ that the missile e_i will successfully reach and destroy its target.

These targets belong to a set of assets: $\{\mathrm{trg}(e_1), \ldots, \mathrm{trg}(e_E)\} \subseteq \{a_1, \ldots, a_A\}$. Asset a_i is given the value $v(a_i)$. The goal is to protects the assets with a limited sets of missile interceptors (or resources) $r_1, \ldots, r_R$. Each such interceptor r_i can intercept any target, and is succesful with a known probability $p(r_i)$ that depends only on the resource, not on the missile being intercepted.

Interceptors fly faster than missiles, so there are several opportunities to send interceptors at a given missile. In this simple model, there are a few discrete time steps $0, \ldots, T$, each lasting for a few tens of seconds. The travel times of the incoming missiles is of one to three minutes, so the number of time steps T is likely to be between 5 and 10. At time 0, no incoming missiles has been intercepted. In each time period $0 \leq t < T$, the decision maker send a number of interceptors to some missiles, and oberve whether these interceptors were succesful. We denote $\mathrm{live}(e_i, t)$ the boolean variable that is *true* if missible e_i has not been succesfully intercepted at the beginning of time period $0 \leq t \leq T$.

The goal is to maximize the expected expected (repetition intended) value of the assets that are not destroyed by the incoming missiles, plus a value for each unused resource. Indeed, at time T, the expected value of the non-destroyed assets is the following function of the vector $\text{live}(e_1, T), \ldots, \text{live}(e_E, T)$:

$$f\left(\text{live}(e_1, T), \ldots, \text{live}(e_E, T)\right) = \sum_{i=1\ldots A} v(a_i) \cdot \prod_{\substack{j=1..E \\ \text{trg}(e_j)=a_i \\ \text{live}(e_j,T)}} \left(1 - p(e_j)\right).$$

The problem consist of finding the way of sending the interceptors that maximizes $\mathbb{E}\left[f\left(\text{live}(e_1, T), \ldots, \text{live}(e_E, T)\right) + c(\text{nbr unused resource})\right]$. Note that the fact that f is the *expected* value of the remaining assets has nothing to do with stochastic optimization, and f can be simply considered as any other utility function.

If there are fewer remaining resources than time periods, then the problem is relatively easy, because exactly one interceptor should be send per time period. Indeed, it would not make sense to send several ones at once, since there is time to wait for the observation of whether each interceptor is succesful, enabling more informed decisions.

Alas, due to the short time horizon it is likely that there will be more resources than time periods. In that case, deciding the number of interceptors to send is in itself difficult.

2.3.2 Scenarios and Offline Problems

A *scenario* can be naturally defined for this problem, ans is simply the subset $S \subseteq \{r_1, \ldots, r_R\}$ of successful resource will be succesful at making an interception. The scenarios do *not* contain information about whether missiles are sucessful, because as explained above this is irrelevant from a stochastic optimization perspective: it is already taken into account in the objective function.

The *offline problem* consist of, knowing the set of succesful interceptors, determining the optimal allocations of these interceptors to the missiles. If there are at least as many successful resources than missiles, then the solution is trivially to intercept them all. In general, the offline problem is not as simple, but we will see that it is solvable in polynomial time.

2.4 The Batch Scheduling Problem

2.4.1 General Description

In production, companies have switched from single-product factories to factories that produce a large number of different items. This is especially true is the automotive industry, where the number of vehicle configurations may well exceed the total number of cars that will ever be produced — this is called the *mass-customization* paradigm. To a lesser extent, this has also affected the chemical industry. Our next motivating problem is an accurate model of the problem of scheduling the production of polystyrene in a factory operated by ABB, a Swiss conglomerate. This problem has been studied in a few papers, for instance [44], in which a stochastic programming approach is proposed, and [49], which proposes to solve a two-stage stochastic program using an evolutionary algorithm for the first stage decisions and integer programming for the second stage ones.

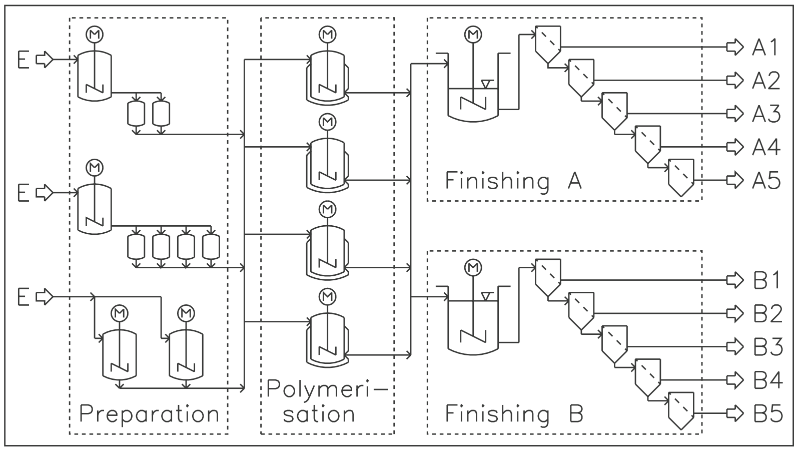


Figure 2.4: The three stages of the polystyrene production process

This plants produces 10 different varieties of polystyrene: it produces polystyrene of two *types*, A and B, and for each type there are 5 *grain sizes*. The ten varieties are denoted A1, . . . , A5, B1, . . . , B5. Figure 2.4 shows the three stages of the polystyrene production. Raw materials, entering on the left, need to be prepared for the polymerizations. Then four identical reactors are available to polymerize the materials. Each polymerization takes 17h, but different recipes produces polystyrene of either A or B type. In addition, each such reaction produces grains of all sizes, but some recipes favor the production of large grains, while others favors the production of small grains. In total, there are 10 possible recipes, each of them producing mostly one of the ten polystyrene varieties. For instance, the recipe that favors A2 produces 65% of A2, but also 8% of A1, 24% of A3, 2% of A4 and 1% of A5.

The polystyrene produced by one polymerization is referred to as a *batch*. Reactors must be used at their nominal capacity, so all polymerizations produces the same total quantity of polystyrene, and one batch will be our unit of quantity.

When a polymerization ends in a reactor, the whole content of the reactor is transferred at once into the *mixer* of the *finishing line* corresponding to the type of polystyrene produced. This mixer has a capacity of three batches, but, if at any time it contains less than 0.1 batch, then it must be shut off and some set up has to be done before it can be restarted. This is costly: a start up and shut down cycle has an estimated cost of 6k$, that is, as much as the cost of doing the polymerization of 6 batches.

The mixer slowly feeds the separators, that dispatches grains according to their sizes. This feed rate must be between 0.10 and 0.25 batches/h. The lag time of the separation stage is 24h, regardless of the feed rate.

The decisions maker has to choose what recipes to use, when to perform the polymerizations, and at what rate to feed the separators, subject to constraints about the capacities of the reactors (a cumulative resource constraint of capacity 4), capacities of the mixers, and capacities of the separators. There are no constraints for the preparation stage: the reactors are the bottleneck of the process, so for any batch production schedule it is possible to accommodate the preparation resources.

Customers place orders online for a given quantity of a given variety of polystyrene to be delivered at a given due date. While questions of revenues management would certainly be interresting, here we assume that the unit price for each variety of polystyrene is public, and that orders cannot be denied. In our regular instance, the price is \$2k per batch. This is a just-in-time (JIT) scheduling problem: if the polystyrene is produced before the due date, it must be hold until then, implying an inventory cost of \$50/(batch· day). On the other end, if the order is fulfilled after the due date, then late fees of \$50/(batch· day) are paid to the customer.

In real life, there are several sources of uncertainty in that problem. The most important is demand. The proportion of each grain size produced by a polymerization reaction is also slightly uncertain. In addition, it happens (rarely) that a polymerization reaction fails, in that case its product has to be discarded. Finally, reactors might breakdown. For the sake of simplicity, we will consider only demand uncertainty in our study.

2.4.2 Details of the Model Used

Time Discretization The model of the problem we use will follow closely the model presented in [44]. This is a simplification of the problem that recognizes that, because the inventory and late fees cost for a day or two are small compared to the other costs and revenues involved, the precise hour-by-the-hour scheduling of the polymerization is not of critical importance. Therefore, a time discretization, with periods of 48 hours, is used.

Because the duration of all polymerizations are identical, and because it takes a fixed duration of 24 hours for polystyrene to go from the mixer to the inventory, three time scales will be used depending on the context. A first one, used for polymerization decisions, will have its period 0 for time $t = 0$ to $t = 48$h, period 1 from $t = 48$h to $t = 96$h, etc. A second time scale, used when dealing with mixers decisions, will have its period 0 from $t = 17$h to $t = 65$h, its period 1 from $t = 65$h to $t = 113h$, etc. Finally, the last one, used for inventory, due dates, and sales, will have its period 0 from $t = 41$h to $t = 89$h, period 1 from $t = 89$h to $t = 137$h, etc. Thanks to the fixed duration of the polymerization and finishing process, a polymerization done in period i for the first time scale will be in the finishing line in period i of the second time scale and arrive in the inventory in period i of the third time scale. Because of this fact we will never need to make what time scale we are using explicit.

Stochastic Demand Model The demand model found in [44] and in subsequent papers is not sufficient for online simulations, because it does not have a concept for when orders are placed, and so we had to refine it. For each polystyrene type, we model the arrival of orders by a Poisson process. As time is discrete, there is some probability that one order will come in each period. This order has then the following characteristics. It is for a quantity that is generated uniformly at random between 0 and a maximal size for orders. In all studied instances this maximal size was 4.0 batches.

The due date of this order follows a geometric distribution with parameter 1/2, with an offset of the order placement period plus two. Orders are placed at the beginning of a period, but due dates are with respect to the end of a period. That is, and order arriving at the beginning of period 3 will be due at the end of period 5 with probability 1/2, at the end of period 6 with probability 1/4, and so on.

Reactor Capacity Constraints The capacity constraints due to reactors is simply that the number of polymerizations to be scheduled in each period cannot exceed 12. Indeed, there are 48 hours per period, and polymerizations takes 17 hours, so on one reactor $\lceil \frac{48}{17} \rceil = 3$ reactions can be started, and so $3 \times 4 = 12$ reactions can be started on all the reactors. In the previous computations, rounding was to the next integer rather than the previous one, because thanks to the shifted time scales, what matter is only the start of a polymerizations: that a reaction ends after the end of the period is irrelevant.

Note that while 12 polymerizations can start in a period, only 23 can start in two consecutive periods, because security constraints do not allow to start two polymerizations on two different reactions less than 4 hours apart. This constraints can be enforced by having reactor capacity constraints for all time windows, instead of period-by-period. For a first study, we chose to use the simpler period-by-period constraint, which is slightly too relaxed.

Mixers and Finishing Lines Constraints The cost of starting and shuting down finishing lines, together with the obligation not to run the mixers empty, is what makes this problem difficult. Recall that there is one mixer and one finishing line dedicated to each type. A natural idea is to introduce variables storing the mass of each product in each mixer at a given time. Then, when mass is transfered from the mixer to the finishing line, because all the products are blended together, the ratios of product feed rates equal the ratios of their mass in the mixer. In other words, if there is a mass $m_1(t), \ldots, m_5(t)$ of product A1,…,A5 in mixer A at time t, then the physics of the feeding process respects:

$$\frac{\mathrm{d}m_1}{\mathrm{d}t} = \frac{\mathrm{d}m_2}{\mathrm{d}t} = \cdots = \frac{\mathrm{d}m_1}{\mathrm{d}t} = \frac{\mathrm{d}(m_1 + \cdots + m_5)}{\mathrm{d}t}$$

This is a non-linear constraints which is difficult to incorporate into an offline problem. To avoid dealing with it, [44] suggested not to use variables representing the mass of products in the mixers. The idea is to model the implications of the mass constraints in the mixers without explicitly introducing mass-in-mixer variables.

We derive an online model by adapting the ideas proposed in the cited article for the offline model as follows. At the beginning of each period, each mixer will be either ON or OFF. For each period, a decision has to be made for each finishing line. If a line is ON at the beginning of the period, feasible decisions are ON and ON_OFF. If it is OFF, feasible decisions are OFF, OFF_ON, and OFF_ON_OFF. Decisions ON and OFF incur no cost, decision ON_OFF incurs the cost of a shutdown, decision OFF_ON the cost of a startup, and decision OFF_ON_OFF the cost of the startup and a shutdown.

There are lower and upper capacity constraints related to the finishing lines. First, upper capacity constraints simply state that no production can be done when the line is off. Lower constraints state that, for any time window (i.e., any range of consecutive time periods), the production for the considered polystyrene type must be at least 4.8 batches times the number of time period the line is ON, and minus the maximal decrease of the mass stored in the mixer during that time window. For example, if at the beginning of the time window the line is on, there can be up to 3 batches in the mixer. If at the end, it is still on, there must be at least 0.1 batches in the mixer, so there is a maximal decrease of 2.9 batches.

2.4.3 Scenarios and Offline Problem

Scenarios are very easy to define in this model, since the only uncertainty comes from demands. A scenario is thus a collection of orders. Recall that an orders is defined by:

- a placement time;
- a product;
- a quantity;
- a due date.

The offline problem consist of computing, for all future periods:

- the number of polymerizations to start for each recipe;
- the status decisions for each finishing lines;
- the amount of each product to take away from inventory as to fulfill each order;
- the inventory level of each product.

This offline problem can be modelled as an integer program. The number of polymerizations per recipe and per period is integral, as are indicator variables of the finishing line status decisions. Other variables (inventory levels, sales, shortage, and cost) and continuous.

2.5 Differences, Common Characteristics, Classification

The four problems presented above differ widely in their field of application; yet from the point of view of an operations researcher, they present many similarities. To delineate precisely the scope of this doctoral work, we will now compare these problems, and specify which of there common characteristics condition the applicability of the methods discussed in the next chapters.

First of all, each of these four problems is an Online Stochastic Combinatorial Optimization (OSCO) problem. For each of them, there is an alternation of observations of some uncertainty (request arrival, yield of a polymerization...) and decision-making. Also, all these problems are combinatorial, because decisions are discrete (accept or deny a request; schedule one test or another...), and because of the presence of *resource constraints* (labs, interceptors, reactors...). The case of continuous decision space will only be briefly discussed at the end of this thesis: for now, we restrict our attention to purely discrete action spaces.

Finally, for each of these problems, we we able to define a concept of scenario, and of offline problem. This is an important characteristic, that not all stochastic optimization problems share: these problems have *exogenous uncertainty.* To explain this in greater detail, here follows a classification of stochastic optimization problems.

Traditionally, these problems were separated into two classes according to the exogenous or endogenous nature of their uncertainty. We refine this classification, using three categories instead of two.

Purely Exogenous Problems. These are problems in which the uncertainty, and the way it is observed, is independent of the decisions. Online stochastic combinatorial optimization problems in which the uncertainty comes from the behavior of customers or suppliers are purely exogenous. In this class, there is a natural concept of scenario (e.g., the sequence of customer requests) and, given two scenarios, it is possible to compute when they become distinguishable. Nature is typically considered exogenous (e.g., water inflow in hydroelectric power scheduling), as well as prices in perfect markets, since an atomic agent cannot influence prices. The SMK belongs to that class.

Purely Endogenous Problems. These are problems for which there is no natural concept of scenarios. Most benchmark problems for Markov Decision Processes are of this nature. For instance, problems of controlling robots typically have uncertainty derived from the imperfection of the actuators, and depends strongly on the signals they receive. It is not completely impossible to define scenarios on these problems, in the form of a collection of statements such as "if at time t signal x is sent to actuator a, the response will be r", but such scenarios have no clear meaning or value.

Stoxuno Problems (STochastic Optimization problems with eXogenous Uncertainty and eNdogenous Observations). These are problems like the S-RCPSP, for which the underlying uncertainty is exogenous, but observations depend on the decisions. In these problems, the concept of scenario is well-defined and meaningful. However, given two scenarios, it is not possible to decide when a decision maker will be able to distinguish them. Many scheduling problems with uncertainty on tasks belong to this category, as does the lot sizing problem in [24]. The BS and WRM problems also elong to that class.

A discussion is needed here, because the boundary between Stoxuno and purely endogenous problem is somewhat fuzzy. Indeed, it is always possible to ignore the presence of scenarios, and to model a stoxuno problem in a purely endogenous fashion. Reciprocaly, one can always introduce a concept of scenario in an endogenous problem: simply introduce a random variable rich enough to provide the source of all future uncertainty. Despite this apparent equivalence, I claim that it makes sense to separate endogenous and stoxuno problems, because some problems will fit more naturally in models with or without scenarios. For example, when introducing an artificial random variable to represent all the uncertainty in an endogenous problem, it is likely that the associated offline problem will have no usable special structure, and thus be very hard.

Amsaa, the algorithm that constitute the heart of this thesis, applies to both purely exogenous and stoxuno problems.

Chapter 3

A Review of the State of the Art

A lot of work has already been achieved on optimization under uncertainty. We will now review some techniques that have been proposed for decision making under uncertainty in general, and discuss their applicability to the problems we have described.

3.1 Control Theory

Maybe the most well-established part of decision making under uncertainty is control theory. The problem here is to control an attribute of a physical system (the position or the speed of an object, the temperature of an oven or a room,. . .) by the mean of actuators (e.g., motors, heaters, air conditioners, etc). Because physical systems cannot be modelled perfectly, there is uncertainty about the effect of using the actuators. Therefore, the ensure the desired effect is achieved, the actuators are controlled by a controller which monitors the state of the physical system thanks to sensors (e.g., optical wheels to measure angles, accelerometers, compasses, temperature sensor,. . .).

Control theory is very well understood, and used in many every day objects. An oven thermostat is the most simple example. A more complex control system is the power steering system of a car controls a motor turning the stirring wheel, and measures the torque that the drivers hands apply to the wheel. Also in cars, the cruise control system is a controller that controls the throttle (and on some cars the gears) so that the car goes at a target speed. It is easy on this example to see why control problems belong to optimization under uncertainty: for a cruise control system, the natural objective function to minimize is the mean square of the difference between the actual speed and the target speed.

Alas, control theory is not applicable to the OSCO problems we consider. First, in control theory, time is continuous: the input of the actuators exists a any point in time, and so this input is continuously being adjusted. Also, while there is uncertainty, no stochastic distribution of the output for a given input is taken into account. Finally, control systems needs a simple model of the physical system being controlled, and constraints such as precedence and resource constraints in our problems make the systems we consider too complex to model for control theory.

3.2 Robust Optimization

While control theory consider systems in which the decision (the input) can be adjusted at all time, robust optimization consider problems in which the decision maker has to make some commitment about decisions that cannot be revised. Typically, it is problems in which, when the uncertainty is observed, the decision maker has little or no mean to react, and thus must have made robust decisions, i.e., decisions that we ensure some goal to be reached regardless of the uncertainty.

Every one of us does robust optimization in our lives. For example, when scheduling a taxi ride to catch a flight, we add slack time to take into account the uncertainty about whether the taxi will be on time and about the duration of the ride.

In production systems, many parameters might be uncertain. Robust optimization consist of optimizing an objective function for a worst case, given that a number of parameters are only know to belong to some interval.

Robust optimization is not what we want to use on the OSCO problems previously described, because in these problems the decision maker has opportunities for new decisions after observing some uncertainty, and because probabilistic knowledge is available, rather than merely being given intervals for parameters.

3.3 Online Algorithms

A branch of algorithmic theory is devoted to online problems, in which there is an infinite alternation of observations and decisions, and the objective is defined asymptotically.

3.3.1 Adversarial Setting

The majority of the work of online algorithms is concerned by worst case performance analysis, where the performance metric considers the worst possible sequence of observations for a given algorithm. Consider for instance the the online bin packing problem [45], defined as follow. A decision maker has an infinite number of identical bins to its disposal. An infinite sequence of items of weight in the interval $[0, 1)$ arrives, one by one. A bin can contain any number of items, provided that the sum of their weights does not exceeds one. At each time step, the decision maker observes an incoming item, and must either assign it to a previously used bin, or put it in a new bin. The number of used bin will of course goes to infinity, but the goal of the decision maker is to minimize the rate of the number of bin used. More accurately, denote $\mathcal{A}$ is a decision-making algorithm, and σ a finite sequence of incoming items. Let $c_{\mathcal{A}}(\sigma)$ be the number of bins used by $\mathcal{A}$ faced by the sequence σ, and let $c(\sigma)$ be the minimal number of bins in which the items in σ can fit. The objective is to find an algorithm that has a minimal asymptotic performance ratio $R_{\mathcal{A}}$, defined as

$$R_{\mathcal{A}} = \limsup_{k \to \infty} \sup_{\sigma, c(\sigma)=k} \frac{c_{\mathcal{A}}(\sigma)}{k}.$$

Considerable effort have been made studying this problem and other similar ones. To date, it is known that the optimal asymptotic performance ratio for online bin-packing is in the interval $[1.540, 1.589]$ [45].

It is important to note that this objective function is based on a worst-case: the sup operator is based on all possible finite sequence of a given cost. This is an example of *adversarial uncertainty*: here, the reason there is no probability distribution of the uncertainty is not motivated by the idea that this distribution is not known to the decision maker, but rather that the sequence of item is not random: it is generated by an adversary that can observe how the decision-maker behaves, and can act accordingly.

Therefore, these ideas are not directly useful for the problems we consider. Indeed, uncertainty such as whether a drug is effective, or about future demand, are not governed by an adversary. While you might not know the goals of your customers, it is certainty not to place the sequence of orders that will make scheduling your production the hardest. Therefore, it is much more reasonable to model customer behavior as stochastic rather than as adversarial.

3.3.2 Stochastic Setting

Algorithmic theorists have also studied, perhaps to a lesser extent, the stochastic setting, in which it is assumed that there exists a probability distribution (known or not) of the uncertainty, effectively turning the problem into a purely exogenous OSCO problem. In that case, the used metric in an expectation. For example, in the stochastic variant of the online bin packing, the natural performance metric is

$$ER_{\mathcal{A}} = \limsup_{n \to \infty} \mathbb{E}\left[\frac{c_{\mathcal{A}}\left(\boldsymbol{\sigma}_{1..n}\right)}{c\left(\boldsymbol{\sigma}_{1..n}\right)}\right],$$

where $\boldsymbol{\sigma}_{1..n}$ is the random sequence of the first n items. [19] introduced a remarkable algorithm, $SS^{\star}$, which achieves $ER_{SS^{\star}} = 1$ whenever items weights are independent and identically distributed fractions of bounded denominator.

This algorithm learns the distribution of item, and this idea can and has been used outside of the algorithmic theory field. However, besides that learning component, ideas from this branch of research seems limited in applicability to real-world problems, because the ideas and analysis for online algorithm is very problem-dependant.

3.4 Markov Decision Model-Based Methods

A very broad class of techniques aim at solving stochastic optimization problems modeled as Markov Decision Processes (MDPs). Many terms refers to some subset of techniques for solving MDPs, such as reinforcement learning [47], stochastic dynamic programming [11], and heuristic search algorithms [13]. Because there does not seem to be a consensual term to describe the whole field of MDP solving, we will simply use the phrase MDP-based techniques.

These techniques are applicable to the kind of problems we consider, therefore we will give a more thorough review than for control theory, robust optimizations, and online algorithms.

MDPs are the model of choice for purely endogenous problems, but they can be and have been used also on Stoxuno and purely exogenous problems. There are several variants of MDPs. They are several variants of MDPs. Here follows the definition of MDPs with rewards on final states only (no transition cost), with no discounting, and with finite state spaces. This is the definition we will use is this dissertation. The other variants will be discussed later.

3.4.1 Markov Decision Processes

A finite undiscounted Markov Decision Process $(S, s_0, F, X, \perp, \mathcal{X}, f, \mathcal{P})$ consists of:

- a finite state space S, an initial state $s_0 \in S$, and a set of final states $F \subseteq S$.
- a decision space X containing a decision $\perp$ (denoting no action) and a function $\mathcal{X} : S \to X$ returning the set of feasible decisions in a given state such that $\forall s \in S, 0 < \#\mathcal{X}(s) < \infty$ and that $\forall s \in F, \mathcal{X}(s) = \{\perp\}$.
- a bounded reward function $f : F \to \mathbb{R}$.
- a transition function $\mathcal{P} : S \times X \to \text{prob}(S)$, where $\text{prob}(S)$ is the set of probability distributions over S, satisfying $\forall s \in F, \mathcal{P}(s, \perp)(\{s\}) = 1$.

For convenience, we write $\mathcal{P}(\cdot|s, x)$ instead of $\mathcal{P}(s, x)(\cdot)$. A run of an MDP $(S, s_0, F, X, \perp, \mathcal{X}, f, \mathcal{P})$ starts in initial state s_0. At a given state s, the decision maker selects a decision $x \in \mathcal{X}(s)$ which initiates a transition to state s' with probability $\mathcal{P}(s'|s, x)$ (in case of finite state space. More generally, the transition goes to a state $s' \in A \subseteq S$ with probability $\mathcal{P}(A|s, x)$) for any measurable set A. The resulting sequence of states and decisions, i.e.

$$s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} \dots \xrightarrow{x_{t-1}} s_t \xrightarrow{x_{t+1}} \dots,$$

is called a trajectory. This random process is said to be Markovian because, conditionally on s_i and x_i, the probability distribution of s_{i+1} is independent of the past trajectory.

We also assume that the horizon is finite. That is, there exists an integer T such that all trajectories starting in s_0 are such that s_T is final. A corollary of that assumption is that the state space graph has to be acyclic. (Most of this work would still be valid with the weaker assumption that a final state is reached almost surely in finite time regardless of the decisions made). Under these assumptions, the objective of the decision maker is to maximize $\mathbb{E}\left[f(\boldsymbol{s}_T)\right]$.

3.4.2 Policies, Value functions, and Optimality

A (deterministic) Markovian policy $\pi : S \to X$ is a mapping from states to feasible decisions (i.e., $\forall s \in S,\ \pi(s) \in \mathcal{X}(s)$). The value $v_\pi(s)$ of policy π in state s is the expected value obtained by running policy π from state s. A policy π is optimal if $v_\pi(s_0)$ is maximal among all policies.

A value function v is a map $S \to \mathbb{R}$. The Q-value function canonically associated with v is the mapping $S \times X \to \mathbb{R}$ defined by $Q(s,x) = \sum_{s' \in S} \mathcal{P}(s'|s,x)v(s')$. Given a value function v and a state s, a decision $x \in \mathcal{X}(s)$ is *greedy* if $Q(s,x) = \max_{x' \in \mathcal{X}(s)} Q(s,x)$. We assume that there is a rule to break ties, so we can talk about "the" greedy decision even though it is not unique. The *greedy policy* π_v associated with a value function v is the policy defined by taking the greedy decision in every state. A value function is optimal if the associated greedy policy is optimal. A necessary and sufficient condition for v to be optimal is that, for all state s reachable under π_v, we have $v(s) = f(s)$ if s is final, and $Res_v(s) = 0$ otherwise, where $Res_v(s) = v(s) - \max Q(s,x)$ is called the *Bellman residual* of v at s. Under our assumptions, there is always an optimal value function $v^\star$.

3.4.3 Variants

The above definition of MDPs is the one will we use later in this document, but is actually not the most common variant. Often, rewards are associated to transitions rather than to final states: a reward function $R : S \times X \times S \to \mathbb{R}$ specifies the reward $R(s,x)$ obtained if decision x in made in state s. In that case, for the finite horizon case the objective function is to maximize the total reward among all the transitions in the process. An alternative objective function is the expected total discounted reward: a factor $0 < \gamma < 1$ is given, and the trajectory

$$s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} \ldots \xrightarrow{x_{T-1}} s_T$$

is associated with the reward

$$\sum_{t=0}^{T-1} \gamma^t R(s_t, x_t, x_{t+1}).$$

This discount factor accounts for the preference for the present. In case of monetary rewards, it accounts for the existence of loan markets, that makes it equivalent to get a certain amount of money now, or to be certain to get a $1/\gamma$-times greater amount of money in one time period.

The discounted case also allows to consider infinite horizon, in which the decision process never ends. Indeed, as long as the reward at each step is bounded, then the discounted total reward as a finite limit. When the horizon is infinite, is it possible to have cycles in the state-space.

In presence of immediate rewards and discounting, the Q-value associated with value function v is defined as $Q(s,x) = R(s,x) + \gamma \sum_{s' \in S} \mathcal{P}(s'|s,x)v(s')$, and again a necessary and sufficient condition for v to be optimal is that for all states s, $Res_v(s) = 0$.

3.4.4 Decision Making for MDPs

To make decision in an OSCO problems, one needs to compute a policy, and then makes the decision prescribed by that policy. Many techniques here exists, but all of them manipulates a value function v and aim at computing or approximating the optimal value function $v^\star$ by solving the Bellman's equation $Res_{v^\star}(s) = 0$ for all non-final states. All algorithms belong to one of the two following categories:

1. those in which the value function (or the Q-value function) is stored explicitly by a map from states to reals;

2. those in which there is an an implicit representation of the value function (or of the Q-value function), using a parametrized family of functions (linear functions, polynomials, splines...) of some attributes of states.

The simplest algorithm for solving Bellman's equation when S is finite is *value iteration*: starting from any value function v, loop endlessly over all states s, performing for each of them a Bellman update, that is, the operation $v(s) \leftarrow \max_{x \in \mathcal{X}(s)} Q(s, x)$ (there are several variants of value iteration. The one we described here is more precisely called the Gauss-Seidel Dynamic Programming algorithm [10]). Under relatively mild conditions, the value function v converges to the optimal one $v^\star$.

Value iteration is very simple, but requires to store explicitly a value function for the whole state-space, which is usually prohibitive. For example, modelling the regular instance of the S-RCPSP as an MDP, which has only 5 projects, 17 tasks, and 2 labs, requires about $2 \cdot 10^{11}$ states, which cannot feat in the RAM of today's computers.

Heuristic search algorithms[13] try to mitigate this problem by storing only a small fraction of the value function, and using an upper bound to prove that the rest of the state-space is not worth considering. This can be efficient if each transition can lead to only a small number of successor states, and if very tight upper bounds are available to prune to search space. Alas, this last requirement is hard to meet. We will provide a more comprehensive treatment of search algorithms in chapters 6 and 8, because a search algorithm will be used as a subroutine of *Amsaa*, the algorithm at the heart of this work.

While heuristic search algorithms mitigate the memory problem, many others get rid of it by a more radical approach: they manipulates implicit representations of the value function, which requires constant memory. In this approach, each state is is described as a vector of attributes. For example, such an approach was used for the stochastic multiknapsack problem in [35]. A state for this problem consist of the remaining time and the current assignment of items to bins. The chosen attributes where:

1. for each item type, the number of items of that type currently assigned in the bins;

2. for a number of different capacities interval, the number of bins for which the remaining capacity belong to that interval.

And value functions were restricted to be linear functions. Of course, it is unlikely that the optimal value function is actually a linear function of these attributes, so such methods are approximate.

Algorithms that use an implicit representation fall into two categories. On the one hand, in *approximate dynamic programming*[41], a model (e.g., a linear program) is established to compute the parameters that minimize a measure of violation of the Bellman equation. On the other hand, in *reinforcement learning*, an iterative method is chosen: states are visited in turns, and when visiting a state, parameters are updated so as to decrease the Bellman residual in that state. Of course, because parameters are manipulated, decreasing this residual might increase others, so special care as to be taken to ensure convergence.

To use either approximate dynamic programming and reinforcement learning, the major difficulty using these techniques is to choose the attributes to consider and the family of functions to use.

3.4.5 Use of MDP-based techniques

The MDP model, or some variants such as Partially Observable MDPs, is general enough to model all the kinds of OSCO problems we consider in this dissertation. However, the MDP literature contains few very convincing results on large and complex OSCO problems.

This may be because much of the work has the overly ambitious goal of computing a priori the whole optimal policy, rather than benefiting of the available time in between decisions to run further computations. There are however several research trends that claim to focus on anytime decision making, such as the Real Time Dynamic Programing literature, initiated by [2]. Yet I am not aware of any paper that actually evaluates a proposed algorithm by running simulations of the process and making anytime decisions, so it is difficult to judge how good the proposed algorithms really are.

3.5 Stochastic Programming

Similarly to MDP-based methods, Stochastic Programming (SP) is a set of techniques that aim at providing a general framework for solving stochastic optimization problem. The major difference is that the model used, Stochastic Programs (also abbreviated SPs), represents the uncertainty exogenously, while MDPs represent it endogenously. A recent book on SP is [43].

3.5.1 Models and Examples

The simplest stochastic programs are single-stage and 2-stage SPs defined as follows.

Definition 1 *A* single-stage stochastic program *(1-SP) is a tuple* $(\mathcal{X}, \Xi, \boldsymbol{\xi}, F)$*, where* $\mathcal{X}$ *is a set called the* decision space*,* $\boldsymbol{\xi}$ *is a random variable with values in* Ξ *(the* scenario space*), and* $F : \mathcal{X} \times \Xi \to \mathbb{R}$ *is a measurable function*[1]*.* Solving *a 1-SP consist of finding the decision* $x^\star \in \mathcal{X}$ *that maximizes* $\mathbb{E}\left[F(x, \boldsymbol{\xi})\right]$*.*

A Two-stage stochastic program *(2-SP) is a tuple* $(\mathcal{X}, \Xi, \boldsymbol{\xi}, Y, \mathcal{Y}, F)$ *where* $\mathcal{X}$ Ξ *and* $\boldsymbol{\xi}$ *are as before,* $\mathcal{Y}$ *maps pairs* $(x, \xi) \in \mathcal{X} \times \Xi$ *to non-empty subsets of the set* Y *(called the* 2nd stage decision space*) and* $F : \mathcal{X} \times \Xi \times Y \to \mathbb{R}$ *is a measurable function.* Solving *a 2-SP consist of finding the decision* $x^\star \in \mathcal{X}$ *that maximizes* $\mathbb{E}\left[\max_{y \in \mathcal{Y}(x,\xi)} F(x, \xi, y)\right]$*.*

Note that two-stage SPs can be reduced to single-stage SPs by defining $F'(x, \xi) = \max_{y \in \mathcal{Y}(x,\xi)} F(x, \xi, y)$. Single-stage stochastic programs are well adapted to long-term decisions

[1]we assume that $\mathcal{X}$ and Ξ are equipped with a σ-field. In this dissertation, σ-fields will never be made explicit, and measurability will be assumed whenever necessary

with heavy consequences and little possibility for reactions. For example, consider a mid-sized company that is trying to choose the location for a new warehouse. This is a significant investment, and it is likely that the possibility of opening other facilities in the future will not be considered, because this possibility is too far ahead. If the demand is stochastic, which is usually the case, then the problem of choosing the location of the warehouse can be modelled as a 1-SP. A two-stage version of the problem could be that after buying a warehouse, it is possible to rent one more somewhere else. Single-stage stochastic programs are also used for some simple portfolio management problems, in which once the portfolio is set up no trading is done anymore (this is called a "buy-and-hold" strategy).

None of the problems presented in the previous chapter can be modelled as a 2-SP, given their *online* nature: information comes little by little, and decisions are made at many points in time. Multistage stochastic programs are a generalization of the model for online problems.

Definition 2 *A* multistage stochastic program *(MSP) is defined by*

$$\max_{x_1 \in \mathcal{X}_1} \mathbb{E}\left[\max_{\boldsymbol{x}_2 \in \mathcal{X}_2(x_1,\boldsymbol{\xi}_1)} \mathbb{E}\left[\dots \mathbb{E}\left[\max_{\boldsymbol{x}_T \in \mathcal{X}_T(x_1,\dots,\boldsymbol{x}_{T-1},\boldsymbol{\xi}_1,\dots,\boldsymbol{\xi}_{T-1})} F(x_1,\boldsymbol{x}_2,\dots,\boldsymbol{x}_T,\boldsymbol{\xi}_1,\dots,\boldsymbol{\xi}_{T-1})\right]\dots\right]\right],$$

where $\boldsymbol{\xi} = \boldsymbol{\xi}_1,\dots,\boldsymbol{\xi}_T$ is a stochastic process, the functions $\mathcal{X}_1,\dots,\mathcal{X}_T$ maps their respective arguments to non-empty sets, and F is a real-valued measurable function, and $\boldsymbol{x}_t$ is a random variable that has to be independent of $\boldsymbol{\xi}_t,\dots,\boldsymbol{\xi}_{T-1}$.

There is a striking resemblance between a T-stage SP and a T-horizon MDP. The biggest difference is that in an SP, $\boldsymbol{\xi}$ is a stochastic process that is exogenous: it does not depend on the decisions x's. There are many applications of MSPs, mostly in finance (see [52] for a general survey, and [27] for pension fund applications), energy [40], and transportation [53].

3.5.2 Solving Stochastic Programs

We have described the models used in stochastic programming, but not the methods used for decision making. Almost all the literature in focused on linear, integer, and mixed-integer stochastic programs. Let X be a vector space, which is a minor requirement. Then a multistage SP is linear if (1) the function F is linear, and (2) the function X_t is of the form:

$$\mathcal{X}_t(x_1,\dots,x_{t-1},\xi_1,\dots,\xi_{t-1}) = \left\{x \in \mathbb{R}^{+d} \middle| A_{t,1}(\xi_1)x_1 + \dots + A_{t,t}(\xi_1,\dots,\xi_{t-1})x_t = b_t(\xi_1,\dots,\xi_{t-1})\right\}$$

where the A's are matrix-valued functions and the b's are vector valued functions. The SP is integer if in addition, x has to belong to $\mathbb{Z}^d$, and mixed-integer if some but not all components are restricted to be integers.

When $\boldsymbol{\xi}$ has finite support, one can enumerate all possible scenarios $\xi^1,\dots,\xi^n$ with their probabilities $p_1,\dots,p_n$. Then, a linear (resp. integer, mixed-integer) MSP can be reduced to a standard (i.e., non-stochastic) linear program (LP) (resp. integer program (IP), mixed-integer program (MIP)) by the following technique. First, consider a given scenario ξ. Then the *offline problem* consisting of computing the optimal sequence of decision for scenario ξ is naturally expressed as the following LP (resp. IP, MIP, by adding the necessary integrality constraints), where the ξ arguments have been omitted:

$$\begin{array}{lllllll}
\max & F(x,\xi) & & & & & \\
\text{s.t.} & A_{1,1}x_1 & & & & = b_1 \\
& A_{2,1}x_1 & +A_{2,2}x_2 & & & = b_2 \\
& & & \dots & & \\
& A_{T,1}x_1 & +A_{T,2}x_2 & +\dots & +A_{T,T}x_T & = b_T \\
& x \geq 0 & & & &
\end{array}$$

To solve the stochastic problem, introduce decision variables x_t^s to be the decision variable made at stage t in scenario s. The LP then consist of a replication of the above constraints for each scenario, in addition to the following constraints: if ξ^1 and ξ^2 and t are such that

$$\xi_1^1 = \xi_2^2, \quad \dots \quad, \xi_t^1 = \xi_t^2,$$

then $x_{t+1}^1 = x_{t+1}^2$. This reflects the fact that the random variable $\boldsymbol{x}_{t+1}$ has to be independent of $\boldsymbol{\xi}_{t+1}, \dots, \boldsymbol{\xi}_{T-1}$, and is called a *non-anticipativity constraint.* Most algorithms in stochastic programming aim at solving this large LP (resp. IP, MIP), called the *deterministic equivalent* of the stochastic program.

When $\boldsymbol{\xi}$ does not have a finite support, there is a diversity of methods for two-stage SPs (e.g., a gradient-ascent-like algorithm where at each iteration a direction with a high probability of being an ascending direction is computed). For multistage SPs however, work is almost exclusively based on approximating the distribution by a finite-support one. When this approximation is done by sampling, this is called the Sampling Average Approximation (SAA) method [42, ch. 6].

3.5.3 Extension to Stoxuno Problems

In the MSP model, the decision $\boldsymbol{x}_t$ made at stage t can depend of $\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_{t-1}$. Therefore, the MSP model represent the *observations of the uncertainty* has an exogenous stochastic process, and, as such, cannot be used for stoxuno problems or problems with purely endogenous uncertainty. However, a few papers have presented stochastic programming inspired techniques for Stoxuno problems. In particular, Goel and Grossman have applied a general technique the used on offshore gas field developments [51]. Later, they provided a general model in [24] and applied it to a capacity expansion problem and to a multiproduct lot sizing problem with substitutions.

It is to be noted that this general model is presented in [24] as a general model for multistage optimization problem with endogenous uncertainty. However, a concept of scenario appear in the model, and the model introduces binary variables that are one when a specific random variable has been observed. Therefore, it really is a model for stoxuno problems.

Unfortunately, these models uses a number of disjunctions that is quadratic in the number of scenarios, making the search space huge and the linear relaxation poor, and therefore this technique is not very scalable. The before-mentioned capacity expansion problem had only four scenarios. The lot sizing problems was solved (not to proved optimality) with up to 64 scenarios and 7 stages, and the model required for that problem already had more than 3,000 binary variables, and only three nodes of the search tree was explored in more than 3 hours.

3.6 Anticipatory Algorithms for Online Stochastic Combinatorial Optimization

While MDP algorithms and SPs can be used in an OSCO setting, they tend to be studied and used mostly in an a priori setting, were computations are done only prior to the first decision. In contrast, a line of research was devoted to the study of algorithms for OSCO problems designed to use the computation time available at each decisions. This line of research has focused on the development of *Anticipatory Algorithms* (AAs).

The class of Anticipatory Algorithms is a broad class of decision-making algorithms for stoxuno online stochastic combinatorial optimization problems that share two characteristics: first, they sample scenarios, and second, they make some use of the *anticipative relaxation* of the problem, which is the relaxation consisting of ignoring the order in which observations and decisions are interleaved.

The most natural of these algorithms consist of evaluating each scenario of a sample on each decision, in an offline fashion. That is, for each pair scenario-decision, one computes the best achievable score reachable by starting with the given decision and making all the remaining ones with knowledge of the scenario. The chosen decision is then the one with the highest average offline score. This algorithm has been presented under various names, such as Expectation in [7], Forward Sampling in [3], and Hindsight Optimization in [18]. It belongs to the class of *one-step anticipatory algorithms* (*1s-AA*), that ignores the alternation of decisions and observations immediately after the current decision. We will refer to it as *1s-AA-E*. Other one-step anticipatory algorithms include *1s-AA-C*, or Consensus, in which the chosen decision is the one that is the most frequently offline-optimal for the scenarios in the sample, and *1s-AA-R*, or Regret, that uses an approximation of the offline scores.

These are efficient because offline scores can be computed with ad hoc solvers, leveraging the large body of deterministic combinatorial optimization, and scale well because no computation requires to deal with several scenarios at the same time. Although they are heuristic algorithms, the quality of the decisions they make has proved to be surprisingly good on a number of applications.

For example, it works extremely well in the stochastic multiknapsack problem presented in section **??**, as shown initially by [3], and later refined in [26]. *1s-AA* algorithms were also used with success on a packet scheduling application [7], in dynamic vehicle routing [6], in online mechanism design [39], and job-shop scheduling [48]. Many of these results are summarized in the book [50].

The idea of using the computation time in between decisions, rather than only at the beginning, partly explain the success of these algorithms. Indeed, not only it enables the use of more computing power, but it is also much more tolerant to inaccuracies in the model. For example, search algorithms for MDPs compute partial policies, i.e., policies that prescribe an action only for some of the states. Even if the algorithm guarantees that the returned policy is defined by all states reachable by following it, inaccuracies in the model can lead the system to arrive in a state for which no action is prescribed. To get rid of this problem, *1s-AA* are *memoryless*: each time a new decision is needed, the algorithm is called from scratch.

Unfortunately, one-step anticipatory algorithms produce sub-optimal decisions, and provide little means to bound the optimality gap. One possible way to address the first problem is to use one of the very recent *gap reduction technique* [21] which are based on learning how the anticipative relaxation compares the optimum, so as to be able to correct the inaccuracies introduced by the relaxation. Very good results were obtained on the S-RCPSP. Still, these techniques remaining suboptimal.

This thesis is in the direct continuation of the former work on AAs for OSCO problems. The novelty will be the introduction of a convergent algorithm, that is, that make the optimal decision if enough computation time is permitted. On the S-RCPSP for example, we obtained higher quality decisions than by using gap reduction techniques, at the price of a higher computational cost.

3.7 Summary and Comparison

We have presented six techniques that aim at solving optimization problems under uncertainty:

1. Control theory
2. Robust optimization
3. Online algorithms
4. MDP-based methods
5. Stochastic programming
6. Anticipatory algorithms for OSCO

With respect to the goals of this thesis, I dismissed the first three techniques. Control theory is not applicable to our problems because of the discrete nature of time in OSCO problems and because the complexity to model the controlled system. Robust optimization focuses on a different objective function than the one we want to use in this work, which is the maximal expected utility. Online algorithms (in the algorithmic theory sense) are very problem-dependent and do not seem applicable to weakly structured real-world problems.

The three last ones gets much closer to what we want. MDP-based techniques are interesting, but there are only scalable when they use well-chosen parametrized family of functions of vector of well-chosen state attributes, making them very hard to use for highly combinatorial problems. Stochastic programming techniques are effective in the pure exogenous and linear case, but the literature contain little very convincing results on large integer multistage problems, and the Stoxuno variants of SPs scale very badly with the number of scenarios. Finally, anticipatory algorithms are effective, simple and scalable. Unfortunately the ones that existed so far made suboptimal decisions, and did not provide a good measure of the optimality gap.

The work that I will present in the subsequent chapters is based on the recognition that stochastic programming, MDP-based techniques, and anticipatory algorithms have complementary strengths and limitations. *Amsaa* is an algorithm that exploit the synergies between the ideas from this three fields.

Chapter 4

A Model for Exogenous Online Stochastic Optimization Problems

The previous chapter presented two existing models for stochastic optimization problems: Markov Decision Processes (MDP) and Stochastic Programs (SPs). MDPs are expressive enough to model OSCO problems regardless of the endogeneity or exogeneity of their uncertainty. Stochastic Programs (SPs), on the other hand, are designed exclusively for purely exogenous problems. We now introduce a new model, called Exogeneous Markov Decision Process (X-MDP), that is well suited for problems with exogenous uncertainty, regardless of the exogeneity or endogeneity of the observations. This will be the model used for the rest of this dissertation.

Although X-MDPs have the same expressing power than MDPs, they are worth introducing, because they enable the design of algorithms take make use of the exogeneity. We will at the end of this chapter provide arguments about why there are computational advantages to using X-MDPs model instead of MDPs.

4.0.1 Exogenous Markov Decision Processes

An X-MDP $(S, s_0, F, X, \perp, \mathcal{X}, f, \boldsymbol{\xi}, \mu_{\xi}, \tau)$ consists of:

- a state space S, an initial state $s_0 \in S$, and a set of final states $F \subseteq S$.
- a decision space X containing a decision $\perp$ (denoting no action) and a function $\mathcal{X} : S \to X$ returning the set of feasible decisions in a given state such that $\forall s \in S, 0 < \#\mathcal{X}(s) < \infty$ and that $\forall s \in F, \mathcal{X}(s) = \{\perp\}$.
- a bounded reward function $f : F \to \mathbb{R}$.
- a random variable $\boldsymbol{\xi}$ taking values from a scenario space Ξ whose distribution is μ_{ξ}.
- a (deterministic) transition function $\tau : S \times X \times \Xi \to S$ satisfying $\forall s \in S,\ \forall \xi \in \Xi,\ \tau(s, \perp, \xi) = s$.

Running an X-MDP consists of first sampling a realization ξ of the random variable $\boldsymbol{\xi}$: The realization ξ is not known to the decision maker and is only revealed progressively through observed outcomes of the transitions. Starting in state s_0, the decision maker takes a decision, observes the outcome of the transition, and repeats the process. For a state s and a decision x, the next state becomes $\tau(s, x, \xi)$. The alternation of decisions and state updates defines a trajectory

$$s_0 \xrightarrow[\xi]{x_0} s_1 \xrightarrow[\xi]{x_1} \dots \xrightarrow[\xi]{x_{t-1}} s_t$$

satisfying (1) $x_i \in \mathcal{X}(s_i)$ and (2) $s_{i+1} = \tau(s_i, x_i, \xi)$ for all i. A trajectory corresponding to an unspecified scenario is denoted by

$$s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} \dots \xrightarrow{x_{t-1}} s_t.$$

A scenario ξ is *compatible with a trajectory* $s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} \dots \xrightarrow{x_{t-1}} s_t$ if $\tau(s_i, x_i, \xi) = s_{i+1}$ for all $i < t$. The set of scenarios compatible with the trajectory $s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t$ is denoted by $\mathcal{C}\left(s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t\right)$. A scenario is *compatible with a state* s if it is compatible with a trajectory from s_0 to s. $\mathcal{C}(s)$ denotes the set of scenarios compatible with state s.

We make similar assumptions for X-MDPs than for MDPs. In particular, we also assume a finite horizon, that is the existence of a stage T such that s_T is final regardless of the decisions and the scenario realization. The objective also consists of maximizing $\mathbb{E}\left[f(\boldsymbol{s}_T)\right]$, which is always defined if f is bounded. Finally, we also impose a *Markovian property* for X-MDPs, which ensures the dominance of Markovian policies. In the context of X-MDPs, this property becomes:

$$\text{for all trajectory } s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t, \quad \mathcal{C}\left(s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t\right) = \mathcal{C}\left(s_t\right). \tag{4.1}$$

It is easy to enforce this property in practice: simply include all past observations into the current state. An elementary but important corollary of this assumption is that conditional probabilities on the past trajectory are identical to conditional probabilities on the current state, i.e.,

$$\forall A \subseteq \Xi, \quad \mathbb{P}\left(\boldsymbol{\xi} \in A \,\middle|\, \boldsymbol{\xi} \in \mathcal{C}\left(s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t\right)\right) = \mathbb{P}\left(\boldsymbol{\xi} \in A \mid \boldsymbol{\xi} \in \mathcal{C}\left(s_t\right)\right),$$

Hence, sampling scenarios conditionally on the current state is equivalent to sampling scenarios conditionally on the past trajectory.

4.0.2 Offline Deterministic Problems Associated with X-MDPs

X-MDPs enjoy a fundamental property: they naturally exhibit an underlying *deterministic* and *offline* problem that has no counterpart in MDPs.

Definition 3 *The* offline value *of state s under scenario ξ, denoted by $\mathcal{O}(s, \xi)$, is the largest reward of a final state reachable from state s when $\boldsymbol{\xi} = \xi$. It is defined recursively by:*

$$\mathcal{O}(s, \xi) = \begin{cases} f(s) & \textit{if } s \textit{ is final,} \\ \max_{x \in \mathcal{X}(s)} \mathcal{O}(\tau(s, x, \xi), \xi) & \textit{otherwise.} \end{cases}$$

Consider the instance presented in Section **??**. If ξ_s and ξ_l denote the scenarios in which A.2 is short and long respectively, then $\mathcal{O}(s_0, \xi_s) = 17$ and $\mathcal{O}(s_0, \xi_l) = 15$, as shown on Figure 2.3.

4.0.3 Value Functions and Optimality for X-MDPs

Like for MDPs, it is possible to define the value of a policy for an X-MDP. Let A be an X-MDP and $\pi : S \to X$ be a policy for A. Consider a past trajectory $s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t$, not necessarily generated by π. Recall that for any trajectory s_T is final. Therefore the expected value obtained by following π after this past trajectory is well defined and is denoted by

$$v_\pi \left(s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t \right).$$

Now remember that

$$\forall A \subseteq \Xi, \quad \mathbb{P}\left(\boldsymbol{\xi} \in A \,\middle|\, \boldsymbol{\xi} \in \mathcal{C}\left(s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t \right) \right) = \mathbb{P}\left(\boldsymbol{\xi} \in A \mid \boldsymbol{\xi} \in \mathcal{C}\left(s_t \right) \right).$$

Therefore the (random) future trajectory following π only depends on s_t and not on earlier states and decisions. As a consequence, we can define

$$v_\pi(s_t) = v_\pi \left(s_0 \xrightarrow{x_0} \dots \xrightarrow{x_{t-1}} s_t \right).$$

A policy $\pi^\star$ is optimal if $v_{\pi^\star}(s_0)$ maximizes $v_\pi(s_0)$ over all policies π. For simplicity, in various sections of this paper, $v_{\pi^\star}(s)$ is denoted by $v(s)$.

4.0.4 Benefits of X-MDPs over MDPs

Although X-MDPs can be turned into equivalent MDPs, they have two computational advantages: the existence of offline, deterministic problems and the ability to use exterior sampling.

Offline Problems. The existence of offline problems is one of the reasons for the success of anticipatory algorithms for online stochastic combinatorial optimization [37, 50]. Contrary to MDPs, X-MDPs naturally reveal underlying offline problems, which can then be exploited by algorithms such as *Amsaa*. Indeed, computing $\mathcal{O}(s, \xi)$ is a deterministic combinatorial problem for which a variety of advanced optimization techniques may be applicable. Section 6.4.3 shows how these offline values guide the search in finding optimal policies and the experimental results will demonstrate their fundamental role in achieving good performance. In this paper, as was already the case for [50], we assume that we have at our disposal a black-box to solve these offline problems or to compute good upper bounds of $\mathcal{O}(s, \xi)$ quickly.

Exterior Sampling. In MDPs, one can only sample outcomes of state-decision pairs. In contrast, in X-MDPs, a set of scenarios can be sampled a priori and independently of the decisions. Section 6.2.2 will explain why this ability makes it possible to reduce the number of scenarios needed to find high-quality policies.

4.0.5 Modeling the Stochastic RCPSP as an X-MDP

We now sketch the modeling of the S-RCPSP as an X-MDP. Because tasks cannot be preempted or aborted, the X-MDP states correspond to times when at least one laboratory is available. More precisely, a state contains:

- the current time;
- the set of currently running tasks with their start times (but without lab assignment);
- the set of all past observed task realizations.

The Markov property for X-MDPs is satisfied, since all past observations are stored. There are two types of decisions: (1) scheduling a given task on one of the available laboratories or (2) waiting. In both cases, the successor $\tau(s, x, \xi)$ corresponds to the next time a decision must be taken. This can be the current time when several laboratories are available for schedule, since tasks are scheduled one at a time in our model. In all other cases, the next state corresponds to a later time. The model contains a symmetry-breaking constraint, using an ordering on the projects. If a task of project k is scheduled at time t in state s, none of the tasks in projects $1..k-1$ can be scheduled in a descendent s' of s if state s' is associated with time t too.

Chapter 5

One-Step Anticipatory Algorithms

Among the many existing algorithms for OSCO problems, *one-step anticipatory algorithms* are some of the simplest; and yet have proved to be very efficient in a number of cases. These algorithm work by ignoring the non-anticipativity constraints that arise from the alternation of decisions and observations — except for the decision to be made immediately. In this chapter, we will first decribe the simplest of these algorithms and quicly describes a few variants. We then review some results on a variety of applications obtained by diverse research groups. To understand why such good results can be obtained based on such a crude approximation, we then present a sufficient condition for this algorithm to perform well, and discuss this condition.

5.1 Algorithms

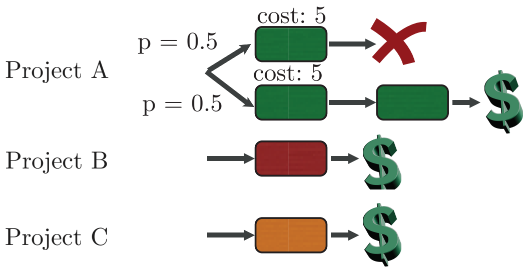
Consider an X-MDP $\mathcal{A} = (S, s_0, F, X, \perp, \mathcal{X}, f, \boldsymbol{\xi}, \mu_{\xi}, \tau)$ with a time horizon T. Remember that $\mathcal{O}(s, \xi)$ is the *offline value* of state s in scenario ξ, that is, the maximal value that can be obtained by a clairvoyant decision maker in state s when the scenario is $\boldsymbol{\xi} = \xi$. For compactness, we denote by $\mathcal{O}(s, x, \xi) = \mathcal{O}(\tau(s, x, \xi), \xi)$ for $x \in \mathcal{X}(s)$.

We call *One-Step Anticipatory Algorithms* any algorithms that ignores all non-anticipativity constraints but the ones that apply to the current decision. A few different algorithms can be derived from that simplification, and the most natural is `MakeDecisionOneStepExpectation` whose pseudo-code follows.

```
Function MakeDecisionOneStepExpectation(X-MDP $\mathcal{A}$, State $s_t$)
    Sample scenarios $\xi^1 \dots \xi^m$, iid from the distribution of $\boldsymbol{\xi}$ conditionnaly on $s_t$.
    foreach $x \in \mathcal{X}(s_t)$ do
        $g(x) \leftarrow \frac{1}{m} \sum_{i=1}^{m} \mathcal{O}(s_t, x, \xi^i)$
    $x_t \leftarrow \text{argmax}_{x \in \mathcal{X}(s_t)} g(x)$
```

In this code, m is a parameter. A few variants exists. For example, if $\mathcal{X}(s_t)$ is large, the above algorithm is unpractical. Instead, one can use the *consensus* variant, defined as follow:

```
Function MakeDecisionOneStepConsensus(X-MDP $\mathcal{A}$, State $s_t$)
    Sample scenarios $\xi^1 \dots \xi^m$, iid from the distribution of $\boldsymbol{\xi}$ conditionnaly on $s_t$.
    foreach $x \in \mathcal{X}(s_t)$ do
        $g(x) \leftarrow \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}_{\operatorname{argmax} \mathcal{O}(s_t, \bullet, \xi^i)}(x)$
    $x_t \leftarrow \operatorname{argmax}_{x \in \mathcal{X}(s_t)} g(x)$
```



There are two labs, one immediately available, one available from time 1 All tasks has a duration of 2. All costs are zero, expect for task A1 that costs 5 The revenues as a function of the completion time of projects are:

Project	≤ 2	3	4	5	≥ 6
A	45	45	45	22	0
B	18	9	0	0	0
C	15	8	1	0	0

Figure 5.1: An instance of the S-RCPSP on which *1s-AA* is sub-optimal

where $\mathbf{1}_{\operatorname{argmax} \mathcal{O}(s_t, \bullet, \xi^i)}(x)$ is the function that has value one if $x = \operatorname{argmax}_{y \in \mathcal{X}(s_t)} \mathcal{O}(s_t, y, \xi^i)$.

While `MakeDecisionOneStepExpectation` needs $m\#\mathcal{X}(s_t)$ calls to the function $\mathcal{O}$, counting the number of scenarios for which a decision is offline-optimal, like in `MakeDecisionOneStepConsensus`, requires to solve only m offline problems. Note that this assumes that the offline solver actually computes the optimal solution, not merely its value, and that ties are not considered.

Function `MakeDecisionOneStepExpectation` being so natural, it has been introduced several times by different research groups. For example, it is called *expectation* in [50], *forward sampling* in [3], and *hindsight optimization* in [18].

In the rest of this chapter, we focus denote by *1s-AA-E* the algorithm `MakeDecisionOneStepExpectation`, and this will be the major focus.

5.2 Suboptimality Example

In general, *1s-AA-E* makes sub-optimal decisions. Figure 5.1 depicts an instance of the stochastic scheduling problem that will illustrate why. Consider the initial decision to be made at time 0. The feasible decisions are to start the first task of project A, B, or C. Because there are only two possible scenarios(project A fails or project A succeeds), the sample distribution will quickly converge to the true distribution, with probability $1/2$ for each scenarios. Let's compute what *1s-AA-E* will do if the sample distribution equals the true distribution. Figure 5.2 depicts the offline schedules for the two scenarios and the three feasible initial decisions. The highest average offline score is obtained by scheduling project B at time 0, and therefore this is the decision made by *1s-AA-E*. Unfortunately, this is suboptimal. Indeed, the two offline optimal schedules that correspond to the decisions of scheduling B first differ at time 1: if A succeeds, it should be scheduled, and if not, C should be

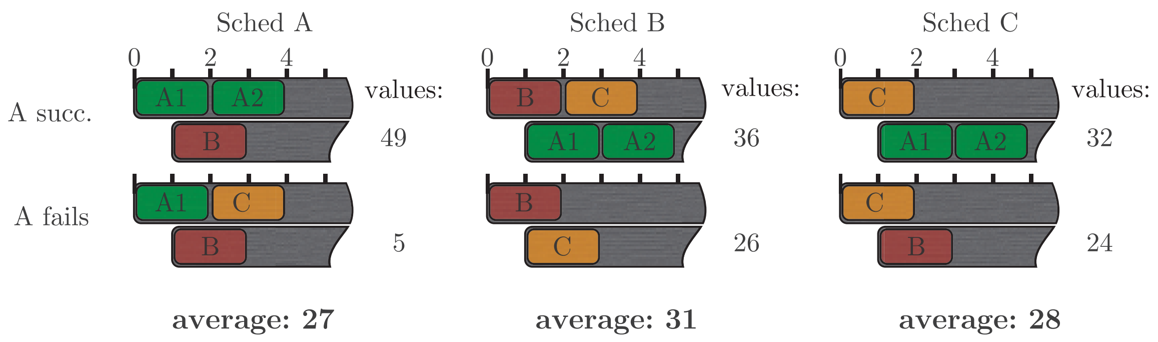


Figure 5.2: Offline optimal scedules and their values

scheduled. Unfortunately, at time 1 the decision maker will not have made any observation that spcifies which scenario is happening. Therefore, it is not possible for a non-clairvoyant decision maker to achieve this average score of 31. We say that these two offline schedules *violates a non-anticipativity constraint.*

Let's assume *1s-AA-E* was used at time 0, and task B was started. What decision will be made at time 1? At that time A and C can be scheduled. The reader can compute that the average offline score of scheduling A is $(1/2(36 + 14) = 25$, and those of scheduling C is $1/2(26 + 26) = 26$. Therefore, *1s-AA-E* will schedule C at time 1, and then never schedules A: on this instance, *1s-AA-E* always gets a score of 26 if the sample size if large enough. This is suboptimal: condider the following online policy. Start A1 at time 0 and B at time 1. Then at time 2, if A1 failed, shedule C, otherwise schedule A2. This policy, that does not violate any non-anticipativity constraints, achieves an expected value of $1/2(49 + 5) = 27$, which is higher than 26.

5.3 Review of Earlier Empirical Results

One-step anticipatory algorithms have been used on many applications, and often with very satisfying results. The book [50] is a comprehensive source of such results, and here we want to quote some of them.

[16] consider the following packet scheduling problem. A router receives a set of packets at each time step and must choose which packet to serve. Packets can be served only for a limited time and they are characterized by a value. The goal is to maximize the values of the served packets. The packet distributions are specified by Markov models whose states specify arrival frequencies for the packet type.

Online reservation systems [3] are another application. Customers place requests in real time for some service at a fixed date. The resources are modeled by a multiknapsack constraint. (Think of tour operators requesting rooms in hotels for a group: the choice of a specific hotel is not pertinent for the group but all group members must be allocated to the same hotel). Customers must be immediately notified of acceptance or rejection of their requests, and accepted requests must be satisfied. Accepted requests must also be assigned to a specific resource at reservation time and this choice cannot be reconsidered. The goal is to maximize the profit of the served requests which come from different types with different characteristics and arrival frequencies.

Online multiple vehicle routing with time windows [4] captures an important class of applications arising in transportation and distribution systems. In these problems, a fleet of vehicles serve clients which are located in many different locations and place request for service in real-time in specific time windows. Clients must be immediately notified of acceptance or rejection of their requests, and all accepted requests must be satisfied. Routing decisions however can be delayed if necessary. The goal is to maximize the number of satisfied requests.

5.4 Analysis of the Anticipatory Algorithm

It is clear that *1s-AA-E* is necessarily suboptimal, even with infinitely many scenarios. However, experimental results have been surprisingly good, especially with the *Regret* algorithm [5, 26] which is an efficient way of implementing step 2. Our goal in this chapter is to demystify these results by providing a theoretical analysis of these algorithms. Section **??** describes the model and the algorithm. Section 5.4 analyses the performance of the online anticipatory algorithm and isolates two fundamental sources of error: a sampling error and a quantity called the *global anticipatory gap* which is inherent to the problem. Section **??** shows how to bound the anticipatory gap theoretically and experimentally. Section 5.7 analyzes the effect of approximating the optimization problem. Section 5.8 compares the anticipatory gap to the expected value of perfect information. Section 5.9 presents directions for future research.

The *Expected Value of the Clairvoyant* (*EVC*) of the problem is the expected value that a decision-maker knowing the future before hand can achieve. That is, $EVC = \mathbb{E}\left[\mathcal{O}(s_0, \boldsymbol{\xi})\right]$, where s_0 is the initial state. For a given decision-making algorithm, we call *Expected Value*, $EV = \mathbb{E}\left[f(\boldsymbol{s_T})\right]$, the expected value achieved by making each decision with the considered algorithm.

The goal of this analysis is to bound $EVC - EV$, which we call the *expected global loss* (*EGL*), when decisions are made using *1s-AA-E*. Note that *EVC* is a constant, so maximizing *EV* is the same thing as minimizing *EGL*. When the considered algorithm is *1s-AA-E* with m scenarios per decision, we denote the expected value *EV*(*1s-AA-E*(m)). Relations involving *EV* without further precisions are valid for any decision-making algorithm.

5.4.1 Local and Global Losses

We first show that the *EGL* is the sum of the expected losses of the stages.

Definition 4 *Let s_t be a state. The* expected local loss *of decision $x \in \mathcal{X}(s_t)$ is defined as*

$$\Delta(s_t, x) = \mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi}) - \mathcal{O}(s_t, x, \boldsymbol{\xi}) | s_t\right].$$

In the expected sense, the global loss is the sum of local losses.

Lemma 1 (Global Loss = Sum of Local Losses) *For any decision process x,*

$$EGL = \sum_{t=0}^{T-1} \mathbb{E}\left[\Delta(\boldsymbol{s}_t, \boldsymbol{x}_t)\right]$$

Proof. We first modify the X-MDP into an equivalent one that has initial state s_{-1} in which there is only one possible decision x_{-1}, and such that for any scenario ξ, $\tau(s_{-1}, x_{-1}, \xi) = s_0$. Therefore both s_{-1} and s_0 are deterministic.

Let $\boldsymbol{C}_t$ be the random variable $\mathcal{O}(\boldsymbol{s}_t, \boldsymbol{x}_t, \boldsymbol{\xi})$ and $A_t = \mathbb{E}\left[\boldsymbol{C}_t - f(\boldsymbol{s}_T)\right]$. Then $A_T = 0$ (remember that $\boldsymbol{s}_T$ is final, so $\boldsymbol{x}_T$ is necessarily $\perp$, the null decision). Now, for $-1 \leq t < T$:

$$\begin{aligned}
A_t &= \mathbb{E}\left[\boldsymbol{C}_t - \boldsymbol{C}_{t+1} + \boldsymbol{C}_{t+1} - f(\boldsymbol{s}_T)\right] \\
&= \mathbb{E}\left[\boldsymbol{C}_t - \boldsymbol{C}_{t+1}\right] + A_{t+1} \\
&= \mathbb{E}\left[\mathcal{O}(\boldsymbol{s}_t, \boldsymbol{x}_t, \boldsymbol{\xi}) - \mathcal{O}(\boldsymbol{s}_{t+1}, \boldsymbol{x}_{t+1}, \boldsymbol{\xi})\right] + A_{t+1} \\
&= \mathbb{E}\left[\mathcal{O}(\tau(\boldsymbol{s}_t, \boldsymbol{x}_t, \boldsymbol{\xi}), \boldsymbol{\xi}) - \mathcal{O}(\boldsymbol{s}_{t+1}, \boldsymbol{x}_{t+1}, \boldsymbol{\xi})\right] + A_{t+1} && \text{by definition of } \mathcal{O}(s, x, \xi)) \\
&= \mathbb{E}\left[\mathcal{O}(\boldsymbol{s}_{t+1}, \boldsymbol{\xi}) - \mathcal{O}(\boldsymbol{s}_{t+1}, \boldsymbol{x}_{t+1}, \boldsymbol{\xi})\right] + A_{t+1} && \text{because } \boldsymbol{s}_{t+1} = \tau(\boldsymbol{s}_t, \boldsymbol{x}_t, \boldsymbol{\xi}) \\
&= \mathbb{E}\left[\Delta(\boldsymbol{s}_{t+1}, \boldsymbol{x}_{t+1})\right] + A_{t+1} && \text{by definition of } \Delta(s, x)).
\end{aligned}$$

Finally $EGL = EVC - EV = \mathbb{E}\left[\mathcal{O}(s_0, \boldsymbol{\xi})\right] - \mathbb{E}\left[f(\boldsymbol{s}_T)\right] = \mathbb{E}\left[\mathcal{O}(s_{-1}, x_{-1}, \boldsymbol{\xi})\right] - \mathbb{E}\left[f(\boldsymbol{s}_T)\right] = A_{-1}$. It follows that $EGL = \sum_{t=-1}^{T-1} \mathbb{E}\left[\Delta(\boldsymbol{s}_t, \boldsymbol{x}_t)\right]$, and the lemma holds because $\Delta(s_{-1}, x_{-1}) = 0$. □

5.4.2 Decomposition of the Local Loss

We now show that the local loss at a state s_t consists of a sampling error and the anticipatory gap.

Definition 5 *The* anticipatory gap *of a state s_t is defined as*

$$\Delta_g(s_t) = \min_{x \in \mathcal{X}(s_t)} \Delta(s_t, x).$$

The choice error *of x wrt s_t is defined as*

$$\Delta_c(s_t, x) = \Delta(s_t, x) - \Delta_g(s_t).$$

Note that the local anticipatory gap depends only on s, and not on the decision process $\boldsymbol{x}$. An equivalent definition is

$$\mathbb{E}\left[\max_{x \in \mathcal{X}(s_t)} \mathcal{O}(s_t, x, \boldsymbol{\xi})\right] - \max_{x \in \mathcal{X}(s_t)} \mathbb{E}\left[\mathcal{O}(s_t, x, \boldsymbol{\xi})\right].$$

This expression shows that this gap can be interpreted as the cost of commuting of $\mathbb{E}$ and max. We now bound $\Delta_c(s_t, x)$.

Lemma 2 (Sampling Error) *Let s_t be a state, and $\boldsymbol{x}_t$ be the random decision computed by* 1s-AA-E *using m samples per decision in that state. Let $x^\star$ be* argmax $\mathbb{E}\left[\mathcal{O}(s_t, x, \xi)\right]$ *(break ties arbitrarily). Then*

$$\mathbb{E}\left[\Delta_c(s_t, \boldsymbol{x}_t)\right] \leq \sum_{x \in \mathcal{X}(s_t)} \Delta_c(s_t, x) \exp\left(\frac{-m\Delta_c(s_t, x)^2}{2\sigma(s_t, x)^2}\right),$$

where $\sigma(s_t, x)$ is the standard deviation of $\mathcal{O}(s_t, x, \boldsymbol{\xi}) - \mathcal{O}(s_t, x^\star, \boldsymbol{\xi})$ conditionnaly on $\boldsymbol{\xi} \in \mathcal{C}(s_t)$.

Proof. Here all probabilities and expectations are implicitly conditional on $\boldsymbol{\xi} \in \mathcal{C}(s_t)$ when necessary. The left-hand side can be decomposed as

$$\mathbb{E}\left[\Delta_c(s_t, x_t)\right] = \sum_{x \in \mathcal{X}(s_t)} \Delta_c(s_t, x)\mathbb{P}(x_t = x).$$

Due to the argmax in `MakeDecision`, the event $x_t = x$ implies $\forall x' \in \mathcal{X}(s_t),\ g(x') \leq g(x)$. Therefore $\mathbb{P}\left(x_t = x\right) \leq \mathbb{P}\left(g(x) \geq g(x^\star)\right)$. Since f is bounded, $\mathcal{O}(s_t, x, \xi) - \mathcal{O}(s_t, x^\star, \xi)$ has a finite expectation and variance. Now,

$$g(x) - g(x^\star) = \frac{1}{m}\sum_{i=1}^{m}\Big(\mathcal{O}(s_t, x, \xi^i) - \mathcal{O}(s_t, x^\star, \xi^i)\Big)$$

and, by the central limit theorem, this difference is normally distributed for m large enough, with mean $-\Delta_c(s_t, x)$ and variance $\frac{1}{m}\sigma(s_t, x)^2$. Finally, if $X \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu < 0$, then $\mathbb{P}\left(X \geq 0\right) \leq \exp\left(-\frac{\mu^2}{2\sigma^2}\right)$ (Chernoff bound). □

5.4.3 Performance of the Algorithm

We now assemble the previous results.

Definition 6 *Let A be a decision making algorithm. We define the* Global Anticipatory Gap *of A to be*

$$GAG(A) = \mathbb{E}\left[\sum_{t=1}^{T} \Delta_g(s_t)\right],$$

and we define the sup-GAG *and the* inf-GAG *to be respectively the supremum and the infemum of $GAG(A)$ over all decision-making algorithms A.*

Theorem 1 *For any X-MDP there is a constant K such that*

$$EGL(\text{1s-AA-E}(m)) \leq GAG(osaaE(m)) + O(e^{-Km}).$$

Proof. We have

$$EGL = \sum_{t=1}^{T} \mathbb{E}\left[\Delta(s_t, x_t)\right] \leq \sum_{t=1}^{T}\Big(\mathbb{E}\left[\Delta_g(s_t)\right] + \mathbb{E}\left[\Delta_c(s_t, x_t)\right]\Big) = GAG(\mathit{1s\text{-}AA\text{-}E}(m)) + \sum_{t=1}^{T} \mathbb{E}\left[\Delta_c(s_t, x_t)\right],$$

and the global sampling error satisfies

$$\sum_{t=1}^{T} \mathbb{E}\left[\Delta_c(s_t, x_t)\right] \leq T\#XF_{max}e^{-Km}.$$

□

Corollary 1 *We have the two following inequalities to bound the performance of the anticipatory algorithm.*

$$EGL(\text{1s-AA-E}(m)) \leq s\text{-}GAG + O(e^{-Km})$$
$$OPT - \mathbb{E}\left[f(\boldsymbol{s}_T)\right] \leq s\text{-}GAG - i\text{-}GAG + O(e^{-Km})$$

Proof. Because *1s-AA-E*(m) is a non-anticipative process, we have $GAG($*1s-AA-E*$(m)) \leq s\text{-}GAG$. This proves the first inequality. The second one comes from the fact that for any non-anticipative decision process, we have $EGL(x) \geq GAG(x)$. Indeed, by definition of the local anticipatory gap, for any state s_t and decision x_t we have $\Delta(s_t, x_t) \geq \Delta_g(s_t)$. Summing on the trajectories, we get

$$\mathbb{E}\left[\sum_{t=1}^{T} \Delta(s_t, x_t)\right] \geq \mathbb{E}\left[\sum_{t=1}^{T} \Delta_g(s_t)\right],$$

and we know by lemma **??** that the left-hand side is $EGL(x)$. As this inequality holds for an optimal decision process, we get that $i\text{-}GAG \leq EVC - OPT$. Therefore, we get

$$i\text{-}GAG \leq EVC - OPT \leq s\text{-}GAG,$$

and inequality follows. □

An important consequence of this theorem is that the sampling error can be made smaller than some constant a by choosing $m \geq {}^{1}\!/\!_{K} \log\left({}^{1}\!/\!_{a} T \# X F_{max}\right)$. [46] argues that the SAA method does not scale to multistage problems, because the number of samples to achieve a given accuracy grows *exponentially* with T. The anticipatory algorithm only requires m to grow *logarithmically* with $T|X|$, which makes them highly scalable. Of course, they only produce high-quality decisions when the anticipatory gap is small, an issue discussed later in the chapter.

5.5 Bounding the Global Anticipatory Gap

To illustrate the concept of sup-GAG, and hint why this sup-GAG may be low on some applications, we show how to bound $s\text{-}GAG$ for a simplified version of the packet scheduling problem. Suppose that there are k types of packets whose values are $v_1 < \ldots < v_k$ respectively. At each step from 1 to $T-1$, a packet of type i arrives with probability p_i. All these random variables are independent. Each packet has a lifetime of 2, meaning a packet received at time t can be scheduled either at time t or at time $t+1$. The utility is the sum of scheduled packets over the T stages. All packets take a single time step to serve. For convenience, we introduce a packet type 0 with value $v_0 = 0$ and probability $p_0 = 1$. It should be clear that this problem satisfies all assumptions above. In particular, the utility is bounded ($0 \leq f \leq T v_k$).

Why is the GAG small on this problem? We show that Δ_g is rarely high, inducing a small GAG. For s_t a state and $x, y \in \mathcal{X}(s_t)$, we say that x dominates y if $\mathcal{O}(s_t, x, \xi) \geq \mathcal{O}(s_t, y, \boldsymbol{\xi})$ almost surely given s_t. Studying $\Delta_g(s_t)$ only requires to focus on non-dominated decisions: there are at most two non-dominated decisions for a given state, which consists of scheduling

- the most valuable packet, of type i, received at time $t-1$ and not already scheduled; or
- the most valuable packet, of type j, received at time t.

Moreover, if $i \geq j$, then choosing j is dominated, since i is more valuable and will be lost if not chosen now. Also, if $i < j$ but the second most valuable packet received at t is of type $k \geq i$, then choosing i is dominated. If one of these two conditions holds, a decision dominates all the other ones, and thus $\Delta_g(s_t) = 0$.

Suppose now that s_t does not satisfy any of them. By the dominance property, scenarios can be partitioned into those where scheduling i (resp. j) is the unique offline, optimal decision and those on which there is a tie. Introduce the random variable y_t, taking values i, j or $\perp$ in these three respective cases. We then have

$$\begin{aligned}\Delta(s_t, i) &= \mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi}) - \mathcal{O}(s_t, i, \boldsymbol{\xi}) | s_t\right] \\ &= \mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi}) - \mathcal{O}(s_t, i, \boldsymbol{\xi}) | s_t, y_t = j\right] \mathbb{P}\left(y_t = j\right)\end{aligned}$$

and symmetrically

$$\begin{aligned}\Delta(s_t, j) &= \mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi}) - \mathcal{O}(s_t, j, \boldsymbol{\xi}) | s_t\right] \\ &= \mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi}) - \mathcal{O}(s_t, j, \boldsymbol{\xi}) | s_t, y_t = i\right] \mathbb{P}\left(y_t = i\right).\end{aligned}$$

Now, if i is scheduled and the optimal offline solution was to schedule j, then the loss cannot exceed $v_j - v_i$, since the rest of the optimal offline schedule is still feasible. Hence $\mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi}) - \mathcal{O}(s_t, i, \boldsymbol{\xi}) | s_t, y_t = j\right] \leq v_j - v_i$. Moreover, for the optimal offline schedule to choose j at time t, it is necessary to have a packet of value greater than v_i arriving at $t+1$ and thus $\mathbb{P}\left(y_t = j\right) \leq 1 - \prod_{k>i} q_k$ where $q_k = 1 - p_k$. Finally, we find

$$\Delta(s_t, i) \leq (v_j - v_i)\left(1 - \prod_{k>i} q_k\right).$$

The other case is harder to study, but a trivial upper bound is $\Delta(s_t, j) \leq v_i$. Now it remains to bound the expectation of $\Delta_g(s_t) = \min(\Delta(s_t, i), \Delta(s_t, j))$ by enumerating the possible values of i and j and weighting each case with its probability of occurrence. This weight is, in fact, bounded by the product of the probabilities of the 3 following events:

- a packet of type i arrived at time $t-1$;
- the most valuable packet arrived at t is of type j;
- no packet of type $i \leq k < j$ arrived at time t.

Here is a numerical example. Suppose there are 4 types of packets, with the following values and emission probabilities:

type	0	1	2	3	4
value	0	1	2	4	8
prob	1.	.60	.30	.20	.10

The upper bound on Δ_g depending on i and j, is

	1				
	2	.496			
(j)	3	1.00	.560		
	4	1.00	1.68	.400	
		1	2	3	4 (i)

We find that $\mathbb{E}[\Delta_g(s_t)] \leq .125$. On the other hand, a simple lower bound on the *EVC* is the expectation of the value of the most valuable packet arriving at each stage multiplied by the number of stages. In this case, that leads to $EVC \geq 2.25T$. As a result, the ratio of *GAG* over *EVC* on this problem is less than 5.55%. Because this analysis is not tight – especially the lower bound of the *EVC*–, the anticipatory algorithm is likely to have an even better expected global loss. This analysis also hints at why online anticipatory algorithms are so effective on packet scheduling and why they outperform competitive algorithms on these instances.

5.6 Empirical Validation of the Theorem

We will now proceed to an empirical validation of theorem 1. In order to do so, we need to estimate the Expected Global Loss and the Global Anticipatory Gap of *1s-AA-E* with varying number of scenarios. The theorem then states that the GAG should be lower, and that the EGL and the GAG should converge toward the same value.

5.6.1 Theory of Local Anticipatory Gap Estimation

An estimator of the GAG for a given decision-making algorithm can be derived from an estimator of the local anticipatory gap: from a sample of runs on different realizations of the uncertainty, an estimation of the GAG is the average, over these realizations, of the sum of estimation of the LAG for all decisions made in the corrsponding run. Therefore, we first study how to design a good estimator of Δ_g.

In all this section, it is assumed that there is a fied given state s_t, and all the expectations are conditionned on being in this state, i.e., $\boldsymbol{\xi} \in \mathcal{C}(s_t)$. $\mathcal{X}$ will be used instead of $\mathcal{X}(s_t)$ and $\mathcal{O}(x, \boldsymbol{\xi})$ instead of $\mathcal{O}(s_t, x, \boldsymbol{\xi})$.

In section 5.4.2, we gave the following alternative definition for the *LAG*:

$$\Delta_g = \mathbb{E}\left[\max_{x \in \mathcal{X}} \mathcal{O}(x, \boldsymbol{\xi})\right] - \max_{x \in \mathcal{X}} \mathbb{E}\left[\mathcal{O}(x, \boldsymbol{\xi})\right].$$

This definition suggests the following estimator of the *LAG*, that we denote $\widehat{\Delta}_g^{1,1}$: observing the sampled scenarios $\boldsymbol{\xi}^1, \ldots \boldsymbol{\xi}^m$, we define:

$$\widehat{\Delta}_g^{1,1} = \frac{1}{m}\left(\sum_{i=1}^{m} \max_{x \in \mathcal{X}} \mathcal{O}(x, \boldsymbol{\xi}^i) - \max_{x \in \mathcal{X})} \sum_{i=1}^{m} \mathcal{O}(x, \boldsymbol{\xi}^i)\right).$$

The strong law of large number , in conjonction with $\mathcal{X}$'s finiteness, implies that, when m converges to $+\infty$, $\widehat{\Delta}_g^{1,1}$ this converges almost surely to $\Delta_g(s_t)$. Moreover, it can be computed while running the 1step-AA with no additional computational cost, since all that is needed is the offline values for the samples scenarios. However, this estimator has the following drawback.

Lemma 3 $\widehat{\Delta}_g^{1,1}$ *has a nonpositive bias.*

Proof. We have that for all i, $\mathbb{E}\left[\max_{x\in\mathcal{X}} \mathcal{O}(x,\boldsymbol{\xi}^i)\right] = \mathbb{E}\left[\max_{x\in\mathcal{X}} \mathcal{O}(x,\boldsymbol{\xi})\right]$. Therefore, the bias is

$$\mathbb{E}\left[\widehat{\Delta}_g\right] - \Delta_g = \max_{x\in\mathcal{X}} \mathbb{E}\left[\mathcal{O}(x,\boldsymbol{\xi})\right] - \mathbb{E}\left[\max_{x\in\mathcal{X}} \frac{1}{m}\sum_{i=1}^{m} \mathcal{O}(x,\boldsymbol{\xi}^i)\right].$$

Let $x^\star$ be one argmax of $\mathbb{E}\left[\mathcal{O}(x,\boldsymbol{\xi})\right]$. Then $\max_{x\in\mathcal{X}} \sum_{i=1}^{m} \mathcal{O}(x,\boldsymbol{\xi}^i) \geq \sum_{i=1}^{m} \mathcal{O}(x^\star,\boldsymbol{\xi}^i)$, and it follows that the bias is non-negative. $\square$

This bias comes from the fact that the decision $\operatorname{argmax}_{x\in\mathcal{X}} \sum_{i=1}^{m} \mathcal{O}(x,\boldsymbol{\xi}^i)$ is evaluated on the same set of scenarios that is used to select it. Therefore one could think of correcting it by first selecting a decisions with some set of scenarios, and then evaluating it on another set, which can or not intersect the first one. This leads to a whole family of estimators described below.

Definition 7 *Let $0 < \alpha \leq 1$ and $0 < \beta \leq 1$. The estimator $\widehat{\Delta}_g^{\alpha,\beta}$ is defined as:*

$$\widehat{\Delta}_g^{\alpha,\beta} = \frac{1}{\lceil \beta m \rceil} \sum_{i=m-\lceil \beta m\rceil+1}^{m} \max_{x\in\mathcal{X}} \mathcal{O}(x,\boldsymbol{\xi}^i) - \frac{1}{\lceil \beta m\rceil \# M_m^\alpha} \sum_{i=m-\lceil \beta m\rceil+1}^{m} \sum_{x\in M_m^\alpha} \mathcal{O}(x,\boldsymbol{\xi}^i)$$

with

$$M_m^\alpha = \operatorname*{argmax}_{x\in\mathcal{X}} \sum_{i=1}^{\lfloor \alpha m\rfloor} \mathcal{O}(x,\boldsymbol{\xi}^i)$$

The estimator $\widehat{\Delta}_g^{\alpha,\beta}$ uses the first $\lfloor \alpha m \rfloor$ scenarios of the sample to select a decision, and evaluates the exected local loss of that decision on the last $\lceil \beta m\rceil$ scenarios of the sample. Note that when $\alpha = \beta = 1$, this definition is consistent with the one stated earlier.

Theorem 2 *For all $0 < \alpha, \beta \leq 1$, $\widehat{\Delta}_g^{\alpha,\beta}$ is a strongly consistent estimator, that is,*

$$\mathbb{P}\left(\lim_{m\to+\infty} \widehat{\Delta}_g^{\alpha,\beta} \ = \ \Delta_g(s_t)\right) = 1.$$

Proof. First, it is convenient to name the two terms that appear in $\widehat{\Delta}_g^{\alpha,\beta}$. We call $\boldsymbol{L}_m = \frac{1}{\lceil \beta m\rceil} \sum_{i=m-\lceil \beta m\rceil+1}^{m} \max_{x\in\mathcal{X}} \mathcal{O}(x,\boldsymbol{\xi}^i)$ and $\boldsymbol{R}_m = \frac{1}{\lceil \beta m\rceil \# M_m^\alpha} \sum_{i=m-\lceil \beta m\rceil+1}^{m} \sum_{x\in M_m^\alpha} \mathcal{O}(x,\boldsymbol{\xi}^i)$. That is, $\widehat{\Delta}_g^{\alpha,\beta}$ can be written as $\boldsymbol{L}_m - \boldsymbol{R}_m$.

The term $\boldsymbol{L}_m$ is handled using the Strong Law of Large Numbers (SLLN). Indeed, it can be decomposed as $\boldsymbol{L}_m = \frac{1}{\lceil \beta m\rceil}\left(m\boldsymbol{A}_m - (m - \lceil \beta m\rceil)\boldsymbol{B}_m\right)$, where

$$\boldsymbol{A}_m = \frac{1}{m}\sum_{i=1}^{m} \max_{x\in\mathcal{X}} \mathcal{O}(x,\boldsymbol{\xi}^i) \qquad \text{and} \qquad \boldsymbol{B}_m = \frac{1}{m - \lceil \beta m\rceil} \sum_{i=1}^{m-\lceil \beta m\rceil} \max_{x\in\mathcal{X}} \mathcal{O}(x,\boldsymbol{\xi}^i).$$

Now, both $\boldsymbol{A}_m$ and $\boldsymbol{B}_m$ converge almost surely to $\mathbb{E}\left[\max_{x\in\mathcal{X}} \mathcal{O}(x,\boldsymbol{\xi})\right]$, and thus so does $\boldsymbol{L}_m$.

The second term $\boldsymbol{R}_m$ is harder to handle. We prove that it converges a.s. to $\max_{x\in\mathcal{X}} \mathbb{E}\left[\mathcal{O}(x,\boldsymbol{\xi})\right]$. Let M_∞ be $\operatorname{argmax}_{x\in\mathcal{X}} \mathbb{E}\left[\mathcal{O}(x,\boldsymbol{\xi})\right]$, and $x^\star$ be an element of M_∞.

We proceed in two steps. First we prove that almost surely $M_m^\alpha \subseteq M_\infty$ for M large enough. Indeed, let $x^\star \in M_\infty$, and $x \notin M_\infty$. There exits $\varepsilon > 0$ such that $\mathbb{E}\left[\mathcal{O}(x,\boldsymbol{\xi})\right] < \mathbb{E}\left[\mathcal{O}(x^\star,\boldsymbol{\xi})\right] - \varepsilon$. Now, again by the SLLN, there is almost surely a $N \in \mathbb{N}$ such that for all $m \geq N$,

$$\sum_{i=1}^{\lfloor \alpha m \rfloor} \mathcal{O}(x^\star, \boldsymbol{\xi}^i) > \mathbb{E}\left[\mathcal{O}(x^\star, \boldsymbol{\xi})\right] - \varepsilon,$$
$$\sum_{i=1}^{\lfloor \alpha m \rfloor} \mathcal{O}(x, \boldsymbol{\xi}^i) < \mathbb{E}\left[\mathcal{O}(x^\star, \boldsymbol{\xi})\right] - \varepsilon.$$

Therefore for $m \geq N$, $x \notin M_m^\alpha$.

The second step is to show that in that case, the evaluation of the decisions in M_m^α on the $\lceil \beta m \rceil$ scenarios almost surely converges with $\max_{x\in\mathcal{X}} \mathbb{E}\left[\mathcal{O}(x, \boldsymbol{\xi})\right]$. Let $M \subseteq M_\infty$. Then $\frac{1}{\lceil \beta m \rceil \# M} \sum_{i=m-\lceil \beta m \rceil+1}^{m} \sum_{x\in M} \mathcal{O}(x, \boldsymbol{\xi}^i) = \frac{1}{\# M} \sum_{x\in M} \left(\frac{1}{\lceil \beta m \rceil} \sum_{i=m-\lceil \beta m \rceil+1}^{m} \mathcal{O}(x, \boldsymbol{\xi}^i) \right)$. Each term of the outer sum converges a.s. to $\mathbb{E}\left[\mathcal{O}(x^\star, \boldsymbol{\xi})\right]$, so by finiteness of M, the whole quantity converges a.s. to the same limit. For any $\varepsilon > 0$, there is therefore an integer $N_{\varepsilon,M}$ such that for all $m > N_{\varepsilon,M}$, $\left| \frac{1}{\lceil \beta m \rceil \# M} \sum_{i=m-\lceil \beta m \rceil+1}^{m} \sum_{x\in M} \mathcal{O}(x, \boldsymbol{\xi}^i) - \mathbb{E}\left[\mathcal{O}(x^\star, \boldsymbol{\xi})\right] \right| < \varepsilon$. Now, because M_∞ is finite, its power set is also finite, and so we can define N_ε as $\max_{M \subseteq M_\infty} N_{\varepsilon,M}$. It follows that for any $m > N_\varepsilon$, $\left|\boldsymbol{R}_m - \mathbb{E}\left[\mathcal{O}(x^\star, \boldsymbol{\xi})\right]\right| < \varepsilon$, which ends the handling of $\boldsymbol{R}_m$.

Finally, we have proved that $\boldsymbol{L}_m \xrightarrow[m\to\infty]{\text{a.s.}} \mathbb{E}\left[\max_{x\in\mathcal{X}} \mathcal{O}(x, \boldsymbol{\xi})\right]$, and that $\boldsymbol{R}_m \xrightarrow[m\to\infty]{\text{a.s.}} \max_{x\in\mathcal{X}} \mathbb{E}\left[\mathcal{O}(x, \boldsymbol{\xi})\right]$. Hence the $\widehat{\Delta}_g^{\alpha,\beta} = \boldsymbol{L}_m - \boldsymbol{R}_m$ converges a.s. to Δ_g. $\square$

There are interesting limit cases concerning the values of α and β. We discussed above the case $\alpha = \beta = 1$, for which we proved by the estimator was nonpositively bias. The second one follows.

Lemma 4 *If $\alpha + \beta \leq 1$, then $\widehat{\Delta}_g^{\alpha,\beta}$ has a nonnegative bias.*

Proof. The idea here is that with $\alpha + \beta \leq 1$, then the intervals $[1..\lfloor \alpha m \rfloor]$ and $[m - \lceil \beta m \rceil + 1..m]$ do not overlap. As a consequence, the decisions M_m^α are evaluated on scenarios independent from those which has been use to define M_m^α. Indeed, $\mathbb{E}\left[\widehat{\Delta}_g^{\alpha,\beta}\right] - \Delta_g = (\boldsymbol{L}_m - \boldsymbol{R}_m) - \left(\mathbb{E}\left[\max_{x\in\mathcal{X}} \mathcal{O}(x, \boldsymbol{\xi})\right] - \max_{x\in\mathcal{X}} \mathbb{E}\left[\mathcal{O}(x, \boldsymbol{\xi})\right]\right)$. Now, we have $\mathbb{E}\left[\boldsymbol{L}_m\right] = \mathbb{E}\left[\frac{1}{\lceil \beta m \rceil} \sum_{i=m-\lceil \beta m \rceil+1}^{m} \max_{x\in\mathcal{X}} \mathcal{O}(x, \boldsymbol{\xi}^i)\right] = \mathbb{E}\left[\max_{x\in\mathcal{X}} \mathcal{O}(x, \boldsymbol{\xi})\right]$: this is by linearity of expectation, and because the ξ^i have the same distribution than ξ.

Now we look at $\mathbb{E}\left[\boldsymbol{R}_m\right]$.

$$\begin{aligned}
\mathbb{E}\left[\boldsymbol{R}_m\right] &= \mathbb{E}\left[\frac{1}{\lceil \beta m \rceil \# M_m^\alpha} \sum_{i=m-\lceil \beta m \rceil+1}^{m} \sum_{x\in M_m^\alpha} \mathcal{O}(x, \boldsymbol{\xi}^i)\right] \\
&= \sum_{M \subseteq \mathcal{X}} \mathbb{E}\left[\frac{1}{\lceil \beta m \rceil \# M} \sum_{i=m-\lceil \beta m \rceil+1}^{m} \sum_{x\in M} \mathcal{O}(x, \boldsymbol{\xi}^i) \middle| M = M_m^\alpha\right] \mathbb{P}\left(M = M_m^\alpha\right)
\end{aligned}$$

Because the sets of scenarios do not overlap, for $i \geq m - \lceil \beta m \rceil + 1$, ξ is independent of M_m^α. So we can remove the conditionning:

$$
\begin{aligned}
&= \sum_{M \subseteq \mathcal{X}} \mathbb{E}\left[\frac{1}{\lceil \beta m \rceil \# M} \sum_{i=m-\lceil \beta m \rceil+1}^{m} \sum_{x \in M} \mathcal{O}(x, \boldsymbol{\xi}^i)\right] \mathbb{P}\left(M = M_m^\alpha\right) \\
&= \sum_{M \subseteq \mathcal{X}} \mathbb{E}\left[\frac{1}{\# M} \sum_{x \in M} \mathcal{O}(x, \boldsymbol{\xi})\right] \mathbb{P}\left(M = M_m^\alpha\right). \\
&\leq \max_{M \subseteq \mathcal{X}} \mathbb{E}\left[\frac{1}{\# M} \sum_{x \in M} \mathcal{O}(x, \boldsymbol{\xi})\right] \\
&\leq \max_{x \in \mathcal{X}} \mathbb{E}\left[\mathcal{O}(x, \boldsymbol{\xi})\right]
\end{aligned}
$$

Hence $\mathbb{E}\left[\widehat{\Delta}_g^{\alpha,\beta}\right] - \Delta_g = \max_{x \in \mathcal{X}} \mathbb{E}\left[\mathcal{O}(x, \boldsymbol{\xi})\right] - \mathbb{E}\left[\boldsymbol{R}_m\right] \geq 0.$ $\square$

Since one limit case give a nonnegative bias, and the other a nonpositive one, it is possible that there exists a pair (α, β) which provides almost-unbiased estimation. However, it is not clear how to find such a couple. We will experiment different pair (α, β).

5.6.2 Empirical Performance of the Estimators

We report some experimental data obtained thanks to the estimators designed above. All the following results are based on the runs on different realizations of one instance of the multiknapsack problem. This instance is called `bbcr5nc`, and is defined by:

- The deadline is 1000;
- There are 5 bins, all of capacity 100;
- There are 5 items types. For each type, the arrival of items is a Poisson process, independent of those of the other types, of inter-arrival mean time 173. The values and weights of the items are:

type	1	2	3	4	5
weight	17	20	25	30	33
value	13	26	21	26	39

- Finally, there is no cancellation (more accurately, to be compatible with the file format used in previous work, cancellations are set to be extremely unlikely, with a probablity less than $1e-6$ for each item).

This is a rather easy instance, on which Regret performs very well, and the offline problems are easy to solve. This is intentionnal: here the goal is not to compare the performance of different algorithms. By choosing an easy instance, we avoid polluting the results by effects like having Cplex stopping because of a time-out, in which case we have no garantee on the quality of the offline solution.

The experimental setting is the following: 400 realizations of this instances were generated. Each realization was solved once using each of the four following policies:

1. BEST FIT is the greedy algorithm that always accept an item that fits somewhere, and in that case put it into the bin with smallest residual capacity in which it fits;

2. *1s-AA-E* with 100 scenarios per decisions;

3. DO NOTHING is the policy that reject all items;

4. RANDOM is the policy that choose where to assign items uniformly at random among the bins in which there is enough room, including a dummy bin corresponding to the rejection. That is, if the incoming item fits in 3 out of the 5 bins, it will be rejected with probability $1/4$, and accepted in one of these 3 bins with equal probability.

For all these runs, at each decision to be made, 100 scenarios were generated and the score matrices, compted with *1s-AA-E*, were outputed. When running *1s-AA*, the same score matrix used to make the decision was outputed.

First, let's study the quality of the estimators. Because all the estimators $\widehat{\Delta}_g^{\alpha,\beta}$ are strongly consistent, theire bias converges to zero as the number of scenario increases. As a consequence, it is possible to study their bias by observing the convergence of the estimated values as m increases.

Each estimator of the local AG induces an estimator of the global AG, obtained by computing the mean, over many runs, of the sums of the empirical local AG at each decisions of these runs. We computed these estimations, based on the score matrices that we output, restricting our estimators to use only the first 10, 20,... 90, or all the 100 lines of each score matrix. For each estimator and number of scenarios per decisions, we get one estimation of the GAG of the four policies. This is was is depicted on figure 5.3. The theory tells us that for each policy, all these estimations should converge with the same values, and the experimental results confirm this observation. Our lemmas 3 and 4 are also confirmed, since we oberve that all points on the graphs corresponding to $\widehat{\Delta}_g^{1,1}$ are below the point of convergence, whereas for $\widehat{\Delta}_g^{\frac{1}{2},\frac{1}{2}}$ they are above.

Now, the flatness of the graphs indiquate that the bias is small. Note that not because there are extreme values of α and β for which the bias has a proven opposite sign means there is a pair for which it is zero: it could be that some estimators are positively biased for some policies but negatively for others, so we cannot apply some kind of intermediate value theorem.

However, on the four policies, the graph of $\widehat{\Delta}_g^{\frac{1}{2},1}$ and of $\widehat{\Delta}_g^{.7,.7}$ are quite flat. So on these 4 policies, these two estimator have the smallest bias of the 7 we tried, and we can hope this result can generalize.

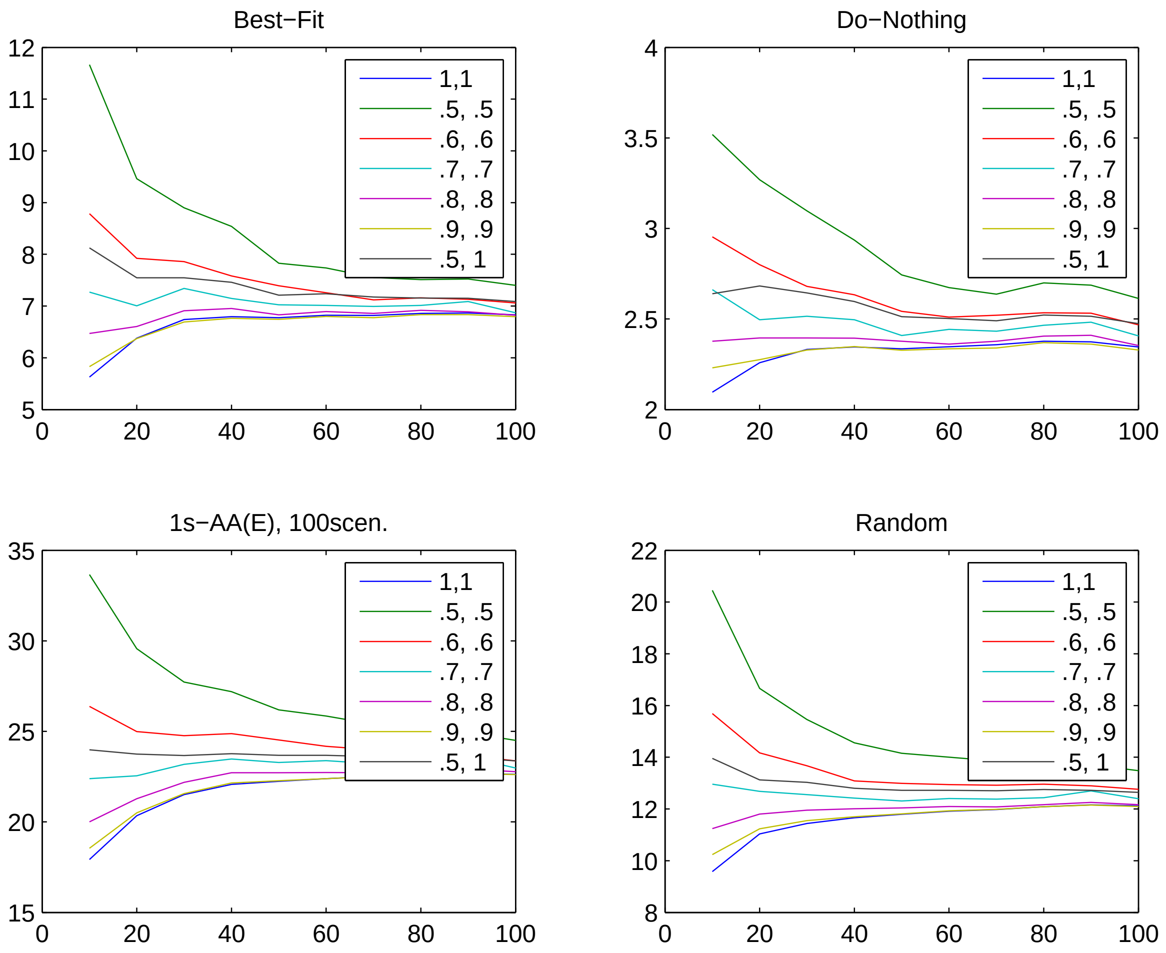


Figure 5.3: Comparaison of 7 different estimators on to measure the GAG of 4 different policies.

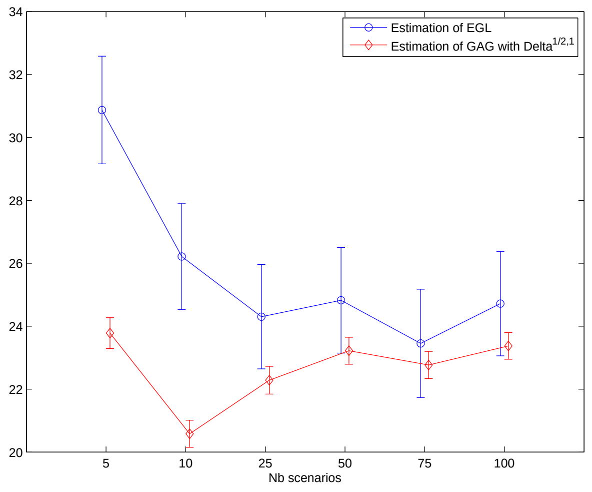


Figure 5.4: Estimations of the EGL and of the GAG on instance bbcr5nc, with 68% confidence intervals

5.6.3 Validating the Theorem using $\widehat{\Delta}_g^{1/2,1}$

The previous section concluded that $\widehat{\Delta}_g^{1/2,1}$ is a good estimator for the local AG. Now we will use the associated estimator of the GAG to empirically verify theorem 1. The experimental protocol is as follow. We have 400 realizations of bbcr5nc. For each of them, we run *1s-AA-E* six times: once with 5 scenarios per decision, one with 10, 25, 50, 75 and 100 scenarios per decisions. In addition, we run a clairvoyant solver to compute the offline score of these 400 realizations. This allows us to estimate, for each number of scenarios per decision, the EGL: the estimation is simply the empirical mean of the global loss.

Concerning the GAG, we derive estimations of the GAG by adding the estimations of the local anticipatory gap, measure thanks to $\widehat{\Delta}_g^{1/2,1}$, over the runs. Figure 5.4 show 68% confidence intervals for the EGL and the GAG. You can see of this figure that with enough scenarios, from 50 scenarios per decision, the two confidence intervals intersects (the one for the GAG is even included in the other with 50 and 100 scenarios). This is in accordance with theorem 1. This experiments show that a very reasonable number of scenarios, here 50, is enough to observe the convergence promised by the theory. On the practical side, this also indicate that if *1s-AA-E* is good on this problem because *s-GAG* is small, then there is no need to sample further: the quality of decisions will not increase.

5.7 Approximating the Offline Problem

Theorem 1 explains why the anticipatory algorithm provides good results when the GAG is small. However, practical implementations use a variant of Algorithm 1 in which $\mathcal{O}$ is replaced by a fast approximation $\widetilde{\mathcal{O}}$. This is the case for the three applications mentioned earlier which use an approximating technique called *Regret* [5, 26]. The *Regret* algorithm can be seen as a discrete version of sensitivity analysis: instead of computing $\mathcal{O}(s_t, x, \xi^i)$ for each for each $x \in \mathcal{X}(s_t)$, the idea is to compute $x^* = \text{argmax}_x(\mathcal{O}(s_t, x, \xi^i))$ first and then to compute a vector of approximations $\left(\widetilde{\mathcal{O}}(s_t, x, \xi^i)\right)_{x \in \mathcal{X}(s_t)}$ using x^*. Each entry in this vector is derived by approximating the loss of selecting a specific x in $\mathcal{X}(s_t)$ instead of the optimal decision x^*. As a result, the *Regret* algorithm ensures (i) $\widetilde{\mathcal{O}} \leq \mathcal{O}$ and (ii) $\max_x(\widetilde{\mathcal{O}}(s_t, x, \xi^i)) = \max_x(\mathcal{O}(s_t, x, \xi^i))$. See [9] for a discussion on the complexity of computing this approximated vector.

It is not easy to provide tight theoretical bounds on the expected global loss for the *Regret* approximation. We thus measured the empirical distribution of $\mathcal{O}-\widetilde{\mathcal{O}}$ on online stochastic reservation systems from [26]. The difference $\mathcal{O}-\widetilde{\mathcal{O}}$ is zero in 80% of the cases and its mean is very small (around .2 while the typical values of $\mathcal{O}$ are in the range [400,500]), although it can occasionally be large. This intuitively justifies the quality of *Regret*, whose expected global loss is not significatively different from the anticipatory algorithm for the same sample size.

Finally, recall that, on online reservation systems, the consensus rate is very high on average. Let $x^\star = \text{argmax}_x \sum_{i=1}^{m} \mathcal{O}(s_t, x, \boldsymbol{\xi}^i)$ and let the consensus rate be α. By properties (i) and (ii) of *Regret*, the approximated "score" $\sum_{i=1}^{m} \widetilde{\mathcal{O}}(s_t, x^\star, \boldsymbol{\xi}^i)$ of decision $x^\star$ may only exhibit errors in $(1-\alpha)m$ scenarios and hence will be very close to $\sum_{i=1}^{\alpha m} \mathcal{O}(s_t, x^\star, \boldsymbol{\xi}^i)$. Moreover, other decisions have an approximated score where (almost all) the terms of the sum has a negative error. Therefore the approximated decision is biased toward consensual decisions and a high consensus rate tends to hide the approximation errors $\mathcal{O} - \widetilde{\mathcal{O}}$ of *Regret*.

In summary, a high consensus rate not only makes the AG small but also allows *Regret* to produce decisions close in quality to the exact anticipatory algorithm. This does not mean that a brutal *Regret* approximation, e.g., assigning zero to each non-optimal decision, would be as effective [5].

5.8 *GAG* Versus *EVPI*

This section studies the relationships between the anticipatory gap and the expected value of perfect information (*EVPI*). Since these concepts are seemingly close, it is useful to explain how they differ and why we chose to introduce the notion of anticipatory gap.

Consider the following two maps assigning values to states: the *offline value* and the *online value* of state s_t, respectively denoted by $\phi(s_t)$ and $\pi(s_t)$ and defined by

$$\phi(s_t) = \max\left\{\mathbb{E}\left[f(x, \boldsymbol{\xi})|s_t\right] \;\mid\; x \text{ dec. proc.}\right\}$$
$$\pi(s_t) = \max\left\{\mathbb{E}\left[f(x, \boldsymbol{\xi})|s_t\right] \;\mid\; x \text{ non-anticip. dec. proc.}\right\}.$$

Note that $\phi(s_t) \geq \pi(s_t)$ for all state s_t. The difference

$$\eta(s_t) = \phi(s_t) - \pi(s_t).$$

is the (local) *expected value of perfect information* (*EVPI*). The expected value of perfect information of the problem is $\eta(s_1)$, that is, the advantage, in the expected sense, of a clairvoyant over a non-clairvoyant (both with infinite computational resources). The next lemma relates the offline problem and ϕ and shows that the operators max and $\mathbb{E}$ commute for clairvoyant decision processes.

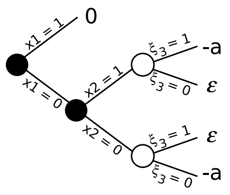


Figure 5.5: low *EVPI* but high Global Anticipatory Gap

Lemma 5 *For any state s_t, $\phi(s_t) = \mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi})|s_t\right]$.*

Proof. Let $x^\star$ be a decision process maximizing $\mathbb{E}\left[f(x, \boldsymbol{\xi})|s_t\right]$. Then for all ξ, $\mathcal{O}(s_t, \xi) \geq f(x^\star, \xi)$, and, as $\mathbb{E}$ is non-decreasing, $\mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi})|s_t\right] \geq \mathbb{E}\left[f(x^\star, \boldsymbol{\xi})|s_t)\right] = \phi(s_t)$. Inversely, let x^ξ be a decision process maximizing $f(x, \xi)$ with $x^\xi_{1..t-1} = x_{1..t-1}$. Define $x^\star$ by aggregation: $x^\star$ does the same thing as x^ξ on the scenario ξ. Then $\phi(s_t) \geq \mathbb{E}\left[f(x^\star, \xi)|s_t\right] = \mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi})|s_t\right]$. □

In two-stage stochastic programming, a low *EVPI* makes the problem much easier because an optimal decision is also a good one for each specific scenario [12, ch. 4]. However, this is no longer true in the multistage case. Consider the three-stage problem depicted in Figure 5.5. Black dots represent decisions variables x_1 and x_2. Stochastic variables ξ_1 and ξ_2 have no influence and are not represented. The white dot represents ξ_3 which take values 0 and 1 with equal probability. Leaves are tagged with their utilities and a is large positive number. The value of the *EVPI* and the anticipatory gap Δ_g for each state are the following:

state	root state	$x_1 = 0$
Δ_g	0	$1/2(\varepsilon + a)$
η	ε	$1/2(\varepsilon + a)$

On this problem, the *EVPI* is ε: an optimal solution has a score of ε, whatever the scenario. The expected value of the optimal policy is zero. However, the online anticipatory algorithm always chooses $x_1 = 0$ and thus has an expected utility of $1/2(\varepsilon - a)$. Therefore anticipatory algorithms may behave poorly even with a low *EVPI.* Moreover, in this case, the inequality of Theorem **??** is tight when m converges to $+\infty$, since the *GAG* equals $1/2(\varepsilon + a)$.

The phenomenon comes from the fact that the *EVPI* of the problem is low although the *EVPI* of the node $(x_1 = 0)$ is $\varepsilon - 1/2(\varepsilon - a) = 1/2(\varepsilon + a)$ and thus much larger. This does not contradict the super-martingale property of [20] because Dempster considers optimal decision processes, which is not the case of anticipatory algorithms. As a result, the expected global loss of the anticipatory algorithm cannot be bounded by the root *EVPI.* The example may suggest that the maximum of the *EVPI*s at each node of the tree gives an upper bound of the *EGL*, but this is not true either. Figure 5.6 presents a stochastic program, where 'Sub' are clones of the problem in Figure 5.5, with variables indices shifted. On this problem, the optimal solutions to the scenarios have an expected expected utility of ε, and those of the anticipatory algorithm (with $m = \infty$) have expected utility $1/4(-3a + \varepsilon)$; the *EGL* thus equals $3/4(a + \varepsilon)$. By Theorem **??**, the *GAG* is not smaller than the *EGL* ($m = \infty$: no sampling error). As a result, the *GAG* is greater than the maximum of the *EVPI*

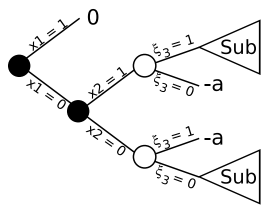


Figure 5.6: *GAG* higher than max of the *EVPI*s

over all nodes, which is equal to $1/2(\varepsilon + a)$.

Finally, the following theorem gives one more reason why the concept of anticipatory gap is of interest.

Theorem 3 *For any state s_t, we have $\eta(s_t) \geq \Delta_g(s_t)$ and there exist cases in which the inequality is strict.*

Proof. $\eta(s_t) = \phi(s_t) - \pi(s_t)$. Recall that $\pi(s_t)$ is the optimal expected utility of a non-anticipative decision process given s_t. Because of non-anticipativity and because $\mathcal{X}(s_t)$ is finite, there exists an optimal decision $x^\star \in \mathcal{X}(s_t)$ such that $\pi(s_t) = \mathbb{E}\left[\pi(s_{t+1})|s_t, x_t = x^\star\right]$. Now

$$\begin{aligned}\max_{x\in\mathcal{X}(s_t)} \mathbb{E}\left[\mathcal{O}(s_t, x, \boldsymbol{\xi})|s_t\right] &\geq \mathbb{E}\left[\mathcal{O}(s_t, x^\star, \boldsymbol{\xi})|s_t\right] \\ &\geq \mathbb{E}\left[\pi(s_{t+1})|s_t, x_t = x^\star\right].\end{aligned}$$

and thus, using lemma 5,

$$\begin{aligned}\eta(s_t) &\geq \mathbb{E}\left[\mathcal{O}(s_t, \boldsymbol{\xi})|s_t\right] - \max_{x\in\mathcal{X}(s_t)} \mathbb{E}\left[\mathcal{O}(s_t, x, \boldsymbol{\xi})|s_t\right] \\ &\geq \Delta_g(s_t).\end{aligned}$$

This proves the inequality. The second part of the theorem is proven by the example of Figure 5.5, on which the root node s_1 satisfies $\eta(s_1) = \varepsilon > 0 = \Delta_g(s_1)$. □

5.9 Conclusion and Research Perspectives

One-Step Anticipatory algorithms have been shown experimentally to be successful in tackling a variety of large multistage stochastic integer programs which were outside the scope of a priori methods such as (PO)MDPs and multistage stochastic programming. This chapter studied the performance of anticipatory algorithms in terms of their sampling error and the anticipatory gap of the problems. It showed that, whenever the *s-GAG* is small, *1s-AA-E* is scalable and provide high-quality solutions in the expected sense with a logarithmic number of samples in the problem size. The chapter also studied how to bound the anticipatory gap both theoretically and experimentally, showing that a simple packet scheduling problem admits a small anticipatory gap and providing experimental evidence on several large multistage stochastic programs. Finally, the chapter indicated that the anticipatory gap is an important concept and studied its relationships with the expected value of perfect information.

The understanding how what makes the GAG small or large is, however, still very primitive, and more work would be needed, for example to improve the techniques used in section 5.5. On the other hand, for many other problems *1s-AA-E* does *not* produce as high-quality decisions. The rest of this disseration is concerned with adressing that case.

Chapter 6

Amsaa: the Anytime Mulstistep Anticipatory Algorithm

This chapter presents the main contribution of the paper: *Amsaa*, the Anytime Multi-Step Anticipatory Algorithm for combinatorial optimization problems with exogenous uncertainty. We start with a high-level overview of *Amsaa* before presenting each step in detail.

6.1 Overview of *Amsaa*

The core of *Amsaa* is the Multi-Step Anticipatory Algorithm (`Msaa`) whose pseudo code follows.

```
Function Msaa(X-MDP A)
1 Approximate the X-MDP A by exterior sampling: replace the
  distribution of $\boldsymbol{\xi}$ by the empirical distribution on a sample of $\boldsymbol{\xi}$
2 Convert the resulting X-MDP to a standard MDP;
3 Solves the resulting MDP with a search algorithm for MDPs, using the
  offline upper bound $h_{\mathbb{E},\max}(s) = \mathbb{E}\left[\mathcal{O}(s, \boldsymbol{\xi}') \mid \boldsymbol{\xi}' \in \mathcal{C}(s)\right]$;
4 return the greedy decision at the root node of the MDP.
```

The first step of `Msaa` approximates the X-MDP by exterior sampling to make it more tractable. It then converts the resulting X-MDP into an MDP, making it possible to apply standard search algorithms for MDPs. The third step applies such an algorithm using an upper bound that exploits the value of the offline problems associated with the approximated X-MDPs. Finally, the fourth step returns the decision selected by the optimal policy at the root node of the MDP. Note that `Msaa` exploits the exogenous nature of the uncertainty in steps 1 and 3, i.e., to approximate the problem and to guide the search towards the optimal policy.

Amsaa is an anytime algorithm based on `Msaa`. It iteratively applies algorithm `Msaa` on increasingly finer approximations of the original X-MDP until some termination condition is met.

Operationally, such a condition is likely to be a time constraint (e.g., "make a decision within one minute") but it could also be a stopping criterion based on some accuracy measure such as the contamination method [22]. This iterative refinement is made efficient by the incremental nature of *Amsaa*: Calls to `Msaa` reuse earlier computations, so that resolving the MDP is fast after a small change in the approximation. We now study each component of *Amsaa*.

6.2 Approximating the X-MDP

The first step of *Amsaa* is to approximate the original X-MDP by replacing the distribution of the scenarios by one with a finite and reasonably small support.

6.2.1 The Sample Average Approximation Method

A simple way of approximating a distribution is by sampling. For stochastic programs, this idea is called the Sample Average Approximation (SAA) method [43] and it extends naturally to X-MDPs. Suppose we want a distribution whose support has cardinality at most n: Sample $\boldsymbol{\xi}$ n times, independently or not, to obtain $\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^n$ and define $\hat{\boldsymbol{\mu}}_n$ as the empirical distribution of $\boldsymbol{\xi}$ induced by this sample, i.e., the distribution assigning probability $1/n$ to each of the sampled scenarios. In the following, we use A and $\hat{\boldsymbol{A}}_n$ to denote the X-MDPs A_μ and $A_{\hat{\boldsymbol{\mu}}_n}$.

6.2.2 The Benefits of Exterior Sampling for X-MDPs

Sampling can be used either to approximate a problem that is then solved exactly (The SAA method) or to compute an approximate solution of the original problem. *Amsaa* approximates the original problem and solves the resulting X-MDP exactly (exterior sampling). In [30], Kearns, Mansour and Ng (KMN) took the other road: they proposed an algorithm to solve approximately an MDP by sampling a number of outcomes at each visited state (interior sampling). Their algorithm was presented for discounted rewards but generalizes to the objective and assumptions of this paper. However, interior sampling does not exploit a fundamental advantage of problems with exogenous uncertainty: *positive correlations.*

Indeed, in a state s, the optimal decision maximizes $Q^\star(s,x)$, where $Q^\star$ is the Q-value function associated to the optimal value function $v^\star$. However, estimating this value precisely is not important. What really matters is to estimate the sign of the difference $Q^\star(s,x_1) - Q^\star(s,x_2)$ for each pair of decisions $x_1, x_2 \in \mathcal{X}(s)$. Now, consider two functions g and h mapping scenarios to reals. Two examples of such functions are

1. the offline values for two decisions: $g(\xi) = \mathcal{O}(\tau(s,x_1,\xi),\xi)$ and $h(\xi) = \mathcal{O}(\tau(s,x_2,\xi),\xi)$;
2. the optimal policy value obtained from a state s after making a first decision. That is, $g(\xi) = v(\tau(s_0,x_1,\xi))$ and $h(\xi) = v(\tau(s_0,x_2,\xi))$ for two decisions $x_1, x_2 \in \mathcal{X}(s_0)$.

If ξ^1 and ξ^2 are iid scenarios, then

$$\begin{aligned}
\operatorname{var}\left(g(\xi^1) - h(\xi^2)\right) &= \operatorname{var}\left(g(\xi^1)\right) + \operatorname{var}\left(h(\xi^2)\right), \\
\operatorname{var}\left(g(\xi^1) - h(\xi^1)\right) &= \operatorname{var}\left(g(\xi^1)\right) + \operatorname{var}\left(h(\xi^1)\right) - 2\operatorname{cov}\left(g(\xi^1), h(\xi^1)\right),
\end{aligned}$$

and therefore

$$\operatorname{var}\left(g(\xi^1) - h(\xi^1)\right) = \left(1 - \operatorname{acorr}\left(g(\xi^1), h(\xi^1)\right)\right) \cdot \operatorname{var}\left(g(\xi^1) - h(\xi^1)\right),$$

where $\operatorname{acorr}(X, Y) = \frac{\operatorname{cov}(X)\operatorname{cov}(Y)}{1/2(\operatorname{var}(X)+\operatorname{var}(Y))}$ is a quantity we call *arithmetic correlation*. Note that $\operatorname{acorr}(X, Y)$ is close to $\operatorname{corr}(X, Y)$ when $\operatorname{var}(X)$ and $\operatorname{var}(Y)$ are close. Now consider an infinite iid sample $\xi^1, \xi^{1'}, \xi^2, \xi^{2'}, \ldots$, and a large integer n. By the central limit theorem, the distributions of

$$\frac{1}{n}\sum_{i=1}^{n} g(\xi^i) - h(\xi^i) \qquad \text{and of} \qquad \frac{1}{n\gamma}\sum_{i=1}^{n\gamma} g(\xi^i) - h(\xi^{i'})$$

are almost the same when $1/\gamma = 1 - \operatorname{acorr}\left(g(\xi^1), h(\xi^1)\right)$. Therefore, for some specified accuracy, the number of required scenarios to estimate the expected difference between $g(\xi)$ and $h(\xi)$ is reduced by this factor γ when the same scenarios (exterior sampling) are used instead of independent scenarios (interior sampling). This argument is not new and can be found in, say, [43, Ch. 6]. However, no empirical evidence of high correlations were given, which we now report. Consider an SAA problem approximating the standard instance of the S-RCPSP application with 200 scenarios generated by iid sampling, and consider the offline and optimal policy values in the initial state for the 6 possible initial decisions. Associating a column with each decision, the values for the first 8 scenarios are:

$$OfflineValue = 1e4 \times \begin{pmatrix}
0 & 2.170 & 2.170 & 2.125 & 2.130 & 2.170 \\
0 & -0.050 & -0.030 & -0.065 & -0.060 & -0.060 \\
0 & -0.030 & -0.025 & -0.060 & -0.055 & -0.060 \\
0 & 1.440 & 1.470 & 1.405 & 1.470 & 1.410 \\
0 & 1.160 & 1.160 & 1.135 & 1.130 & 1.185 \\
0 & -0.025 & -0.050 & -0.065 & -0.055 & -0.060 \\
0 & 0.829 & 0.804 & 0.829 & 0.789 & 0.769 \\
0 & 2.015 & 2.015 & 2.005 & 2.065 & 2.065
\end{pmatrix}$$

$$OptPolicyValue = 1e4 \times \begin{pmatrix}
0 & 2.110 & 2.038 & 1.910 & 1.893 & 2.170 \\
0 & -0.265 & -0.275 & -0.275 & -0.275 & -0.225 \\
0 & -0.205 & -0.230 & -0.230 & -0.170 & -0.170 \\
0 & 1.375 & 1.405 & 1.279 & 1.345 & 1.365 \\
0 & 1.045 & 1.070 & 1.015 & 1.105 & 1.160 \\
0 & -0.140 & -0.195 & -0.255 & -0.275 & -0.180 \\
0 & 0.829 & 0.804 & 0.789 & 0.230 & 0.255 \\
0 & 1.811 & 1.955 & 1.955 & 2.015 & 2.015
\end{pmatrix}$$

The first columns correspond to the decision of not scheduling anything, which is why it is always zero. Other columns correspond to scheduling the first task of each project respectively. The correlation is evident. The arithmetic correlation matrices, computed over the 200 scenarios, are:

$$OfflineArithCorr = \begin{pmatrix} NaN & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & .9977 & .9958 & .9975 & .9966 \\ 0 & .9977 & 1 & .9963 & .9968 & .9973 \\ 0 & .9958 & .9963 & 1 & .9968 & .9974 \\ 0 & .9975 & .9968 & .9968 & 1 & .9972 \\ 0 & .9966 & .9973 & .9974 & .9972 & 1 \end{pmatrix}$$

$$OptPolicyArithCorr = \begin{pmatrix} NaN & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & .9874 & .9886 & .9583 & .9404 \\ 0 & .9874 & 1 & .9934 & .9662 & .9486 \\ 0 & .9886 & .9934 & 1 & .9645 & .9429 \\ 0 & .9583 & .9662 & .9645 & 1 & .9886 \\ 0 & .9404 & .9486 & .9429 & .9886 & 1 \end{pmatrix}$$

These correlations are very high, the smallest being 99% for the offline values and 94% for the optimal policy values. Moreover, the minimal correlation for the optimal policy values becomes 98.7% when only decisions 2–4, which have the highest Q-value, are considered.

It remains to see whether these correlations are a characteristic of the problem or even of the instance. We conjecture that, instead, this will be the case in the overwhelming majority of X-MDPs originating from OSCO applications. Indeed, in most problems, some scenarios are more favorable than others regardless of the decisions. For example, in the S-RCPSP, scenarios with many successful projects bring more money than scenarios with many failures, exhibiting a positive correlation between the values of the actions. Finding realistic OSCO problems in which decisions are not positively correlated is hard. For example, in a portfolio management problem, one might expect the decision of selling some stock to be negatively correlated to the decision of buying more of the same stock. But this is not true: if trading fees are small and if the optimal policy is to buy more, then the optimal policy following the decision to sell the stock will be to rebuy them. They will be a loss due to fees and to the price difference between to sell and the rebuy, but there will be a positive correlation of values for the two decisions. As a result, we conjecture that, for most OSCO problems, exterior sampling will converge with far fewer scenarios than interior sampling.

6.3 Converting the X-MDP to an MDP

The second step of *Amsaa* consists of converting the approximated X-MDP into an MDP. It proceeds in two stages: First trimming the X-MDP to remove unreachable states, before performing the actual conversion.

6.3.1 Trimming X-MDPs

Trimming consists in eliminating unreachable states and in marking as final those state in which all the uncertainty has been revealed.

Definition 8 *Given an X-MDP A with state-space S and final states set F, the* trimmed *X-MDP B induced by A is the X-MDP that is in all equal to A, except:*

1. *its state space is $S' = \{s \in S | \mathcal{C}(s) \neq \varnothing\}$;*

2. *its set of final states set is $F' = F \cup \{s \in S' | \#\mathcal{C}(s) = 1\}$;*

3. *its feasible decision function $\mathcal{X}'(s)$ which is $\mathcal{X}(s)$ if $s \notin F' \setminus F$ and $\{\perp\}$ otherwise;*

4. *its reward function f' is defined over the states in $F' \setminus F$ and has value $f(s) = \mathcal{O}(s, \xi)$ where ξ is the unique scenario compatible with s.*

Lemma 6 *Let A be an X-MDP and B be its trimmed version. Then:*

1. *For any policy π in A optimal for states in $F' \setminus F$, we have for all $s \in S'$, $v_\pi^A(s) = v_\pi^B(s)$;*

2. *For any policy π in B, the policy $\tilde{\pi}$ in A defined by $\tilde{\pi}(s) = \pi(s)$ for $s \in S' \setminus (F' \setminus F)$ and by $\tilde{\pi}(s) = \operatorname{argmax}_{x \in \mathcal{X}(s)} \mathcal{O}(\tau(s, x, \xi), \xi)$ for states in $F' \setminus F$, with ξ the unique scenario compatible with s, satisfies $\forall s \in S', v_{\tilde{\pi}}^A(s) = v_\pi^B(s)$.*

6.3.2 Converting Trimmed X-MDPs

It remains to show how to transform a trimmed X-MDP into an MDP.

Definition 9 *Let $B = (S', s_0, F', X, \perp, \mathcal{X}', f, \boldsymbol{\xi}, \mu_\xi, \tau)$ be the trimmed version of X-MDP A. Define $\mathcal{P}$ from $S \times X$ to the set of probability distributions on X by:*

$$\forall s, s' \in S', x \in X \quad \mathcal{P}(s'|s, x) = \mathbb{P}\left(\tau(s, x, \boldsymbol{\xi}) = s' \mid \boldsymbol{\xi} \in \mathcal{C}(s)\right).$$

Then $C = (S', s_0, F', X, \perp, \mathcal{X}', f, \mathcal{P})$ is the MDP induced *by X-MDP A.*

Lemma 7 *For any policy π, we have $\forall s \in S$, $v^B(s) = v^C(s)$.*

This lemma is a consequence of the Markov property for X-MDPs, which implies that, following π in B or C, for all t the distribution of $\boldsymbol{s}_t$ is the same in B and in C.

Proof. (We assume S' finite here, which is the only case in which we use this theorem) Consider the random trajectory defined by running π in B starting in s, on the random scenario $\boldsymbol{\xi}$, denoted

$$s = \boldsymbol{s}_1^B \xrightarrow[\boldsymbol{\xi}]{\pi(\boldsymbol{s}_1^B)} \boldsymbol{s}_2^B \xrightarrow[\boldsymbol{\xi}]{\pi(\boldsymbol{s}_2^B)} \dots$$

and the one defined by running π in C starting in s, denoted

$$s = \boldsymbol{s}_1^C \xrightarrow{\pi(\boldsymbol{s}_1^C)} \boldsymbol{s}_2^C \xrightarrow{\pi(\boldsymbol{s}_2^C)} \dots.$$

By induction, we prove that the distributions of $\boldsymbol{s}_t^B$ and $\boldsymbol{s}_t^C$ are the same. This is true for $t = 1$. Now, suppose it is true at t. Consider s and s' in S'. We have

$$\begin{aligned}\mathbb{P}\left(\boldsymbol{s}_{t+1}^B = s' \;\middle|\; \boldsymbol{s}_t^B = s\right) &= \mathbb{P}\left(\tau(s, \pi(s), \boldsymbol{\xi}) = s' \;\mid\; \boldsymbol{\xi} \in \mathcal{C}(s)\right) \\ &= \mathcal{P}(s'|s, \pi(s)),\end{aligned}$$

with this last equality following the definition of $\mathcal{P}$. Now,

$$\begin{aligned}\mathbb{P}\left(\boldsymbol{s}_{t+1}^B = s'\right) &= \mathbb{E}\left[\mathbb{P}\left(\boldsymbol{s}_{t+1}^B = s' \mid \boldsymbol{s}_t^B\right)\right]\\ &= \mathbb{E}\left[\mathcal{P}(s'|\boldsymbol{s}_t^B, \pi(\boldsymbol{s}_t^B))\right]\\ &= \sum_{s\in S}\mathcal{P}(s'|s,\pi(s))\mathbb{P}\left(\boldsymbol{s}_t^B = s\right)\end{aligned}$$

and by induction hypothesis

$$\begin{aligned}&= \sum_{s\in S}\mathcal{P}(s'|s,\pi(s))\mathbb{P}\left(\boldsymbol{s}_t^C = s\right)\\ &= \mathbb{E}\left[\mathcal{P}(s'|\boldsymbol{s}_t^C, \pi(\boldsymbol{s}_t^C))\right]\end{aligned}$$

which, by definition of an MDP, is

$$= \mathbb{P}\left(\boldsymbol{s}_{t+1}^C = s'\right).$$

Hence $\boldsymbol{s}_{t+1}^B$ and $\boldsymbol{s}_{t+1}^C$ are equally distributed. By recursion, for $t = T$ we have $\mathbb{E}\left[f(\boldsymbol{s}_T^B)\right] = \mathbb{E}\left[f(\boldsymbol{s}_T^C)\right]$. Now, these two quantities are precisely $v_\pi^B(s)$ and $v_\pi^B(s)$, so they are equal. □

Theorem 4 *Let A be an X-MDP, B its trimmed version, and C the MDP induced by B.*

1. *For any policy π in A optimal for states in $F' \setminus F$, we have for all $s \in S'$, $v_\pi^A(s) = v_\pi^C(s)$;*
2. *For any policy π in C, the policy $\tilde{\pi}$ in A defined by $\tilde{\pi}(s) = \pi(s)$ for $s \in S' \setminus (F' \setminus F)$ and by $\tilde{\pi}(s) = \operatorname{argmax}_{x\in\mathcal{X}(s)} \mathcal{O}(\tau(s,x),\xi)$ for states in $F' \setminus F$, with ξ the unique scenario compatible with s, we have that $\forall s \in S', v_{\tilde{\pi}}^A(s) = v_\pi^C(s)$.*

Proof. Direct consequence of lemmas 6 and 7. □

6.4 Solving MDPs

Once the X-MDP is converted into an MDP, it is possible to apply existing algorithms for solving the MDP optimally. In this section, we describe such an algorithm from a class called *heuristic search algorithms*, which, despite their names, are exact algorithms. The presentation here follows [14] which contains an excellent synthesis of these algorithms.

6.4.1 Heuristic Search Algorithms for MDPs

Heuristic search algorithms for MDPs perform a partial exploration of the state space, using a — possibly monotone — upper bound to guide the search. A value function $h : S \to \mathbb{R}$ is an *upper bound* if $\forall s \in S,\ h(s) \geq v^\star(s)$. A value function is a *monotone upper bound* if it is an upper bound and if $Res_h(s) \geq 0$ for all state s. Intuitively, a monotone upper bound is an optimistic evaluation of a state that cannot become more optimistic if a Bellman update is performed.

Function `findAndRevise`, introduced by [13], captures the general schema of heuristic search algorithm for MDPs and returns an optimal value function upon termination. At each step, the

```
Function findRevise(MDP $A$)
  precondition: $h$ is a upper bound for $A$, $h(s) = f(s)$ if $s$ is final
1 foreach $s \in S$ do
2     $v(s) \leftarrow h(s)$
3 repeat
4     Pick a state $s$ reachable from $s_0$ and $\pi_v$ with $|Res_v(s)| > 0$
5     $v(s) \leftarrow \max_{x \in \mathcal{X}(s)} Q(s, x)$
6 until no such state is found
7 return $\text{argmax}_{x \in \mathcal{X}(s)} Q(s, x)$
```

algorithm selects a state reachable with the current policy π_v whose Bellman residual is non-zero and performs a Bellman update. When h is monotone, only strictly positive (instead of non-zero) Bellman residuals must be considered, and the value function v remains a monotone upper bound during the entire execution of the algorithm. Different instantiations of this generic schema differ in the choice of the state to reconsider. They include, among others, HDP [13], Learning Depth-First Search (LDFS) [14], Real-Time Dynamic Programming (RTDP) [2], Bounded RTDP [36], and LAO* [25]. Of course, since the state space may be extremely large, these instantiations only manipulate partial value functions defined on the states visited so far. It is only when a new state s is visited that the initialization $v(s) \leftarrow h(s)$ is performed. The rest of this section describes the acyclic LDFS algorithm and the upper bound used by *Amsaa*.

6.4.2 Learning Depth-First Search

Functions `LDFS`, `LDFSAux`, and `evalQ` describe the LDFS algorithm for acyclic MDPs. LDFS requires the upper bound to be monotone, so no Bellman residual is ever negative. The algorithm applies `LDFSAux` on state s_0 until the value $v(s_0)$ has converged. Function `LDFSAux` is recursive. When `LDFSAux` (s) is called, there are two possibilities:

- either $Res_v(s) = 0$, which means that there exists at least one decision $x \in \mathcal{X}(s)$ satisfying $v(s) = Q(s, x)$. In that case, LDFS performs a recursive call for each such decision and each of its possible transitions (lines 6–7). If these recursive calls all return true, the values of these successor states have converged and the value $v(s)$ has converged as well (lines 8–9). Otherwise, `LDFSAux` performs a Bellman update (line 10) and returns false (line 11).
- or $Res_v(s) > 0$. This means s is a candidate for an update (line 5 in `findAndRevise`). In that case, all the tests in line 4 fail and a Bellman update is performed in line 10.

`LDFS` explores the state-space in an iterative deepening fashion that comes from the behavior of `LDFSAux` on a state s explored for the first time. Indeed, when `LDFSAux`(s) is called for the first time for a given state s, its value has never been updated and thus $v(s) = h(s)$. But the Q-values computed by `evalQ` (line 4) are based on values $v(s')$ of each successor s' of s. As a result, it is likely that $Res_v(s)$ will be positive, and the path exploration will not go any deeper (because of the

Function `LDFS(MDP` A`)`

```
1 while not LDFSAux($s_0$) do { }
2 return $\mathrm{argmax}_{x \in \mathcal{X}(s_0)}$ evalQ($s_0, x$)
```

Function `LDFSAux(State` s`)`

```
1 if $solved[s]$ then
2     return True
3 foreach $x \in \mathcal{X}(s)$ do
4     if evalQ($s, x$) $= v(x)$ then
5         $solved[s] \leftarrow$ True
6         foreach $t \in \{s' \in S | \mathcal{P}(s'|s,x) > 0\}$ do
7             $solved[s] \leftarrow solved[s] \wedge$ LDFSAux($t$)
8         if $solved[s]$ then
              // $x$ is the optimal decision, and $v(x) = v^\star(x)$
9             return True
  // Not solved yet
10 $v(s) \leftarrow \max_{x \in \mathcal{X}(s)}$ evalQ($s, x$)
11 return False
```

Function `evalQ(State` s`,Decision` x`)`

```
1 $Succ \leftarrow \{s' \in S | \mathcal{P}(s'|s,x) > 0\}$;
2 foreach $s' \in Succ$ do
3     if $\neg visited[s']$ then
4         $visited[s'] \leftarrow$ True;
5         $solved[s'] \leftarrow$ ($s'$ is final);
6         $v(s') \leftarrow h(s')$;
7 return $\sum_{s' \in Succ} v(s')\mathcal{P}(s'|s,x)$
```

test in line 4). This contrasts with RTDP-like algorithms, which always explore paths deeply in the state space. LDFS is an attractive algorithm for use in *Amsaa* for several reasons:

1. it is applicable since our problems are acyclic and a good monotone upper bound is available;
2. *Amsaa*'s upper bound is reasonably strong but expensive to compute, justifying why the iterative deepening feature is interesting. Once the successors of a state are evaluated, the state may no longer be reachable under π_v and there is no immediate benefit in further exploration.
3. the *solved* flags allow the algorithm to test for convergence, as opposed to RTDP;
4. its code is simpler than those of HDP and of LAO*, since it exploits acyclicity.

6.4.3 The Upper Bound $h_{\mathbb{E},\max}$

The performance of heuristic search algorithms strongly depends on the quality of the heuristic function h. A standard upper bound, that is defined for any MDP, is called $h_{\max,\max}$. For a state s,

$h_{\max,\max}(s)$ is the highest reward of a final state reachable from s. In other words, it correspond to a relaxation of the problem in which the decision maker can choose not only its decisions, but also random outcomes.

For very stochastic problems, this is often a poor bound. On the standard instance of the S-RCPSP, $h_{\max,\max}(s_0)$ is about 4 times the value of the optimal policy $v^\star(s_0)$. Fortunately, for MDPs induced by X-MDPs, a much better heuristic function can be derived from the deterministic offline problems (see Definition 3). More precisely, for a state s, the heuristic consists of solving the deterministic offline problems for the scenarios compatible with s in the original X-MDP and taking the resulting expected offline value , i.e.,

$$h_{\mathbb{E},\max}(s) = \mathbb{E}_\mu\left[\mathcal{O}(s,\boldsymbol{\xi}) \mid \boldsymbol{\xi} \in \mathcal{C}(s)\right],$$

where μ is $\boldsymbol{\xi}$'s distribution. Function $h_{\mathbb{E},\max}$ is a good heuristic because it leverages both the combinatorial and stochastic structures of the application. Moreover, on the approximated problem, the sets $\mathcal{C}(s)$ are small and the expectation can be computed exactly as a sum, which is why the approximation was introduced in the first place. It remains to show that $h_{\mathbb{E},\max}(s)$ is an upper bound for state s in the induced MDP.

Theorem 5 *Let $A = (S, s_0, F, X, \perp, \mathcal{X}, f, \boldsymbol{\xi}, \mu, \tau)$ be an X-MDP and B be the induced MDP with transition probability function $\mathcal{P}(\cdot|s,x)$. Then $h_{\mathbb{E},\max}$ is a monotone upper bound for B.*

Proof. (for simplicity we assume S and Ξ finite. The theorem holds without these assumption, but we only use it in that case). Let s be a state, and $x \in \mathcal{X}(s)$. Then

$$\begin{aligned}
Q_{h_{\mathbb{E},\max}}(s,x) &= \sum_{s' \in S} h_{\mathbb{E},\max}(s')\mathcal{P}(s'|s,x) && \text{by definition of a Q-value}\\
&= \sum_{s' \in S} \mathbb{E}_\mu\left[\mathcal{O}(s',\boldsymbol{\xi}) | \boldsymbol{\xi} \in \mathcal{C}(s')\right]\mathcal{P}(s'|s,x) && \text{by definition of } h_{\mathbb{E},\max}\\
&= \sum_{s' \in S} \mathbb{E}_\mu\left[\mathcal{O}(s',\boldsymbol{\xi}) | \boldsymbol{\xi} \in \mathcal{C}(s')\right]\mathbb{P}\left(\tau(s,x,\boldsymbol{\xi}) = s' | \boldsymbol{\xi} \in \mathcal{C}(s)\right) && \text{by definition of } \mathcal{P}\\
&= \sum_{\substack{s' \in S \ \mid \\ \mu(\mathcal{C}(s'))>0}} \left(\sum_{\xi \in \mathcal{C}(s')} \frac{\mu(\xi)}{\mu(\mathcal{C}(s'))}\mathcal{O}(s',\xi)\right)\frac{\mu(\mathcal{C}(s'))}{\mu(\mathcal{C}(s))}\\
&= \sum_{\substack{s' \in S \ \mid \\ \mu(\mathcal{C}(s'))>0}} \sum_{\xi \in \mathcal{C}(s')} \frac{\mu(\xi)}{\mu(\mathcal{C}(s))}\mathcal{O}(\tau(s,x,\xi),\xi)\\
&= \sum_{\xi \in \mathcal{C}(s)} \frac{\mu(\xi)}{\mu(\mathcal{C}(s))}\mathcal{O}(\tau(s,x,\xi),\xi) && \text{since } \{\mathcal{C}(\tau(s,x,\xi)), \xi \in \mathcal{C}(s)\} \text{ partitions } \mathcal{C}(s)\\
&= \mathbb{E}_\mu\left[\mathcal{O}(\tau(s,x,\boldsymbol{\xi}),\boldsymbol{\xi}) \mid \boldsymbol{\xi} \in \mathcal{C}(s)\right]
\end{aligned}$$

Now, by definition of the offline problem, we have $\forall \xi \in \mathcal{C}(s),\ \mathcal{O}(s,\xi) = \max_{x \in \mathcal{X}(s)} \mathcal{O}(\tau(s,x,\xi),\xi)$, and thus $\forall x \in \mathcal{X}(s),\ \mathcal{O}(\tau(s,x,\xi),\xi) \leq \mathcal{O}(s,\xi)$. Therefore,

$$Q_{h_{\mathbb{E},\max}}(s,x) \leq \mathbb{E}_\mu\left[\mathcal{O}(s,\boldsymbol{\xi}) \mid \boldsymbol{\xi} \in \mathcal{C}(s)\right] = h_{\mathbb{E},\max}(s).$$

This proves that $Res_{h_{\mathbb{E},\max}}(s) \geq 0$ for all states. Moreover, if s is final, we have for all scenario ξ, $\mathcal{O}(s,\xi) = f(s) = v^\star(s)$. The result follows since a value function v satisfying $Res_v \geq 0$ for all states are upper bounds [36]. $\square$

6.5 Example of a Run

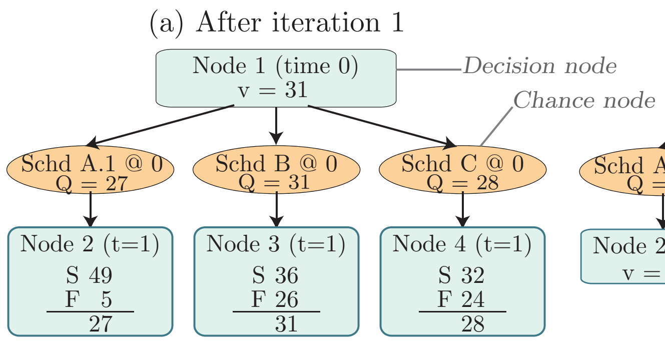
To clarify the previous description of the `Msaa` algorithm, we will now go through a run step by step. The instance considered in the previously presented small instance of the S-RCPSP on which *1s-AA* is suboptimal depicted in figure 5.2. `Msaa` (with Learning DFS) needs four iterations to terminates. Like when we studied the behavior of *1s-AA* on the same instance, we assume that the considered sample has one of each scenario: one where A succeeds (denoted S), and one where it fails (denoted F). The explored state space, with current values of decisions and Q-values are depicted in figure 6.1.

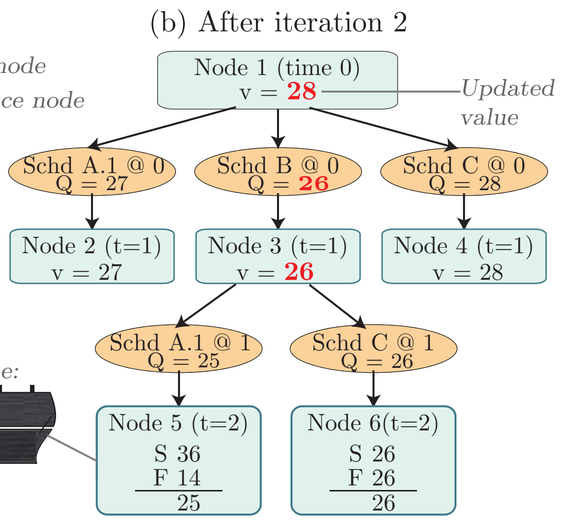
6.6 Incrementality and Anytime Decision Making

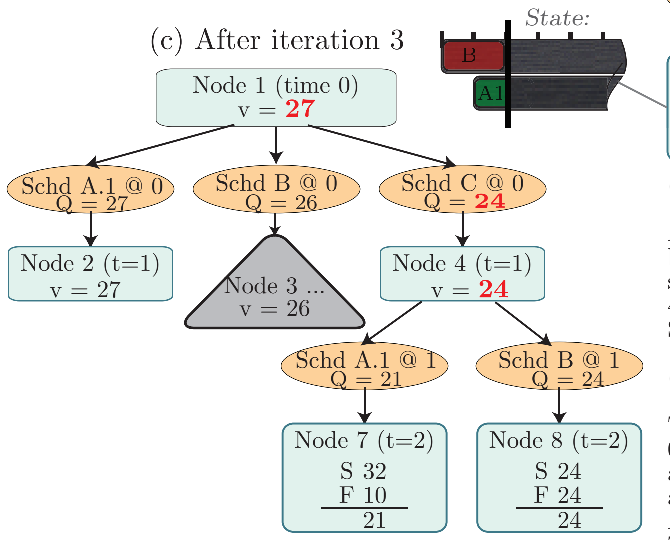
Amsaa is an incremental algorithm: It reuses earlier computations when computing the optimal policy of a finer approximation to the original X-MDP. Its incremental nature stems from two essential components: (1) a good *incremental upper bound*, that is, an upper bound for the new approximation that derives from the optimal policy values of the earlier approximation; and (2) the reuse of the internal data structures. Note also that incrementality is not only beneficial for runtime performances; It is also opens the door to sequential importance sampling [20] and to the contamination method [22].

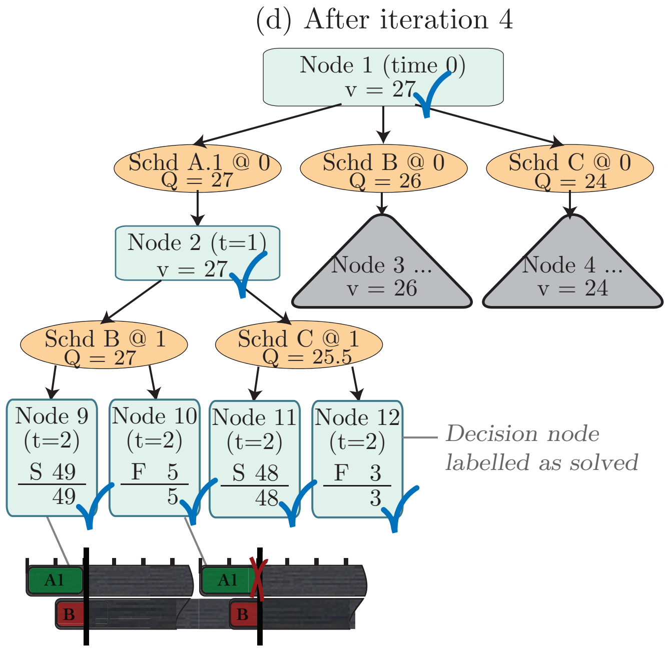
6.6.1 Incremental Upper Bound

To convey the intuition behind the incremental upper bound, consider the following example under iid sampling. We have solved the approximated problem with distribution μ that gives weights $1/3$ to scenarios a, b and c and we are interested in adding two scenarios d and e to obtain a solution to the finer approximation with distribution ρ in which each of these 5 scenarios has weight $1/5$. Consider a state s which is compatible with a, b, and d. Assume that LDFS has shown that $v^\star_\mu(s) \leq 3$, and assume $\mathcal{O}(s,d) = 6$. How can we compute an upper bound of $v^\star_\rho(s)$? If a scenario generated by ρ is in $\mathcal{C}(s)$, then it is in $\{a,b\}$ with probability $p = \rho(\{a,b\})/\rho(\{a,b,d\})$ and is d with probability $1-p$. Let ν be the probability distribution that gives probability $1/2$ to each of the two new scenarios d and e. For the problem with distribution ρ, consider a decision maker that can query whether the actual scenario is in $\{a,b,c\}$. Depending on the answer, she will follow the policy defined by $v^\star_\mu$ or by $v^\star_\nu$. Therefore, its expected value if s is $pv^\star_\mu(s) + (1-p)v^\star_\nu(s)$. Of course, the decision maker has no access to this information: this assumption relaxes of the problem, and so the expected value of this "partially clairvoyant" decision maker is an upper bound of $v^\star_\rho(s)$. It follows that









(a) First iteration

At the first iteration, the offline upper bound is computed for the immediate successors of the root node: nodes 2, 3, and 4. For each of these nodes, the two scenarios S and F are compatible.

(b) Second iteration

The most promising branch, scheduling B at 0, is explored. The transitions after node 3 are deterministic because they go to nodes 5 and 6, in which only B has comleted, and A.1 or C is running.
The value of node 3 goes down from 31 to 26 and scheduling C at 0 becomes the most promising decision, with an upper bound of 28.

(c) Third iteration

Like at iteration 2, the exploration decreases the upper bound. Here the value of node 4 goes down to 24, and now scheduling A.1 at zero appears the most promising.

(d) Fourth iteration

The exploration of this branch is different, because after scheduling A.1 at zero and B or C at 1, task A.1 completes and is thus observed. Therefore, nodes 9 to 12 only have one compatible scenarios, instead of two for all the previous ones. Hence their respective upper bound can be labeled as tight, and by dynamic programming this labeled is carried upward to the root node.

Figure 6.1: A run of `Msaa`

$v^\star_\rho(s) \leq p \cdot 3 + (1-p) \cdot 6 = 2 + 2 = 4$. The incremental upper bound theorem below formalizes this idea. To connect it with this example, note that $\rho = \frac{3}{5}\mu + \frac{2}{5}\nu$ and $p = {}^{3}/_{5} \cdot {}^{\mu(\mathcal{C}(s))}/_{\rho(\mathcal{C}(s)}$.

Given two distributions μ and ρ, we say that μ is *absolutely continuous* with respect to ρ if, for any set A such that $\rho(A) = 0$, $\mu(A) = 0$. In that case, we define the function $\frac{\mu}{\rho}$ by

$$\frac{\mu}{\rho}(A) = \begin{cases} \frac{\mu(A)}{\rho(A)} & \text{if } \rho(A) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

This function ensures that $\mu(A) = \frac{\mu}{\rho}(A)\rho(A)$ for all A.

Theorem 6 *Let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be three X-MDPs that differ only by their respective distributions μ, ν, and ρ and let $\rho = \lambda\mu + (1-\lambda)\nu$ for some $0 < \lambda < 1$. Let h_μ and h_ν be monotone upper bounds for $\mathcal{A}$ and $\mathcal{B}$ respectively. Define $h : S \to \mathbb{R}$ by:*

$$h(s) = \lambda\frac{\mu}{\rho}(\mathcal{C}(s))\ h_\mu(s) \quad + \quad (1-\lambda)\frac{\nu}{\rho}(\mathcal{C}(s))h_\nu(s).$$

Then h is well-defined and is a monotone upper bound for the induced MDP of $\mathcal{C}$.

Proof. First, h is well-defined because μ and ν are absolutely continuous wrt ρ. Let $x \in \mathcal{X}(s)$ and define $\boldsymbol{s}'$ as the random variable $\tau(s, x, \boldsymbol{\xi})$. Consider $Q_h(s, x)$, the Q-value in $\mathcal{C}$ for the value function h. We have:

$$\begin{aligned} Q_h(s,x) &= \mathbb{E}_\rho\left[h(\boldsymbol{s}')|\boldsymbol{\xi} \in \mathcal{C}(s)\right] \\ &= \mathbb{E}_\rho\left[\lambda\frac{\mu}{\rho}(\mathcal{C}(\boldsymbol{s}'))h_\mu(\boldsymbol{s}') + (1-\lambda)\frac{\nu}{\rho}(\mathcal{C}(\boldsymbol{s}'))h_\rho(\boldsymbol{s}')\middle|\boldsymbol{\xi} \in \mathcal{C}(s)\right] \\ &= \lambda\mathbb{E}_\rho\left[\frac{\mu}{\rho}(\mathcal{C}(\boldsymbol{s}'))h_\mu(\boldsymbol{s}')\middle|\boldsymbol{\xi} \in \mathcal{C}(s)\right] + (1-\lambda)\mathbb{E}_\rho\left[\frac{\nu}{\rho}(\mathcal{C}(\boldsymbol{s}'))h_\rho(\boldsymbol{s}')\middle|\boldsymbol{\xi} \in \mathcal{C}(s)\right] \\ &= \lambda\frac{\mu}{\rho}(\mathcal{C}(s))\mathbb{E}_\mu\left[h_\mu(\boldsymbol{s}')|\boldsymbol{\xi} \in \mathcal{C}(s)\right] + (1-\lambda)\frac{\nu}{\rho}(\mathcal{C}(s))\mathbb{E}_\nu\left[h_\rho(\boldsymbol{s}')|\boldsymbol{\xi} \in \mathcal{C}(s)\right]. \end{aligned}$$

Consider now the difference between $h(s)$ and $Q_h(s, x)$:

$$\begin{aligned} h(s) - Q_h(s,x) &= \lambda\frac{\mu}{\rho}(\mathcal{C}(s))\Big(h_\mu(s) - \mathbb{E}_\mu\left[h_\mu(\boldsymbol{s}')|\boldsymbol{\xi} \in \mathcal{C}(s)\right]\Big) \\ &\quad + (1-\lambda)\frac{\nu}{\rho}(\mathcal{C}(s))\Big(h_\nu(s) - \mathbb{E}_\nu\left[h_\rho(\boldsymbol{s}')|\boldsymbol{\xi} \in \mathcal{C}(s)\right]\Big) \\ &\geq \lambda\frac{\mu}{\rho}(\mathcal{C}(s))Res^{\mathcal{A}}_{h_\mu}(s) + (1-\lambda)\frac{\nu}{\rho}(\mathcal{C}(s))Res^{\mathcal{B}}_{h_\nu}(s) \geq 0, \end{aligned}$$

where $Res^{\mathcal{A}}_{h_\mu}(s)$ (resp. $Res^{\mathcal{B}}_{h_\nu}(s)$) is the residual in $\mathcal{A}$ (resp. $\mathcal{A}$) of value function h_μ (resp. h_ν). These residuals are non-negative because h_μ and h_ν are monotone, Since this inequality holds for all x, it follows that $Res_h(s) \geq 0$. This proves the monotonicity of h, that h is also an upper bound. □

Amsaa applies this theorem as follows. The empirical distribution of the X-MDP used in the previous iteration is μ and the empirical distribution of the new scenarios is ν. An optimal policy for $\mathcal{A}$ has been computed with the upper bound $h^\mu_{\mathbb{E},\max}$ (the bound $h_{\mathbb{E},\max}$ for the problem with distribution μ) and the returned value function $v_\mathcal{A}$ is a monotone upper bound for $\mathcal{A}$ which is optimal for all states reachable under policy $\pi_{v_\mathcal{A}}$. The function $v_\mathcal{A}$ corresponds to h_μ in the theorem. The function h_ν is $h^\nu_{\mathbb{E},\max}$ and is computed by solving the offline problem on each new scenario. When the number of new scenarios is small compared to the previous sample size, the incremental upper bound h is:

- optimal for any state s reachable under $\pi_{v_\mathcal{A}}$ satisfying $\nu(\mathcal{C}(s)) = 0$;
- tight for most states s reachable under $\pi_{v_\mathcal{A}}$, because the term $(1-\lambda)\frac{\nu}{\rho}(\mathcal{C}(s))h^\nu_{\mathbb{E},\max}$ will be usually small compared to $\lambda\frac{\mu}{\rho}(\mathcal{C}(s))\ v_\mathcal{A}(s)$;
- at least as good as $h^\rho_{\mathbb{E},\max}$ everywhere because $v_\mathcal{A}(s) \leq h^\mu_{\mathbb{E},\max}(s)$ for all states s.

Computing $\lambda\frac{\mu}{\rho}(\mathcal{C}(s))$ or $(1-\lambda)\frac{\nu}{\rho}(\mathcal{C}(s))$ for a state s is easy. In the case of iid sampling, it suffices to count how many scenarios in $\mathcal{C}(s)$ are present in the earlier iteration and how many are being added. For example, if μ is based on 90 scenarios and ν on 10, then $\lambda\frac{\mu}{\rho}(\mathcal{C}(s_0)) = \frac{90}{90+10} \cdot \frac{1.0}{1.0} = 0.9$ at the root node. Obviously, this theorem is most useful for states s where $\lambda\frac{\mu}{\rho}(\mathcal{C}(s))$ is close to one.

6.6.2 Objects reuse

To make efficient use of this incremental upper bound, *Amsaa* must reuse data structures created when solving $\mathcal{A}$. Indeed, when adding a small number of new scenarios, only a small number of states will be affected and it would be inefficient to compute the incremental upper bound of all explored states. Instead *Amsaa* updates states lazily. The iterative refinement define a sequence of X-MDPs $A_1, \ldots, A_n, \ldots$. Each state stores a *time stamp*, that is, an integer i stating that the stored information is valid wrt to A_i. When a out-of-date state is encountered (its time stamp is smaller than the index of the current X-MDP), it is updated.

6.7 Theoretical Results on the Sampling Average Approximation for X-MDPs

The SAA method was developed for stochastic programs. Fortunately, several results for multistage programs can be adapted to X-MDPs. We present two such results.

The first result is important to evaluate the solution quality of *Amsaa*. Indeed, The SAA method provides a positively biased estimator for the objective value of stochastic programs. This stems from the fact that the same sample is used to find a policy and to evaluate it, which [34] calls appropriately "inside information", producing decisions optimized for the sample, not for the whole scenario space. The following theorem extends this result to X-MDPs. Some technicalities, in particular the fact that the number of scenarios compatible with a given state is random, pollute this otherwise simple proof.

Theorem 7 (Positive bias) *Let A be a finite-horizon X-MDP and denote by $\hat{\boldsymbol{A}}_n$ the random X-MDP obtained by sampling n replications $\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^n$ of $\boldsymbol{\xi}$ (independence is not required). For a state*

$s \in S$, denote by $\hat{\boldsymbol{v}}_n(s)$ the value of s in $\hat{\boldsymbol{A}}_n$ when it exists. Then, for any $n \in \mathbb{N}$, we have:

$$\mathbb{E}\left[\hat{\boldsymbol{v}}_n(s_0)\right] \;\geq\; v(s_0).$$

Proof. First, define the *stage* of a state s as T, the horizon, minus the highest number of transitions from s to a final state in A. The proof establishes the following property by induction on the stage t: " Let $s \in S$ be a state at stage t such that $\mathcal{C}(s) \neq \varnothing$. For all $0 < k \leq n$, $\mathbb{E}\left[\hat{\boldsymbol{v}}_n(s) | \#\boldsymbol{\mathcal{C}}_n(s) = k\right] \;\geq\; v(s)$. "

All final states satisfying $\hat{\boldsymbol{v}}(s) = f(s) = v(s)$, the property holds at the last stage T. Consider now a stage $t < T$ and suppose the property holds at $t+1$. Consider a random sample problem $\hat{\boldsymbol{A}}_n$ and a state s in stage t that is non-final in A and such that $\#\boldsymbol{\mathcal{C}}_n(s) = k$. Denote the scenarios in $\boldsymbol{\mathcal{C}}_n(s)$ by $\boldsymbol{\xi}^{s,1}, \ldots, \boldsymbol{\xi}^{s,k}$. and define for $x \in \mathcal{X}(s)$

$$\hat{\boldsymbol{g}}(x) = \frac{1}{k}\sum_{i=1}^{k} \hat{\boldsymbol{v}}(\boldsymbol{s}^{x,i}),$$

where $\boldsymbol{s}^{x,i} = \tau(s, x, \xi^i)$. By definition of max, we have $\hat{\boldsymbol{g}}(x') \leq \max_{x \in \mathcal{X}(s)} \hat{\boldsymbol{g}}(x)$ for all $x' \in \mathcal{X}(s)$. By taking a conditional expectation on both sides, we obtain

$$\mathbb{E}\left[\hat{\boldsymbol{g}}(x') \mid \#\boldsymbol{\mathcal{C}}_n(s) = k\right] \leq \mathbb{E}\left[\max_{x \in \mathcal{X}(s)} \hat{\boldsymbol{g}}(x) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right], \tag{6.1}$$

and by taking the max over x',

$$\max_{x \in \mathcal{X}(s)} \mathbb{E}\left[\hat{\boldsymbol{g}}(x) \mid \#\boldsymbol{\mathcal{C}}_n(s) = k\right] \leq \mathbb{E}\left[\max_{x \in \mathcal{X}(s)} \hat{\boldsymbol{g}}(x) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right]. \tag{6.2}$$

Now, we have

$$\begin{aligned}
\mathbb{E}\left[\hat{\boldsymbol{v}}_n(s) \mid \#\boldsymbol{\mathcal{C}}_n(s) = k\right] &= \mathbb{E}\left[\max_{x \in \mathcal{X}(s)} \hat{\boldsymbol{g}}(x) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right] && (6.3)\\
&\geq \max_{x \in \mathcal{X}(s)} \mathbb{E}\left[\hat{\boldsymbol{g}}(x) \mid \#\boldsymbol{\mathcal{C}}_n(s) = k\right] && (6.4)\\
&= \max_{x \in \mathcal{X}(s)} \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} \hat{\boldsymbol{v}}(\boldsymbol{s}^{x,i}) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right] && (6.5)
\end{aligned}$$

and, by linearity,

$$= \max_{x \in \mathcal{X}(s)} \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}\left[\hat{\boldsymbol{v}}(\boldsymbol{s}^{x,i}) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right]. \tag{6.6}$$

Decompose this last conditional expectation as follows:

$$\mathbb{E}\left[\hat{\boldsymbol{v}}(\boldsymbol{s}^{x,i}) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right] = \mathbb{E}\left[\mathbb{E}\left[\hat{\boldsymbol{v}}(\boldsymbol{s}^{x,i}) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(\boldsymbol{s}^{x,i})\right] \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right] \tag{6.7}$$

Now, thanks to the Markov property for X-MDPs, $\hat{\boldsymbol{v}}(\boldsymbol{s}^{x,i})$ is independent of $\boldsymbol{\mathcal{C}}_n(s) \setminus \boldsymbol{\mathcal{C}}_n(\boldsymbol{s}^{x,i})$, so we can apply the induction hypothesis to the inner expectation for each possible value of $\#\boldsymbol{\mathcal{C}}_n(\boldsymbol{s}^{x,i})$:

$$\mathbb{E}\left[\hat{\boldsymbol{v}}(\boldsymbol{s}^{x,i}) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right] \geq \mathbb{E}\left[v(\boldsymbol{s}^{x,i}) \;\middle|\; \#\boldsymbol{\mathcal{C}}_n(s) = k\right] \tag{6.8}$$

and, because the scenarios are replications of $\boldsymbol{\xi}$,

$$= \mathbb{E}\left[v(\tau(s,x,\boldsymbol{\xi})) \mid \boldsymbol{\xi} \in \mathcal{C}(s)\right] \tag{6.9}$$

Combining equations 6.6 and 6.9 leads to

$$\mathbb{E}\left[\hat{\boldsymbol{v}}_n(s) \mid \#\boldsymbol{\mathcal{C}}_n(s) = k\right] \geq \max_{x\in\mathcal{X}(s)} \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}\left[v(\tau(s,x,\boldsymbol{\xi})) \mid \boldsymbol{\xi} \in \mathcal{C}(s)\right] \tag{6.10}$$

$$= \max_{x\in\mathcal{X}(s)} \mathbb{E}\left[v(\tau(s,x,\boldsymbol{\xi})) \mid \boldsymbol{\xi} \in \mathcal{C}(s)\right] \tag{6.11}$$

$$= v(x). \tag{6.12}$$

This shows that the induction property holds for state s in stage t. The theorem follows from the property at state s_0 and because $\#\boldsymbol{\mathcal{C}}_n(s_0) = n$. □

The second result is mostly of theoretical interest. Let A be an X-MDP, v be its optimal value, and $\mathbb{X} \subseteq \mathcal{X}(s_0)$ be the set of optimal initial decisions. Define equally $\hat{\boldsymbol{v}}_n$ and $\hat{\mathbb{X}}_n$ as the optimal value and set of optimal initial decisions of the random X-MDP $A_{\hat{\boldsymbol{\mu}}_n}$. The SAA theory is concerned with the convergence of $\hat{\boldsymbol{v}}_n$ to v and of $\hat{\mathbb{X}}_n$ to $\mathbb{X}$ in appropriate senses. The following theorem proves this convergence when Ξ is finite, generalizing a well-known results from stochastic programs to X-MDPs. Unfortunately, the proof requires more scenarios that there are in Ξ, defeating the purpose of sampling. For this reason, we only sketch the proof.

Theorem 8 (Estimator Consistency) *Let A be an X-MDP such that Ξ is finite, and $\hat{\boldsymbol{A}}_n$ the random X-MDP obtained by iid sampling of n scenarios. Then, almost surely:*

$$\hat{\boldsymbol{v}}_n \to v(s_0) \tag{6.13}$$

$$\#\left(\hat{\mathbb{X}}_n \setminus \mathbb{X}\right) \to 0 \tag{6.14}$$

Proof.(Sketch) This theorem relies on the strong law of large numbers. Consider, for $x \in \mathcal{X}(s_0)$, the value $Q(s_0,x)$ of x in A, and similarly the value $\hat{\boldsymbol{Q}}_n(s_0,x)$ of x in $\hat{\boldsymbol{A}}_n$. Recall that $v = \max_{x\in\mathcal{X}(s_0)} Q(s_0,x)$ and $\hat{\boldsymbol{v}}_n = \max_{x\in\mathcal{X}(s_0)} \hat{\boldsymbol{Q}}_n(s_0,x)$. We will prove that for all $x \in \mathcal{X}(s_0)$, $\hat{\boldsymbol{Q}}_n(s_0,x)$ converges almost surely to $Q(s_0,x)$. By finiteness of $\mathcal{X}(s_0)$, this implies eq.6.13.

Moreover, let $\varepsilon > 0$ be such that, for all $x \in \mathcal{X}(s_0)$, either $Q(s_0,x) = v$, or $Q(s_0,x) < v - 3\varepsilon$. If all the $\hat{\boldsymbol{Q}}_n(s_0,x)$ converge almost surely, then there is a random variable $\boldsymbol{N}$ such that $\forall x \in \mathcal{X}(s_0)$, $\forall n \geq \boldsymbol{N}$, $\left|\hat{\boldsymbol{Q}}_n(s_0,x) - Q(s_0,x)\right| < \varepsilon$, and $\boldsymbol{N}$ is almost surely finite. Let $n > \boldsymbol{N}$. For any $x \notin \mathbb{X}$ we have $\hat{\boldsymbol{Q}}_n(s_0,x) < Q(s_0,x) + \varepsilon < v - 2\varepsilon$ and, for any $y \in \mathbb{X}$ and $n \geq \boldsymbol{N}$, $\hat{\boldsymbol{Q}}_n(s_0,y) > Q(s_0,y) - \varepsilon > v - \varepsilon$. It follows that $\hat{\boldsymbol{Q}}_n(s_0,x) < \hat{\boldsymbol{Q}}_n(s_0,y)$, and so $x \notin \mathbb{X}_n$. Hence, proving that the $\hat{\boldsymbol{Q}}_n(s_0,x)$ converge almost surely also implies eq.6.14.

It remains to prove that the $\hat{\boldsymbol{Q}}_n(s_0,x)$ converge almost surely. Thanks to the strong law of large numbers, we have for all $s \in S$, and $Z \in \Xi$:

$$\mathbb{P}\left(\boldsymbol{\xi}_{\hat{\boldsymbol{\mu}}_n} \in Z \mid \boldsymbol{\xi}_{\hat{\boldsymbol{\mu}}_n} \in \mathcal{C}(s)\right) \to \mathbb{P}\left(\boldsymbol{\xi} \in Z \mid \boldsymbol{\xi} \in \mathcal{C}(s)\right),$$

and the proof is a backward induction on the property that $\hat{\boldsymbol{v}}_n(s) \to v(s)$ using this fact. □

Chapter 7

Empirical Results

Having described in details how *Amsaa* produces decision in limited time for MDPs, we are now in position to report experimental results on the diverse problems we considered.

7.1 Stochastic Multiknapsack

As mentioned previously, on the Stochastic Multiknapsack problem (SMK), the one-step algorithm performs very well, even compared to the expected value of the clairvoyant. As a result, there is very little room for improvement. We compare *1s-AA-E* and *Amsaa* on an instance called bbcr5_T30_nc, which is very close to the one mentioned in chapter 5, bbcr5nc, but in which time is discretized: there is 30 time periods. On this instance, there are 5 bins to accomodate the items. At each time period, exactly one item arrives, with uniform probability among 5 item types.

We compare *1s-AA-E* and *Amsaa* in an anytime fashion, using respectively 4s and 16s per decisions, and a 2s time limit for offline problems. In the vast majority of cases, this time limit is not reached: CPLEX solves most (offline) multiknapsack problems to optimality in about 0.1 to 0.2s.

The reason a discrete time instance was used is that this way, independent and identically distributed (iid) sampling can be used. Indeed, if items arrive in continuous time, and iid sampling is used, than *Amsaa* reduces to *1s-AA-E*, because the probability that non-anticipativity constraints will have to be enforced in null. In case of promising results, more study on continuous time, using non-iid sampling, would be done.

On this instance, the expected value of the clairvoyant lies in $[540.2, 543.7]$ with 95% confidence. The following table reports the empirical expected value using the two compared algorithms:

time per decision	4 (s)	16 s
1s-AA-E	526.2	526.6
Amsaa	525.8	526.8
p-value *Amsaa* better than *1s-AA-E*	19%	84%

For both decision time, the expected value of the two algorithms are extremely close, and, despite using 1,000 runs runs, the difference is not statically significant. One might wonder whether *Amsaa* does not get better scores because the stochastic model makes that there are few non-anticipativity to enforce with iid sampling. But this is not the case. Indeed, with 16s per decision, *Amsaa* uses on average 15 scenarios to make its first decision, and this causes the average depth of the leaves of the solution state space to be about 2.5 (the root being at depth zero). That 1.5 units more than if all scenarios causes transitions to different states at the first transition.

Moreover, for later decisions, offline problems get easier because the remaining capacities decreases. For instance, at the 15th period, that is, at the middle of the runs, *Amsaa* uses on average 82 scenarios, and the average depth of the leaves of the optimal policy for the sample problem goes up to 3.6.

7.2 Stochastic Resource Constraint Project Scheduling Problem

This section describes the experimental results on *Amsaa* is organized in four main parts. We first report experimental results about the quality of the decisions produced by *Amsaa* and several other algorithms under various time constraints. We then study *Amsaa*'s behavior in more details and discusses its computational complexity and its convergence in practice. A comparison of *Amsaa* and a mathematical programming approach to solve the SAA problem is then given for completeness. Finally, we compare *Amsaa* to gap-reduction techniques for one-step anticipatory algorithms.

7.2.1 Quality of Anytime Decisions

Experimental Setting

The benchmarks are based on the collection of 12 instances for the S-RCPSP from [17] described below. For each instance, 1,000 realizations of the uncertainty were generated. A *run* of an algorithm on a realization consists of simulating one trajectory in the X-MDP. At each encountered state, the online algorithm takes a decision with hard time constraints. If the online algorithm has not enough time to decide, a default decision, closing the labs, is applied. The algorithms were tested on all the realizations and various time limits. With 4 tested algorithms and time limits of 31 ms, 125 ms, 500 ms, 2s, 8s, and 32s, this gives a total of 288,000 runs. With more than 10 decisions on average per run, this represents more than $4 \times 12 \times 1000 \times (31ms + 125ms + .5s + 8s + 32s) \times 10 = 2,000$ hours of cpu time.

The 12 Instances The benchmarks are based on the the collection of instances for the S-RCPSP [17] defined in [21]. The reference instance has two laboratories and 5 projects, each of which having 3 or 4 tasks, giving a total of 17 tasks. The number of realizations for each task range from 3 to 7, giving a total of $1.2 \cdot 10^9$ possible scenarios. The different variants are the following:

1. Reg: the reference instance;

2. Agr: the various realizations of a given task corresponding to a failure (resp. success) are merged into a single realization, whose cost and duration are the averages of the original realizations. In other word, each task has a most two realizations, one for success and one for failure.

3. Cost2 and Cost5: the costs of the tasks are scaled by a factor 2 and 5 respectively.

4. D.6 and D1.5: the revenue for completing a project at time t in D.6 (resp. D1.5) is the one for completing the same project at time $t/0.66$ (resp. $t/1.5$) in instance Reg.

5. PX (P1, P2, P3 and P4): The last X tasks of each project do not fail. For instance, in P3, 3-task projects never fail and 4-task projects can only fail at the first task.

6. R.6 and R1.5: the revenues are scaled by a factor 0.66 and 1.5 respectively (equivalent to choiCost1.5 and choiCost0.66).

These instances explore various tradeoffs between the combinatorial and stochastic aspects and specific algorithms may exhibit radically different behaviors on some of them. Note that Cost5 is a pathological instance for which we can prove that the optimal policy is to schedule no project.

The Compared Algorithms. We report experimental results for four algorithms implemented in Java and sharing significant code.

- *Amsaa* is used with iid sampling and sample sizes growing by increments of 10%. It uses the branch and bound algorithm from [21] which we describe below.

- *1s-AA* is the one-step anticipatory algorithm with iid sampling. It uses the same offline solver as *Amsaa*.

- B-RTDP is a variant of the Bounded Real-Time Dynamic Programming algorithm [36], in which the decision is taken greedily with respect to upper bounds, instead of lower bounds as in the original algorithm. The lower bound $h^{-}(s)$ correspond to scheduling no project after state s. The upper bound is $h^{+}(s)$ is a very slight relaxation of $h_{\max,\max}$: It uses the offline solver on an hypothetical best scenario in which tasks realizations have the smallest cost and the smallest duration of all realizations with non-zero probability. We also tried using $h_{\max,\max}$, but could not find a better algorithm than enumerating the Pareto frontier of scenarios with non-zero probability, which is very slow. As a result, for anytime decision making, our relaxation of $h_{\max,\max}$ produces much better decisions. We tried the original B-RTDP algorithm, taking decisions with respect to lower bounds. We do not report these results here because, on most instances, the algorithm does not schedule any project within the time constraints. Taking decisions with respect to upper bounds, as in (not bounded) RTDP algorithms, is much better on this problem.

- HC-DP is the Heuristically-Confined Dynamic-Programming algorithm from [17] enhanced into an anytime algorithm. The original algorithm uses an offline learning phase, common to all the runs, to obtain a policy used during execution. The policy, computed by dynamic programming, is the solution of a restricted MDP whose state space consists of those states reached by running 3 heuristics on 50,000 scenarios each. During execution, the algorithm follows this learned policy, with some basic recourse heuristic when reaching a state outside the confined space. We use the anytime version of this algorithm proposed in [21]. When given 32 seconds per decision, this new version significantly outperforms the original algorithm on all instances but Cost5 on which both versions produce the same result.

The Offline Optimization Algorithm. *Amsaa*, *1s-AA*, and B-RTDP all use offline solver based on branch and bound algorithm for the S-RCPSP [21]. The upper bound in the algorithm relaxes the resource constraints. Its branching procedure is chronological and always schedules a task as soon as a laboratory is available. A preprocessing step removes jobs not worth scheduling and factors out costs into the rewards. This offline solver, implemented in C, solves offline problems sampled at the root node of instance Reg in less than 1 ms on average.

A Note on Modeling. The MDP used by B-RTDP and by HC-DP is *not* the MDP induced by the X-MDP used by *Amsaa*. Section 4.0.5 explained why. Indeed, when modeling the S-RCPSP as an X-MDP, it is necessary to store all the past task realizations to satisfy the Markovian property. This is not the case when modeling the problem as an MDP. As a result, the MDP used by B-RTDP and HC-DP has a smaller state space than the X-MDP used by *Amsaa*.

Comparison of the Decisions Quality

Figures 7.1 and 7.2 summarize the results for anytime decision making. It contains a table for each of the 12 instances. The first line of this table contains the empirical mean value obtained by running *Amsaa*. The three lines below report the relative gap between the expected value of the considered algorithm and *Amsaa* with the same time constraint (except for Cost5, for which the expected value is reported for all algorithms). In addition, the background color carries information about the statistical significance of the results, at the 5% level. It indicates whether the considered algorithm is better than *Amsaa*-32s (no occurrence here); not worse than *Amsaa*-32s (dark gray, e.g., *Amsaa*-500ms on Cost2); significantly worse than *Amsaa*-32s, but better than *Amsaa*-31ms (gray, e.g., *1s-AA*-31 ms on P3); worse than *Amsaa*-32s, but not than *Amsaa*-31ms (light gray, e.g., B-RTDP-2s on Agr); or worse than *Amsaa*-31ms (white, e.g., HC-DP-32s on Reg).

Overall *Amsaa* exhibits excellent performance. The solution quality of *Amsaa*-32s is often higher by at least 10% than *1s-AA*-32s, HC-DP-32s, and B-RTDP-32s, and is robust across all instances. With 32s, *Amsaa* is significantly better than all other algorithms on 11 instances and as good as any other algorithm on Cost5. Moreover, the remaining three algorithms lacks robustness with respect to the instances: They all rank last at least once. Note that, on Cost5, the optimal policy

	31 ms	125 ms	500 ms	2 s	8 s	32 s
	Instance Reg (regular)					
Amsaa	7.86E+3	8.11E+3	8.31E+3	8.41E+3	8.42E+3	8.45E+3
1aa	-5.52%	-5.15%	-7.12%	-8.07%	-7.86%	-8.28%
HC-DP	-5.09%	-8.59%	-10.26%	-11.04%	-10.97%	-11.35%
B-RTDP	-100.00%	-100.00%	-37.54%	-33.23%	-25.15%	-22.99%
	Intance Agr (agregated outcomes)					
Amsaa	1.22E+4	1.24E+4	1.26E+4	1.27E+4	1.27E+4	1.28E+4
1aa	1.14%	-0.60%	-1.45%	-2.17%	-2.49%	-2.49%
HC-DP	-4.12%	-6.29%	-7.54%	-8.23%	-8.51%	-8.56%
B-RTDP	-100.00%	-100.00%	-4.28%	-4.47%	-2.44%	-1.74%
	Instance Cost2 (costs x 2)					
Amsaa	4.34E+3	4.50E+3	4.82E+3	4.90E+3	4.85E+3	4.89E+3
1aa	-17.66%	-20.13%	-24.74%	-26.26%	-25.27%	-26.05%
HC-DP	0.41%	-1.71%	-10.02%	-9.80%	-9.20%	-10.06%
B-RTDP	-100.00%	-100.00%	-66.81%	-54.79%	-45.34%	-43.05%
	Instance Cost5 (costs x 5)					
Amsaa	-3.23E+3	-2.64E+3	-1.71E+3	-3.38E+2	6.50E+0	0.00E+0
1aa	-3.15E+3	-3.14E+3	-3.16E+3	-3.13E+3	-3.14E+3	-3.15E+3
HC-DP	-2.55E+1	0.00E+0	0.00E+0	0.00E+0	0.00E+0	0.00E+0
B-RTDP	0.00E+0	-9.42E+3	-9.10E+3	-8.49E+3	-7.86E+3	-6.94E+3
	Instance D.6 (.66 less time before revenues start decreasing)					
Amsaa	6.14E+3	6.71E+3	6.89E+3	6.89E+3	6.89E+3	6.89E+3
1aa	-11.53%	-19.62%	-22.71%	-22.34%	-22.31%	-21.43%
HC-DP	6.79%	-2.21%	-4.87%	-5.41%	-5.07%	-5.08%
B-RTDP	-77.04%	-86.00%	-78.56%	-72.46%	-62.12%	-52.53%
	Instance D1.5 (1.5 more time before revenues start decreasing)					
Amsaa	1.04E+4	1.06E+4	1.07E+4	1.07E+4	1.07E+4	1.08E+4
1aa	-12.31%	-14.29%	-14.61%	-15.11%	-15.16%	-14.87%
HC-DP	-2.54%	-3.94%	-4.22%	-4.87%	-4.89%	-4.93%
B-RTDP	-24.29%	-20.94%	-17.05%	-13.17%	-11.04%	-8.98%
	Instance P1 (no failure at the last task of each project)					
Amsaa	1.42E+4	1.46E+4	1.45E+4	1.46E+4	1.47E+4	1.47E+4
1aa	-3.61%	-5.96%	-6.26%	-6.83%	-7.27%	-7.72%
HC-DP	-11.84%	-13.93%	-13.72%	-13.99%	-14.41%	-14.80%
B-RTDP	-100.00%	-100.00%	-25.41%	-20.46%	-15.97%	-13.57%

Figure 7.1: Experimental Results for Anytime Decision Making on the S-RCPSP (1/2).

is to schedule no project. HC-DP is able to realize that quickly because it uses very fast heuristics. *Amsaa*-32s and HC-DP with at least 125 ms are also optimal on this problem.

Amsaa is also robust with respect to the available computation time. On most instances, the rankings of the four algorithms do not vary much with respect to the computation times. One might think that with very strong time constraints, *1s-AA* is preferable to *Amsaa*, because *1s-AA* can use more scenarios in the same amount of time. Yet, there are only two instances on which *1s-AA*-31ms beats *Amsaa*-31ms (Agr and P3) and 3 on which they exhibit similar results. Note that B-RTDP-31ms has a zero score on many instances due to the fact that even a single B-RTDP trial has to go deep in the state space and compute the bounds h^+ and h^- for many states. Under such strict time constraints, B-RTDP cannot even perform one trial before the deadline.

	31 ms	125 ms	500 ms	2 s	8 s	32 s
Instance P2 (no failure at the 2 last tasks of each project)						
Amsaa	1.79E+4	1.85E+4	1.87E+4	1.88E+4	1.90E+4	1.90E+4
1aa	0.54%	-1.01%	-0.49%	-0.70%	-1.64%	-1.65%
HC-DP	-9.87%	-13.07%	-13.79%	-14.43%	-15.26%	-15.22%
B-RTDP	-100.00%	-100.00%	-19.68%	-13.87%	-12.83%	-10.60%
Instance P3 (no failure at the 3 last tasks of each project)						
Amsaa	2.31E+4	2.60E+4	2.69E+4	2.69E+4	2.69E+4	2.70E+4
1aa	4.47%	1.10%	-0.28%	-0.44%	-0.49%	-0.50%
HC-DP	-0.55%	-11.63%	-14.11%	-14.22%	-14.18%	-14.29%
B-RTDP	-100.00%	-100.00%	-13.39%	-8.64%	-12.22%	-9.42%
Instance P4 (no failures)						
Amsaa	1.91E+4	2.71E+4	2.89E+4	2.90E+4	2.91E+4	2.91E+4
1aa	2.33%	1.57%	0.06%	-0.27%	-0.48%	-0.46%
HC-DP	11.35%	-21.75%	-26.64%	-26.97%	-27.24%	-27.08%
B-RTDP	-100.00%	-100.00%	-13.87%	-15.14%	-11.84%	-10.12%
Instance R.66 (revenues x0.66)						
Amsaa	3.67E+3	3.88E+3	4.03E+3	3.97E+3	4.12E+3	4.13E+3
1aa	-3.55%	-9.89%	-13.44%	-11.91%	-15.63%	-15.58%
HC-DP	-0.62%	-4.29%	-7.82%	-6.20%	-9.24%	-9.72%
B-RTDP	-100.00%	-100.00%	-50.17%	-40.49%	-38.22%	-30.97%
Instance R1.5 (revenues x1.5)						
Amsaa	1.45E+4	1.52E+4	1.54E+4	1.53E+4	1.56E+4	1.55E+4
1aa	-5.98%	-11.81%	-12.93%	-12.87%	-13.94%	-14.01%
HC-DP	-6.20%	-9.99%	-10.89%	-10.15%	-11.85%	-11.71%
B-RTDP	-100.00%	-100.00%	-31.16%	-24.84%	-22.42%	-18.75%
Instance L4-10J (twice more labs and projects than in Reg)						
Amsaa	1.41E+004	1.50E+004	1.56E+004	1.63E+004	1.65E+004	1.70E+004
1aa	2.97%	-2.33%	-7.81%	-7.55%	-7.02%	-9.16%
HC-DP	-1.01%	-4.81%	-6.05%	-10.73%	-10.66%	-13.41%
B-RTDP	-100.04%	-100.02%	-99.91%	-46.33%	-43.91%	-41.66%
Instance L3-8J (1.5x more labs and 1.6x more projects than in Reg)						
Amsaa	1.00E+4	1.04E+4	1.05E+4	1.08E+4	1.08E+4	1.08E+4
1aa	-3.85%	-5.99%	-6.01%	-8.25%	-8.83%	-9.44%
HC-DP	-11.78%	-14.21%	-14.06%	-17.32%	-16.80%	-16.52%
B-RTDP	-83.76%	-64.65%	-62.01%	-60.16%	-53.89%	-47.19%

Figure 7.2: Experimental Results for Anytime Decision Making on the S-RCPSP (2/2).

7.2.2 Complexity and Convergence of *Amsaa*

We now study the behavior of *Amsaa* experimentally.

Empirical Complexity of *Amsaa*

We first study the empirical complexity of *Amsaa*. Figure 7.3 depicts various experimental results obtained from 20 runs of *Amsaa* on instance Reg for different numbers of scenarios per decision (from 100 to 1800 by steps of 100). We focus on initial decision which is by far the most difficult and takes almost half of the time of a run. On each subfigure, the line with intervals for each data point depicts the empirical mean of the measured data, as well as a 95% confidence interval. The smooth lines depict the values predicted by fitted models. The models used have at most 2 parameters to learn so that the fit can be considered excellent when the prediction lies in the confidence interval for the 18 empirical measures.

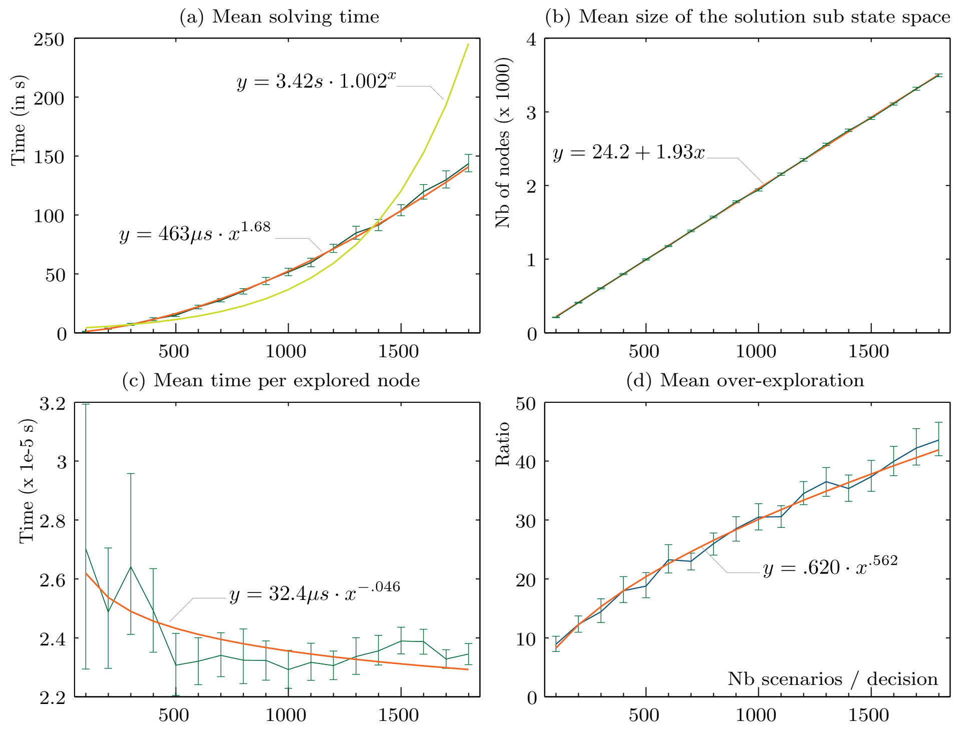


Figure 7.3: Runtime Behavior of *Amsaa* for the Initial Decisions on Reg.

Figure 7.3(a) reports the mean execution time to solve the initial SAA problem. Because *Amsaa* is exponential in the worst case, it is tempting to fit an exponential model $y = a \cdot b^n$. The best fit for such a model is shown by the light green line, which lies outside the confidence intervals of many data points: This model can be ruled out. The orange line depicts a power model of the form $y = a \cdot n^b$ which is an excellent fit and has small exponent (1.68). Therefore, on this problem, *Amsaa* is largely subquadratic in the number of scenarios.

However, one may argue that this behavior may be a consequence of iid sampling and is not a convincing evidence that *Amsaa* exhibits good performance. Indeed, in the case of a continuous distribution of the uncertainty, all the scenarios would almost surely be dispatched to different states after the first observation and *Amsaa* with iid sampling would have a linear complexity. The stochastic RCPSP has finite distributions but a similar behavior, i.e., a fast divergence of the scenarios, may explain its good performance. Obviously, *Amsaa* produces better decisions than other approaches but it is still interesting to address this issue convincingly.

On stochastic programming problems, thanks to pure exogeneity, this concern could be addressed by looking at the topology of the scenario tree (e.g., its depth and the number of nodes). There is no scenario tree for X-MDPs, but a natural generalization of this metric to X-MDPs is the number of nodes in the solution state-space. More precisely, the idea is to count, upon termination on *Amsaa*, the number of the states reachable in $\hat{\boldsymbol{A}}_n$ by following the optimal policy. This is the metric depicted in Figure 7.3(b). With a continuous distribution, the number of nodes in the solution state space

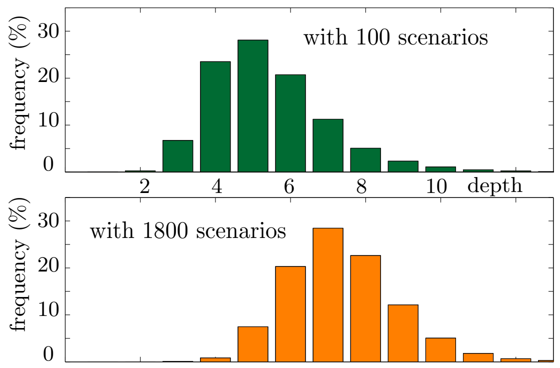


Figure 7.4: Distribution of the Depth of Explored Nodes by *Amsaa* for the Initial Decision.

would almost surely be $n + 1$ for n scenarios: the root node and n leaves. In the case of Bernoulli random variable with parameter one half, the solution state space would be a roughly balanced binary tree with $2m - 1$ nodes. These two extreme cases suggest to fit a linear model of the form $y = a + bn$. Such a model fits perfectly the experimental results with a slope of 1.93, making it much closer to a Bernoulli case than to a continuous distribution. This is an evidence that, because of the finite distributions, scenarios do not diverge too quickly with iid sampling and that the SAA problem become significantly harder with the number of scenarios.

Consider now the following decomposition of the runtime:

$$\text{runtime} = \frac{\text{runtime}}{\text{nbr. explored nodes}} \times \frac{\text{nbr. explored nodes}}{\text{nbr. nodes in solution}} \times \text{nbr. nodes in solution.}$$

Figure 7.3(c) shows how the runtime per explored node evolves. The runtime per explored nodes decreases slowly when the number of scenarios increases. This can be explained by the fact that, with more scenarios, a greater proportion of nodes are deep, making offline problems easier. This hypothesis is confirmed in Figure 7.4. It should be noted that the power model does not fit well the runtime per explored nodes. Finally, the over-exploration, that is, the ratio of the number of nodes explored by *Amsaa* over the number of nodes in the solution state space, is depicted in Figure 7.3(d). The over-exploration does grow, but quite slowly and with an exponent of 0.56.

In summary, *Amsaa* is very scalable with iid sampling, although the difficulty of the SAA problems does grow significantly.

The Importance of the Upper Bound

We now study the benefits of the upper bound $h_{\mathbb{E},\max}$ over the simpler bound h_∞ defined by $h_\infty(s) = f(s)$ is f is final, $+\infty$ otherwise. Remember that, for the induced MDP, a state s if final if $\#\mathcal{C}(s) = 1$, so h_∞ still calls the offline solver at the leaves. Heuristics h_∞ can only be used in *Amsaa* with few scenarios: the size of the state space quickly exceeds the size of the available memory. The following table report results using 10 to 50 scenarios based on 10 SAA problems.

nbr. scenarios	10	20	30	40	50
mean size with h_∞	$22{\cdot}10^3$	$89{\cdot}10^3$	$17{\cdot}10^3$	$26{\cdot}10^3$	$37{\cdot}10^3$
mean size with $h_{\mathbb{E},\max}$	64	$2.4{\cdot}10^2$	$3.9{\cdot}10^2$	$5.2{\cdot}10^2$	$6.7{\cdot}10^2$
geometric mean of ratio	$3.6{\cdot}10^2$	$4.2{\cdot}10^2$	$4.7{\cdot}10^2$	$5.2{\cdot}10^2$	$5.9{\cdot}10^2$

The results show that $h_{\mathbb{E},\max}$ is effective in limiting the size of the explored state space and that the benefits increase with the number of scenarios. In particular, *Amsaa* with $h_{\mathbb{E},\max}$ explores 360 (resp. 590) times fewer nodes than *Amsaa* with h_∞ for 10 (resp. 50) scenarios. This clearly justifies the use of the offline problems.

Other heuristics with great potential for *Amsaa* could be built from upper bounds to the offline problem. More precisely, if a function g satisfies $g(s,\xi) \geq \mathcal{O}(s,\xi)$, then the heuristic for non-final states could be $\mathbb{E}\left[g(s,\boldsymbol{\xi})|\boldsymbol{\xi} \in \mathcal{C}(s)\right]$. Such heuristics might be cheaper to compute, while retaining much of the accuracy of $h_{\mathbb{E},\max}$. Moreover, such an approach recognizes that it is easier to design good upper bounds for deterministic problems than for their stochastic versions.

Convergence of the Sampling Average Approximation

Section 6.2 described theoretical results on the SAA method for X-MDPs. It did not discuss the rate of convergence of the estimators such as $\hat{\boldsymbol{v}}_n(s_0)$, which is not surprising since few results are known even for multi-stage stochastic programs [46]. Instead we report empirical results about the convergence of the SAA estimators based on 1000 realizations of instance Reg and a number of scenarios per decision ranging from 1 to $12,000$. *Amsaa* was executed once for each pair (number of scenarios, realization). The experimental results study the convergence of the expected value, the SAA upper bound, and the selected decisions. This section is thus purely concerned with the sampling average approximation, not the way sample problems are solved.

Convergence of EV and of $\mathbb{E}\left[\hat{\boldsymbol{v}}_n(s_0)\right]$. Figure 7.5 reports the expected objective value (EV) of these runs and an estimation of the expected SAA value $\mathbb{E}\left[\hat{\boldsymbol{v}}_n(s_0)\right]$ which is an upper bound on the optimal expected value $v(s_0)$ (see Theorem 7). Measuring the expected objective values accurately is difficult because of the high variance. The figure depicts a 95% confidence interval on the EV of *Amsaa* with 12,000 scenarios per decision; the interval size is about $1,000$, more than 10% of the empirical EV. This variance is inherent to the problem, not a defect of *Amsaa*. Any other reasonable policy will exhibit a high variance. The confidence intervals are so wide that no conclusion can be drawn about the convergence of the expected objective values by comparing them. Instead, we plotted the confidence intervals of the differences between the expected values of *Amsaa* for n scenarios and *Amsaa* with 12,000 scenarios. Because the set of 1,000 realizations was the same for the different number of scenarios considered, the variances of these differences are much lower than the variances of the objective values themselves. The green area in the figure is the set of values that differ for the empirical EV of *Amsaa*-12k by a quantity within a 95% confidence interval

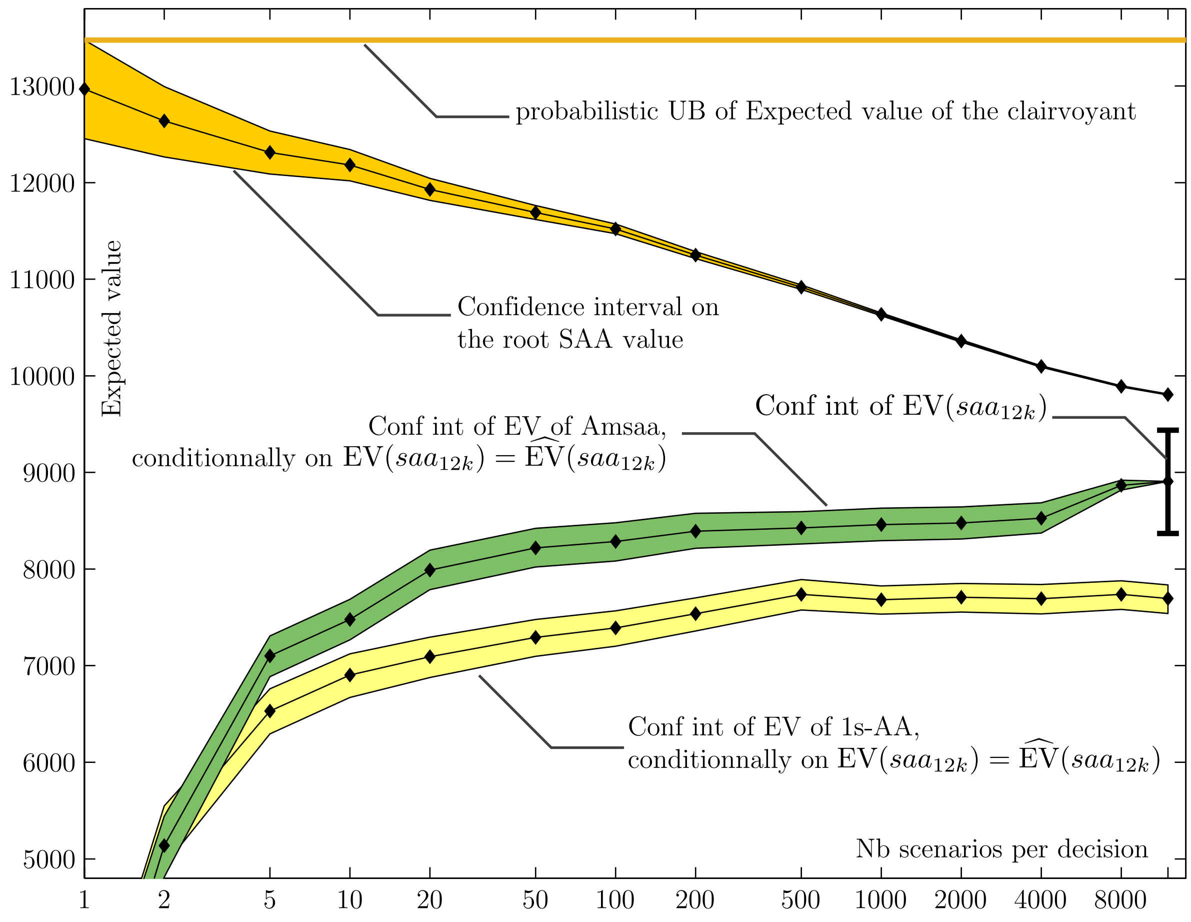


Figure 7.5: Convergence of the SAA expected value and upper bound

of the expected difference. In other terms, although one cannot claim that the expected objective value of *Amsaa* for a varying number of scenarios lie in the green area, each point will have at least 95% chance of being in the region obtained by shifting the green area vertically so that it ends at the expected value of SAA 12,000.

The optimality gap measures how close these values are from the limits. The optimality gap is not greater than the difference between the SAA upper bound (that is, $\mathbb{E}[\hat{\boldsymbol{v}}_n(s_0)]$) and the *EV*s. The orange area on Figure 7.5 represents 95% confidence intervals on $\mathbb{E}[\hat{\boldsymbol{v}}_n(s_0)]$. First observe that the $\mathbb{E}[\hat{\boldsymbol{v}}(s_0)]$ can be measured very accurately: for *Amsaa*-12k, the confidence interval is [9885..9897], less than 0.13% wide. The quantity $\mathbb{E}[\hat{\boldsymbol{v}}(s_0)]$ varies much less than the *EV*s because it concerns an agglomeration of many scenarios, while the value of a run is strongly dependent of the actual realization. Because this upper bound is obtained from the first SAA problem and because fewer runs are necessary due to the low variance, it is possible to obtain a confidence interval for this upper bound given by *Amsaa*-32,000 in reasonable time using only 50 runs: $\mathbb{E}[\hat{\boldsymbol{v}}_{32,000}(s_0)]$ lies in $[9,618..9,647]$ with probability at least 95%. In contrast, it is not computationally reasonable to execute full runs of *Amsaa*-32,000 for all the 1000 realizations. For *Amsaa*-12k, the *EV* is within

nbr. scenarios	50	100	200	500	1,000	2,000	4,000	8,000
first decision								
project B	861	975	999	1000	1000	1000	1000	1000
other	139	25	1	0	0	0	0	0
second decision								
project C	855	965	995	1000	1000	999	918	91
project D	64	21	2	0	0	0	69	856
other	81	14	3	0	0	1	13	53

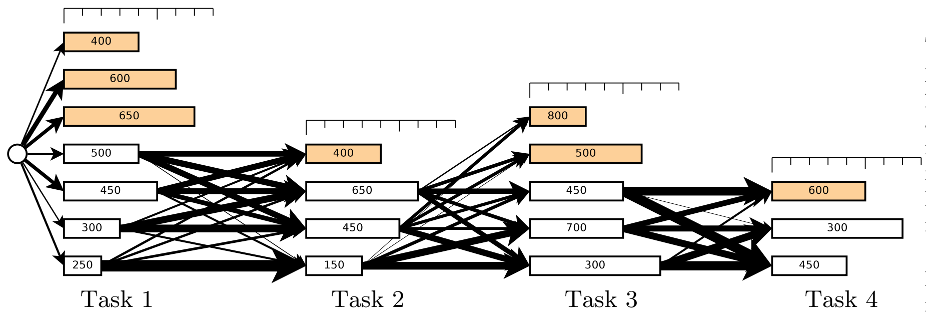
Table 7.1: The Convergence of Decisions in *Amsaa* on Instance Reg.

2% to 14% of $\mathbb{E}\left[\hat{\boldsymbol{v}}_{32,000}(s_0)\right]$ and hence within 14% of the optimal value $EV^{\star}$. This means that more scenarios are needed for the convergence of either EV, the SAA upper bound, or both. Figure 7.5 unfortunately does not rule out any of these possibilities. Yet the regularity of the graph for $\mathbb{E}\left[\hat{\boldsymbol{v}}_n(s_0)\right]$ tends to suggest that its value will continue to decrease beyond 32,000 scenarios, so the 14% optimality gap is probably pessimistic.

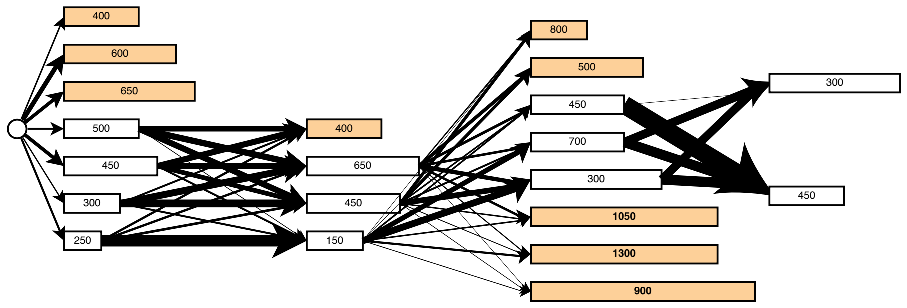
The EVs show a curious behavior investigated below: they grow only slowly from 500 to 4000 scenarios. *Amsaa*-4000 is even not better than *Amsaa*-2000 at the 5% significance level. Yet *Amsaa*-8000 is much better than *Amsaa*-4000 at the 10^{-5} significance level! This empirical study confirms that the convergence of SAA estimators for X-MDPs is much more complex than for two-stage problems. Keep in mind however that in an operational setting, what matters is anytime performance rather than the convergence.

Convergence of the Selected Decisions. We now study the convergence and stability of *Amsaa* decisions on instance Reg. We focus on the first and second decisions, which correspond to the two projects scheduled at time 0, one for each laboratory. Table 7.1 reports the number of times (out of 1,000 runs) a given project was selected in the first and second times as a function of the number of scenarios. The first decision seems to converge quickly: The same project is always selected from 500 scenarios. The second decision is more interesting: Project C is almost always chosen from 200 to 2000 scenarios. However, the decision switches to project D for 8000 scenarios, explaining why the expected value, which seemed to have converged at 4,000 scenarios, rises significantly from 4,000 to 8,000 scenarios. This odd behavior is due to the multistage nature of the problem: on two-stage problems, the estimation of each decision quickly becomes normally distributed and the convergence to the right decision is exponentially fast. On a multi-stage problem, additional sampling triggers new non-anticipativity constraints, possibly creating more complex behaviors.

Can we confirm that non-anticipatory constraints are indeed responsible for this phenomenon? Maybe so. Consider this project C, which seems attractive at first but becomes less attractive with more samples. Its structure is depicted on Fig. 7.6(a). Observe that, when task 3 succeeds with cost 450 or 700, there is a high uncertainty about the success/failure of task 4: Two arrows exiting the realization of task 3 of cost 400 have similar thickness. Project D did not show that structure, as depicted in Fig.7.7 This switch in the second decision may be explained by the non-anticipativity constraints from the realization of task 3 in this project, which might be missing with fewer scenarios.



(a) project C on Reg



(b) project C on ProjCSimpl

The thickness of the arrows is proportional to the transition probability. Failed tasks have a shaded backgroung. The numbers inside the rectangles indicate their costs, and the lengths of the rectangles indicate durations.

In ProjCSimpl, the project C never fails at task 4. New realizations are added for task 3 which, in cost and duration, are equivalent to one realization of task 3 in choiNormal followed by the failed realization of task 4.

Transition probabilities to these new realizations are the product of the corresponding transition probabilities in choiNormal between task 2 and 3 and between task 3 and 4.

Figure 7.6: The project C in Reg and in a simplified instance

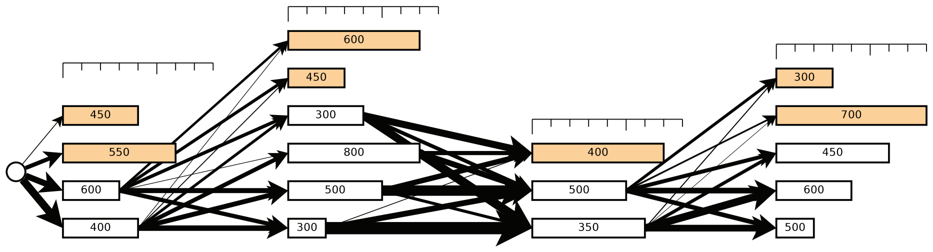


Figure 7.7: The project D in Reg

First, observe that the average depth of the leaves of the solution subtree moves from 7.2 to 8.1 when the number of scenarios increases from 4,000 to 8,000 on this instance. This depth increase can cause the SAA sample to contains scenarios indistinguishable until the observation of task 3 of project C. Second, the hypothesis was tested on a new instance ProjCSimpl, which is identical to Reg, except that project C is replaced by the one depicted in Figure 7.6(b). In this instance, task 4 of project C never fails and new realizations are added for task 3. The transition probability were corrected to make the new instance as close as possible as Reg, while allowing the failure at task 4 to be recognized one step earlier. In particular, the expected offline values are exactly the same for Reg and ProjCSimpl, as are the expected value of the optimal policy for the problems consisting of only the project C, simplified or not. If the hypothesis is correct, the second decision should also switch from project C to project D, but with fewer scenarios than on Reg. The following table show

nbr. scenarios	200	500	1,000	2,000	4,000	8,000
			first decision			
project B	1000	1000	1000	1000	1000	1000
other	0	0	0	0	0	0
			second decision			
project C	999	1000	1000	991	644	2
project D	0	0	0	6	258	943
other	1	0	0	3	98	55

Table 7.2: The Convergence of Decisions in *Amsaa* on ProjCSimpl.

the first and second decisions on 1000 runs on ProjCSimpl. The table does show a faster switch to project D. Although this experiment does not prove that the non-anticipativity constraints after observing task 3 of project C is the only reason for the switch, it does show that it plays a role in the phenomenon.

7.2.3 A Mathematical Programming Approach

Stochastic programming traditionally focuses on purely exogenous problems. However, [24] recently proposed an integer programming (IP) formulation of a Stoxuno problem (which they presented as an endogenous problem). This section evaluates a similar approach for the stochastic RCPSP using their model ($P2$). We start by describing the model in some detail and then compare the approach to *Amsaa*. The presentation of the IP is not self-contained: we will refer to notations and constraints from [24]. It can be skipped in a first reading: the comparison does not require a deep understanding of the model.

An Integer Program for the SAA problem

We directly adapted model ($P2$) from [24] and use the same notations. The decision variables in the model correspond to scheduling decisions for a task in a scenario:

- $x_{s,j,i,t}$: binary. $x_{s,j,i,t} = 1$ iff the task i of job j starts at time t in scenario s.

The model also uses some auxiliary variables to state the constraints:

- $b_{s,j,i,r,t}$: binary. $b_{s,j,i,r,t} = 1$ iff, at time t in scenario s, whether or not the realization of task i of job j is of index r is revealed. That is, until time $t-1$, both possibilities were possible and, at time t, either the task (j,i) terminates and the realization is revealed, or the task does not terminate but the time for which it has been running excludes realization r.

- $Z_t^{s,s'}$ for $s < s'$: binary. $Z_t^{s,s'} = 1$ iff scenarios s and s' are indistinguishable at time t.

The objective function is

$$\sum_s \left(\left(\sum_{\substack{j| \\ j \text{ is successful} \\ \text{in scen } s}} \sum_t \text{rwd}(j, t + \text{lastTaskDur}(s,j))\ x_{s,\text{nbTsk}(j),i,t} \right) - \left(\sum_{j,i,t} \text{cost}(s,j,i)\ x_{s,j,i,t} \right) \right),$$

where rwd(j,t) is the revenue obtained by successfully completing job j at time t, nbTsk(j) is the number of tasks in job j, lastTaskDur(s,j) is the duration of the last task of job j in scenario s, and cost(s,j,i) the cost of task (j,i) is scenario s. The problems-specific constraints (equivalent of constraints 3,4,5 in $(P2)$) are:

Start-time uniqueness constraints: Tasks can be scheduled zero or one time (it is allowed to drop a project), that is, $\forall s,j,i,\ \sum_t x_{s,j,i,t} \leq 1$;

Cumulative resource constraints: $\forall t,\quad \sum_{(s,j,i,t')\in\text{running}(t)} x_{s,j,i,t'} \quad \leq \quad$ (nbr. of labs), where running(t) is the set of quadruplets (s,j,i,t') such that if, in scenario s, task (j,i) starts at t', it runs at time t;

Precedence constraints: $\forall s,t,j,i,\ x_{s,j,i+1,t} \leq \sum_{t'\in\text{early}(s,j,i,t)} x_{s,j,i,t'}$, where early$(s,j,i,t)$ is the set of t' for which task (j,i) is completed by t if it starts at t' in scenario s;

Information acquisition constraints: These are the constraints linking the b's and the x's. Let $\Delta_{s,j,i,r}$ be the time gap between the start time of task (j,i) in scenario s and the observation of whether or not this task has realization r. $\Delta_{s,j,i,r}$ is the minimum of the duration of actual realization if task (j,i) in scenario s and of the duration of realization r of task (j,i), so it can be computed a priori. The constraints then read: $\forall s,j,i,r,t,\ b_{s,j,i,r,t} = x_{s,j,i,(t-\Delta_{s,j,i,r})}$.

The rest of the constraints are exactly the same as in $(P2)$, that is:

Non-anticipativity constraints: ((17) in $(P2)$). $\forall s,s',t,j,i,\ (Z_t^{s,s'} = 0) \Leftrightarrow (x_{s,j,i,t} = x_{s',j,i,t})$. This is easily linearized since all variables are binary. Following $(P2)$ precisely would also require adding the constraints $\forall s,s',t,j,i,\ (Z_t^{s,s'} = 0) \Leftrightarrow (b_{s,j,i,r,t} = b_{s',j,i,r,t})$. These are implied by earlier constraints in our case.

Indistinguishability constraints: ((18) in $(P2)$). $\forall s,s,',t,\ (Z_t^{s,s'} = 1) \Leftrightarrow \bigwedge_{t'\leq t,j,i}(b_{s,j,i,r,t'} = 0)$.

Performance and Comparison with *Amsaa*

The number of binary variables in the proposed IP grows quadratically in the number of scenarios, since there is a Z-variable for each time t and each pair of scenarios. For instance, the model sizes, before (and after) the CPLEX presolve, for three iid samples with respectively 5, 10, and 20 scenarios are:

nbr. scenarios	5	10	20
nbr. binary variables	$37{\cdot}10^3$ $(7{\cdot}10^3)$	$86{\cdot}10^3$ $(18{\cdot}10^3)$	$179{\cdot}10^3$ $(47{\cdot}10^3)$
nbr. constraints	$69{\cdot}10^3$ $(31{\cdot}10^3)$	$253{\cdot}10^3$ $(153{\cdot}10^3)$	$861{\cdot}10^3$ $(619{\cdot}10^3)$
nbr. matrix nonzeros	$2{\cdot}10^6$ $(1{\cdot}10^6)$	$9{\cdot}10^6$ $(5{\cdot}10^6)$	$36{\cdot}10^6$ $(20{\cdot}10^6)$

The numbers do not show quadratic growth because, for small number of scenarios, there are roughly the same number of x- and b-variables than Z-variables. Still the resulting models are of considerable size. CPLEX 10.1 did not find the optimal integer solution within 10,000 seconds, whereas *Amsaa* solves this problem in 0.1 second. On the problem with 20 scenarios, with all parameters at their default value, the presolve takes one hour, and CPLEX runs out of memory before the first integer solution is found. In contrast, *Amsaa* handles 1,000 scenarios easily. With 1,000 scenarios, the IP would have about 10^8 binary variables (since the number of time steps is about 100 and thus $(10^3)^2 \times 100 = 10^8$). Such a problem is completely unreasonable for today's IP solvers. [24] acknowledge that the IP cannot directly be solved by an IP solver and suggest a branch and bound algorithm based on a Lagrangian relaxations of the non-anticipativity constraints. Yet, with 1,000 scenarios, such a branch and bound algorithm relaxes about 10^9 constraints (there are about 10 non-anticipatory constraints for each Z) and the subgradient algorithm must optimize over a billion multipliers to solve the master problem of the Lagrangian dual, which is not reasonable.

Why is *Amsaa* much more scalable on this problem? The main difference is the way non-anticipativity constraints are handled. In Grossman's approach, these are *relaxed by Lagrangian duality* whereas, in *Amsaa*, they are enforced *lazily*. The lazy approach has two major advantages. First, the presence of Lagrangian multipliers alters the structure of the problem, precluding the use of a highly optimized ad-hoc solver as in *Amsaa*. Second, *Amsaa* exploits the discrete nature of the decisions, using states and transitions instead of discretizing time.

7.2.4 Comparison with Gap Reduction Techniques

In a very recent work, [21] proposed a variety of gap-reduction techniques to reduce the anticipatory gap of one-step anticipatory algorithms. The following table reports the relative gap (in %) between their best algorithm $\mathcal{A}_{TEPR}$ and *Amsaa*-32s. The background color provides significance information: on Cost2 and R.6, $\mathcal{A}_{TEPR}$ beats *Amsaa*-32s at the 5% significance level. On instances Reg, Cost5, and R1.5, the performance of the algorithms are not significantly different. On D.6, $\mathcal{A}_{TEPR}$ is worse than *Amsaa*-31ms. On the remaining instances, $\mathcal{A}_{TEPR}$ is worse than *Amsaa*-32s but better than *Amsaa*-31ms.

Reg	Agr	Cost2	Cost5	D.6	D1.5	P1	P2	P3	P4	R.6	R1.5
-0.24	-1.11	+9.96	0.00	-16.8	-0.43	-1.98	-2.80	-0.57	-0.62	+5.40	+0.39

Gap-reduction techniques are an attractive alternative to *Amsaa* under severe time constraints. Nevertheless, *Amsaa* outperforms them on most instances here, sometimes with a large gap (17% on D.6), and is theoretically more appealing since it converges to the optimal decisions. *Amsaa* also provides the SAA upper bound, allowing to quantify the optimality gap. On the other hand, gap reduction algorithms do not provide any better bound than the expected value of the clairvoyant ($h_{\mathbb{E},\max}(s_0)$), just like *1s-AA*.

7.3 Weapon Resource Management

After this extensive study of the S-RCPSP, we here give the results of succinct experiments obtained on the Weapon Resource Management (WRM) problem. This study is purposely short, because the WRM problem is very prospective. Hence our goal here was a quick confirmation than *Amsaa* can be useful, rather than an extensive comparison with many other techniques.

We will denote a state (t, L, U), where $0 \le t \le T$ is the time (the state is final if $t = T$, and in that case no interceptor can be sent), $L \subseteq \{e_1, \dots, e_E\}$ is the set of live missiles, and $U \subseteq \{r_1, \dots, r_R\}$ is the set of unused interceptors. A decision is a map x from U to $L \cup \{\perp\}$: if $x(r_j) = \perp$, then the interceptor r_j is not launched in the current period, and otherwise it is launched toward $x(r_j)$.

7.3.1 Decisions

One difficulty of the WRM is the number of possible decision at each step: each remaining interceptor can be send to any live missile, so there are a priori $(\#L + 1)^{\#U}$ possible decisions. To tackle this difficulty, we used dominance properties, symmetry breaking, and decomposition of decisions.

Dominance properties. Some decisions can easily be ruled out as suboptimal, and thus need not be considered. Consider a given asset a_i. Wlog, we assume that the set of missiles targeted to it is $e_1, \dots, e_k$, and that there are sorted by non-increasing probability of success: $p(e_1) \ge \dots \ge p(e_k)$. Suppose that in a given time step, the decision maker decides to send exactly one interceptor to one of the missiles threatening a_i. Then it is optimal to send it to e_1. Indeed, if the interceptor fails, then it did not matter where it was send. If it succeeds, we will reach a state that dominates the one reached by successfully intercepting a missile in $e_2, \dots, e_k$.

This reasoning can be generalized as follow:

Lemma 8 *Let x be a decision to be made in state (t, L, U). Let $a \in \{a_1, \dots, a_A\}$ be an asset threatened by two live missiles $e, e' \in L$ such that $p(e) > p(e')$. If there exists an interceptor r' such that $x(r') = e'$, but no interceptor r such that $x(r) = e$, then the decision x is strictly dominated.*

For example, if three missiles of respective success probability $p(e_1) = .6$, $p(e_2) = .4$, $p(e_3) = .3$ are all targeted to the same asset, and that three identical interceptors are available, this lemma rules out the following decisions, to name a few:

1. send one interceptor to e_2;
2. send one interceptor to e_3;
3. send one interceptor to e_1 and one to e_3.

The following decisions are NOT ruled out:

1. send one interceptor to e_1;
2. send two interceptors to e_1;
3. send one interceptor to e_1 and one to e_2.

Symmetry breaking. It seems plausible that many interceptors will be identical; but there might be a few different types of interceptors. For example, a more recent, more effective design might be available, but some older interceptors have not been taken out of service. This introduces symmetries between the assignment decisions. They can be broken by giving an arbitrary order of use to the interceptors of each type. In our experiments, there were 1 to 3 types of interceptors.

Decomposition. Despite the dominance properties and the symmetry breaking, there are still an exponential number of decisions in the number of resources. We therefore change the model to introduce the concept of *partial decisions*. Instead or representing a state by a tuple (t, L, U), we use a tuple (t, L, U, P, m). Like previously, t is the time, L the set of live missiles, and U the set of unused interceptors. Additionally, $P \cap U = \varnothing$ is the set of pending resources, that is, resources for which an assignment has been decided in the current time period, but whose success or failure have not yet been observed. m is the assignment map from P to L. With this modelling, a decision is simply the assignment of one interceptor to one missile, or the decision that we have assigned enough resources for the current period and that we want to observe their outcome and move on to the next period. Note that this decomposition of actions adds a new symmetry due to the commutativity of partial decisions; new symmetry breaking rules are thus added.

7.3.2 The offline problem

The specificity of this problem is that the offline problem is polynomial: it can be solved by dynamic programming.

Consider a state (t, L, U, P, m) and a compatible scenario. Such a scenario can be represented by the set $Y \subseteq U \cup P$ of unused and pending resources that successfully destroy their target. Note that such a representation is possible because the interception probability depends only on the resource, not on the missile that it is targeted to.

The problem can be preprocessed thanks to the following observations. First, because interceptors in P have already been assigned, the set of live enemies after they have been launched can be computed: it is $L' = L \setminus \{m(r), r \in Y \cap P\}$, and these resources don't have to be considered. So the problem is equivalent to the one in state (t, L', U). Second, note that for the offline point of view, time doesn't matter, as no observation is necessary. Therefore offline solutions are dominated by solutions in which all launched resources are launched during the current time period. Finally, from the offline point of view, success probabilities of interceptors are irrelevant, so all that matters is how many of them are successful.

Hence the preprocessed offline problem takes the following simple form. The input is:

- a set of asset $a_1, \ldots, a_A$ of respective value $v(a_1), \ldots, v(a_A)$;
- a set of missiles $e_1, \ldots, e_E$ with respective targets $\text{trg}(e_1), \ldots, \text{trg}(e_E)$ and respective success probabilities $p(e_1), \ldots, p(e_E)$;
- a number R of available successful resources and a value c for each unused interceptor.

The objective is to find the subset $I \subseteq \{e_1, \ldots, e_E\}$ of missiles to intercept. This set should be of cardinality $\#I \le R$ and should maximize

$$g(I) = -c\#I \quad + \sum_{i=1\ldots A} v(a_i) \cdot \prod_{\substack{j=1..E \\ \text{trg}(e_j)=a_i \\ e_j \notin I}} (1 - p(e_j)) .$$

It is actually possible to preprocess the problem further. Indeed, suppose that the optimal set I of missiles to intercept has k missiles threatening a given asset a. Then these k missiles are necessarily the k of greatest success probability among the ones targeted to a. Denote $e_1^a, \ldots, e_E^a(a)$ the missiles targeted to a, ordered by non-increasing success probability. Then the expected value of asset a if the k most efficient of them are intercepted is:

$$V_{a,k} = v(a) \cdot \prod_{k<j\le E(a)} \left(1 - p(e_j^a)\right) .$$

Note that for $k \ge E(a)$, the product is over an empty set, and thus $V_{a,k} = v(a)$ in that case. These values $V_{a,k}$ for all assets a and all possible k can be computed in $O(AR)$. After this second stage of preprocessing, a dynamic program can computes how best to assign the R resources. Denotes $M_{i,k}$ the maximal expected value of assets $a_1, \ldots, a_i$ protected with up to k successful interceptors. In other words,

$$M_{i,k} = \max_{\substack{I \subseteq \{e_1,\ldots,e_E\} \\ \#I \le k}} \sum_{j=1\ldots i} v(a_j) \cdot \prod_{\substack{l=1..E \\ \text{trg}(e_l)=a_j \\ e_l \notin I}} (1 - p(e_l)) .$$

All the M's can be computed thanks to the following limit conditions and recursion formulae. First, if $k = 0$, then there are no resources, so no possible choices: $M_{i,0} = \sum_{j=1\ldots i} V_{a_i,0}$. Equally, with only one asset, all resources should be used to protect it: $M_{1,k} = V_{a_1,k}$. Finally, when adding one more asset a to protect, the number of resources that should be used to protect it is between 0 and $\min(E(a), k)$:

$$M_{i+1,k} = \max_{x=0..\min(E(a),k)} \bigl(V_{a_{i+1},x} + M_{i,k-x}\bigr).$$

Thanks to this formula, the vector $M_{A,0}, \ldots, M_{A,R}$ can be computed in $O(AR^2)$-time and $O(R)$-space. It remains to choose the optimal number of resources to use globally, taking into account the value for unused interceptors:

$$\max_{\substack{I \subseteq \{e_1,\ldots,e_E\} \\ \#I \le R}} g(I) = \max_{k=0\ldots R} \bigl(M_{A,k} - ck\bigr).$$

All in all, the whole preprocessing, dynamic programming phase, and then deciding the optimal number of interceptors to use globally takes $O(AR^2)$-time. Moreover, given a state (t, L, U, P, m) and two scenarios $Y, Y' \subseteq U \cup P$, note that if $Y \cap P = Y' \cap P$, then the processing and dynamic programming phase can be shared by the two offlines: only the last phase, which is not the bottleneck, might differ. Therefore, when enumerating the scenarios compatible with a state, we can use a caching mechanism to store and reuse the M values.

To illustrate the algorithm, let's run an example. Let a, b, c be three assets of value $v(a) = 40$, $v(b) = 40$ $v(c) = 20$. There are four successful interceptors, and the value for remaining unused resource is 2. Incoming missiles are as follow:

Missile	e_1^a	e_2^a	e_3^a	e_1^b	e_2^b	e_1^c	e_2^c
Target	a	a	a	b	b	c	c
Probability	.6	.5	.25	.8	.75	.75	.2

Using the above formula for V, we compute the expected value of each asset after intercepting the k most threatening missiles:

$k =$	0	1	2	3	4
asset a	6	15	30	40	40
asset b	2	10	40	40	40
asset c	4	16	20	20	20

We can now run the dynamic program. Here we show a two-dimensional array for clarity, but only one column at a time needs to be stored.

M	asset a	assets a, b	assets a, b, c
$k = 0$	6	$6 + 2 = 8$	$8 + 4 = 12$
$k = 1$	15	$\max(2 + 15, 10 + 6) = 17$	$\max(4 + 17, 16 + 8) = 24$
$k = 2$	30	$\max(2 + 30, 10 + 15, 40 + 6) = 46$	$\max(4 + 46, 16 + 17, 20 + 8) = 50$
$k = 3$	40	$\max(2 + 40, 10 + 30, 40 + 15) = 55$	$\max(4 + 55, 16 + 46, 20 + 17) = 62$
$k = 4$	40	$\max(2 + 40, 10 + 40, 40 + 30) = 70$	$\max(4 + 70, 16 + 55, 20 + 46) = 74$

Finally, the value of unused interceptors is taken into account: the following table gives the optimal score achievable depending on the number of used interceptors:

k	0	1	2	3	4
value	$12 + 4 \times 2 = 20$	$24 + 3 \times 2 = 30$	$50 + 2 \times 2 = 54$	$62 + 1 \times 2 = 64$	$74 + 0 \times 2 = 74$

And we find that the optimum is to use all the interceptors, to intercept $I = \left\{e_1^a, e_2^a, e_1^b, e_2^b\right\}$, which has a score of $g(I) = 74$. Note how a greedy algorithm, that would assign the interceptors one by one to the missile leading to the largest increase in value, would instead intercept e_1^a, e_2^a, e_3^a and e_1^c, getting a value of $V_{a,3} + V_{b,0} + V_{c,1} = 58$, which is suboptimal.

7.3.3 Empirical Results

7.3.4 Computing Optimal Policies

While *Amsaa* was primarily designed for anytime decision making, using sampling approach, if the distribution has a small enough support it can also be used to compute optimal policy. To do so, the scenario sampler is replaced by a scenario generator that enumerates all possible realizations. There are 2^R possible scenarios, so this approach is reasonable up to about 12 interceptors. (Remember that *Amsaa* stores explicitly the list of compatible scenarios at each decision node. With 12 interceptors, the list of compatible scenarios will take 16kB, assuming pointers to scenarios take 4 bytes each).

In this experiment, we compare the computation time of *Amsaa* to to compute the optimal policy with the one from a prototype developed by David Grabiner of Lincoln Lab. We have at hand 100 instances, with 1 to 5 assets to protect, 1 to 10 incoming missiles, and 1 to 10 available identical interceptors (David's prototype does not support non-identical resources). The following table provides some quantiles of the runtime needed to compute the optimal policy by the two programs among these 100 instances.

quantile	50%	95%	98%	99%	100%
DP cpu time	30 ms	15.7 s	36.0 s	48.7 s	239 s
Amsaa cpu time	130 ms	16.4 s	32.0 s	33.2 s	43.1 s

It appears that while a pure dynamic programming approach (without upper bounds) is faster in most cases, on the 2% hardest instances, *Amsaa* is faster, and the gap widens on the very hard instances. Hence it appears that the $h_{\mathbb{E},\max}$ upper bound used by *Amsaa* provides some protection against combinatorial explosion in hard instances.

7.3.5 Comparison With *1s-AA*

The previous experiments revealed that *Amsaa* was an attractive solution to compute optimal policies, as long as the number of resources is small enough. It did not measure whether there was any value to using the optimal policy instead of a one-step approach. In this experiment, we will compare *Amsaa* to *1s-AA-E*. To take away completely the sampling error, we will evaluate *1s-AA-E* using all the possible scenarios, which is a brutal way of measuring the performance of *1s-AA* after convergence.

We ran *1s-AA-E* with this perfect scenario generation on 100 realization for each of the 100 instances. The following array reports the distribution of the optimality gap of *1s-AA-E*, compared to the optimum online policy.

quantile	5%	25%	50%	75%	95%
optimality gap	66%	25%	5.8%	0.41%	0.07%

The results yielded by *1s-AA* are on this problem very suboptimal on many instances: the median optimality gap is 5.8%, and on 5 instances out of 100 *1s-AA* returns an expected value of just a third of the optimal, which is definitely not satisfying. There is a very clear explanation of why. Remember that from an offline point of view, timing is not relevant: all that matters is what missiles are intercepted, not when. Therefore, in a one-step approach, and for all time periods but the last, there is never any incentive to send interceptors at the current time period, when their outcome is unknown, since at the next time period their outcome will be known. Hence, the one-step approach postpones as much as possible the sending of the interceptors. If the model allows to send no interceptors in a time period, this will always be chosen. The model we used enforces to send at least one interceptor per time period, and therefore the one-step algorithm never uses more than one resource for each period, except at the last one.

This is an important limitation of *1s-AA*, which reveals one case in which it cannot be used satisfyingly. In all the problems studied where *1s-AA* was very successful (e.g., stochastic vehicle routing and stochastic multiknapsack), the temporality of decisions matters offline too: in vehicle routing, because sending the vehicles too late would cause not having times to complete tours, and in SMK, because the currently considered item can be taken only right now. If the item is rejected, there will be no further opportunity to take it.

A similar situation occurred in the S-RCPSP, but to a lesser extent: the revenues were non-increasing functions of the completion time of projects, and so postponing a scheduling decision (offline) would cause a reduction in the revenues. However, this was not always the case, because the revenues functions were flat on some intervals. Interestingly, [21] reported that on the S-RCPSP, *1s-AA-E* yielded better results if leaving a lab temporarily unused while a task was ready to start was disallowed.

As a conclusion, a necessary condition for *1s-AA* to perform well is that decisions cannot be postponed in the offline problem without suffering a loss. If this is not the case, *1s-AA* tends to always wait, because of the deceptive belief that it will make more informed decisions later.

7.4 Stochastic Batch Scheduling

We now turn to the stochastic batch scheduling problem, on which want to verify the applicability of *1s-AA* and *Amsaa*, and measure whether *Amsaa* outperforms one-step approaches.

7.4.1 Two Types of Partial Decisions

At each time period, in the online model used, the number of polymerizations for each recipe and the status of the finishing lines must be decided. As these are decisions of very different nature, it is natural to decompose the decision making. There are few possible decisions for the finishing lines: at most, there are three possibilities for each line (OFF, OFF_ON, OFF_ON_OFF). Since there are two finishing lines, that's nine possible decisions.

Moreover, choosing the finishing line statuses constrains the reactions that can be started. For instance, if the corresponding line is OFF, no polymerizations can be done, because there would be nowhere to place the products of the reaction. If the line is ON, then sufficiently many polymerizations must be done to keep the mixer from running empty.

Therefore, we decompose the online decision into two partial decisions. First we choose the finishing line statuses, and then we choose the number of polymerizations for each recipe. There is no observation in between, as this decomposition is purely internal to the decision making algorithm.

7.4.2 Offline-Driven Decisions Pool Generator

On the stochastic Batch Scheduling problem, a major difficulty to use either *1s-AA* or *Amsaa* is the number of possible decisions in each state: there are 10 recipes, and up to 12 polymerizations can be started in each state. This leads to roughly $\frac{10^{12}}{10!}$ possible decisions, which is more than 10^5, for polymerizations alone. This is to be compared with 6 possible decisions in the instances of the stochastic multiknapsack instances we studied, and up to 10 decisions of the regular instance of the S-RCPSP.

In the WRM problem, we are able to mitigate the explosion of feasible decision using dominance properties and symmetry breaking. Here, there are no symmetry between polymerizations with different recipes, and no obvious dominance properties. Therefore, in order to use any algorithm that enumerates decisions, such that *1s-AA-E* and *Amsaa*, the number of decisions has to be limited in some way.

We introduce a heuristic we call the *offline-driven decision pool generator*. The idea is to first compute the optimal offline solutions for a number of scenarios without enforcing the initial decisions. Then, we derive heuristically a set of good candidates for the first decision by looking at the shape of the set of optimal initial offline decisions in the 10-dimensional space of number of polymerization per recipes.

Suppose that all initial offline decisions are very clustered in one region of this 10-dimensional space. It is then very reasonable to think that a good online decision will also lie in that region. Suppose now that the initial decisions are mostly spread out along one direction in the polymerization space. Then, while it is not clear where the optimal online decision is, it seems reasonable to try several candidate along that same direction.

The previous paragraph might suggest the use of Principal Component Analysis (PCA) to analyze the shape of the initial offline decisions set and determine the principal directions it is spread out along. In fact, we chose not to use PCA, mostly because we need to generate *integer* points: the number of polymerizations has to be an integer for each recipe.

Instead, we generate the set of candidates by feeding a simple rounding models to CPLEX, and uses the solution pool feature to generate a few good roundings. Suppose that the optimal offline initial decision for scenario i is the vector x_i. The model is, very simply, with n the number of scenarios used to generate the offline decision:

$$\begin{array}{rc} \text{Minimize} & \sum_{i=1}^{n} \|x - x_i\| \\ \text{Subject to} & x \text{ is a feasible online decision.} \end{array}$$

The norm used is the $L1$-norm, which is easy to model in an integer program. This model is very natural here, because the dimensions of the polymerizations space are very homogeneous: they all represent a number of polymerizations to do for a recipe. If the decision space had very dissimilar dimensions, for example, a component that is a natural number and a component that is a mass, then the use of $L1$ norm would be harder to advocate.

CPLEX permits to store a parameterizable number k of integer solutions while searching for the optimum. Depending on the parameters, this can be used to find either the k first solutions, the k best solutions, or a set of k diverse solutions. Generating the k-best solutions and proving they are the best is very computationally intensive. We use parameters that, while trying to generate the k-best solutions, allow a proprietary algorithm to limit space. More specifically, the parameters are:

parameter	value	meaning of the value
`SolnPoolIntensity`	2	Limit the computational overhead by a proprietary algorithm
`SolnPoolReplace`	1	When pool capacity is reached, replace worst solution if a new and better one is found
`PopulateLim`	10,000	Stop after 10,000 times a solution is added to the pool (possibly in replacement of another one)
`SolnPoolCapacity`	variable (from 1 to 8)	variable number of solutions to return

7.4.3 Offline Model

The offline problems for the batch-scheduling problem can be modeled as integer programs. However, these IPs are relatively complex to express, involving many different types of variables (the main decision variables are polymerizations variables and finishing line status variables, but there are also sales, inventory and shortage variables). While expressing such a model would be manageable using a modelling language like AMPL or OPL, these causes integration issues with the rest of the program, written in Java. On the other hand, using the Java API (Abstract Programming Interface) for Ilog CPLEX (called the Concert technology), enables tight an easy integration with the rest of the program, but the Java syntax is not well adapted to express mathematical constraints, and the resulting code is very lengthly and hard to read.

To solve this problem, we use a modelling language called OptimJ [1]. This language has the unique feature of being a superset of Java and enables the use of any Java object without any interface. OptimJ can be seen as a front-end for solver APIs that permits the use of a math-friendly syntax, 5 to 10 times more compact than the Java code calling the API. I thank Ateji, OptimJ's editor, for providing me with a complimentary license for the OptimJ compiler and for their support.

7.4.4 Scenario Trees for *Amsaa*

While on other problems we could use iid sampling, this will not be sufficient on this problem. Indeed, remember that at each period, in `highDem`, there is a probability of $1/2$ for the arrival of an order, for each product. Because order quantities are continuous, there is a zero probability than two sampled scenarios have orders with the same quantity. Therefore, from a given state s, two sampled scenarios $\boldsymbol{\xi}^1$ and $\boldsymbol{\xi}^2$ would satisfy $\tau(s, x, \boldsymbol{\xi}^1) = \tau(s, x, \boldsymbol{\xi}^2)$ only if none has an order arrival at the time period immediately those of s. This will happen with probability $({}^1\!/_{2^{10}})^2 \approx 10^{-6}$ on `highDem`.

Therefore, in order for the sample problem to have non-anticipativity constraints, non-independent sampling is required. Fortunately, because demands are exogenous, we can use standard techniques for generating *scenario trees* [22]. The simplest idea, that we use, consist of the following steps:

1. Choose a depth k and a list of degrees $d_0, \ldots, d_{k-1}$. d_i will be the degree of the scenario tree at depth i. We choose to use $k = 2, d_0 = 4, d_1 = 2$. This means there will be four sampled realizations of the order placement at the initial period, and then for each of them, two at the next period, for a total of 8 scenarios.

2. Initialize the set of current partial scenarios to contain only an empty partial scenario (i.e., that does not specify the value of any random variable);

3. Recursively, go through the list of degrees. If the next degree is d, then for each partial scenario $\boldsymbol{\xi}_{0..t}$, sample d realizations $\boldsymbol{\xi}^1_{t+1}, \ldots, \boldsymbol{\xi}^d_{t+1}$ of the next random variable to observe and in the set replace this partial scenario by d partial scenarios formed by the possible concatenations, from $\boldsymbol{\xi}_0, \ldots, \boldsymbol{\xi}_t, \boldsymbol{\xi}^1_{t+1}$ to $\boldsymbol{\xi}_0, \ldots, \boldsymbol{\xi}_t, \boldsymbol{\xi}^d_{t+1}$;

4. at the end, complete the partial scenarios into full scenarios by iid sampling.

The above algorithm is well known, and do not introduce any bias is the SAA problem.

7.4.5 Results

Algorithms are tested on seven different instances that differ a few different parameters. Instance `regular` is the main instance from which the others are derived, and is inspired by the instance provided in [44].

In `regular`, the parameter for the orders arrivals stochastic model is 0.4 — that is, at each period, and for each product, there is a probability 0.5 that an order will arrive. This order will be for a quantity drawn uniformly at random between 0 and 4. This demand correspond on average to 67% of the production capacity of the reactors. In the `highDem` instance, this probability is 0.5, while the distribution for the time between order placement and order due date and for the quantities are the same. In `lowDem`, this probability is 0.3. These demand correspond respectively to 83% and 50% of the production capacity of the reactors.

In `regular`, the cost of of startup or of a shutdown of the finishing line is \$$3k$. In contrast, this cost is \$$2k$ in `lowSwtCst`, and \$$4k$ in `highSwtCst`. This cost is to be compared with the cost of polymerizing one batch, which is \$$1k$ for all recipes in all instances.

In `regular`, the inventory cost for a 48h period is \$100 per batch, and the shortage penalty is also \$100 per batch and per 48h-period. Both these costs are \$50 in `lessJIT`, and \$200 in `moreJIT`.

This gives a total of seven instances – the main one, and variants obtained by changing the finishing line mode switch costs, the frequency of the arrival of new orders, and how important it is to complete the orders just-in-time.

The following table indicates the empirical expected value of the following algorithms:

1. *Amsaa* with 4 candidate solutions per time period and $4-2$ scenario tree;
2. *Amsaa* with 1 candidate solutions per time period and $4-2$ scenario tree;
3. *Amsaa* with 10 scenarios obtained by iid sampling, 1 candidate solution per time period;
4. *1s-AA-E*, 10 scenarios, iid sampling, 1 candidate solutions per time period;
5. *1s-AA-E*, 10 scenarios, iid sampling, 4 candidate solutions per time period;
6. *1s-AA-E*, 10 scenarios, iid sampling, 8 candidate solutions per time period;
7. a clairvoyant solver, that knows the all future before hand;
8. blind, an online solver that at each time step assumes no more orders will arrive;
9. exp-sc, an online solver that at each step uses an expected scenario as a point forecast, and makes the optimal decision for that scenario.

The results are obtained by running algorithms on 100 realizations of each instance.

	`regular`	`highDem`	`lowDem`	`lowSwtCst`	`highSwtCst`	`lessJIT`	`moreJIT`
clairvoyant	52.1	64.2	37.9	54.0	49.1	57.3	43.3
Amsaa, nr=4, tree 4-2	49.1	60.9	35.0	51.5	45.6	54.6	40.3
Amsaa, nr=1, tree 4-2	-1.10%	-0.75%	-1.64%	-0.90%	-1.19%	-1.00%	-1.01%
1s-AA-E, nr=1, iid-8	-1.21%	-0.36%	-1.14%	-1.38%	-1.25%	-1.33%	-1.87%
1s-AA-E, nr=4, iid-8	+0.02%	+0.52%	+0.09%	-0.35%	+0.43%	+0.03%	+0.72%
1s-AA-E, nr=8, iid-8	+0.24%	+0.71%	+0.38%	-0.09%	+0.28%	+0.39%	+0.99%
exp-sc	-0.91%	-1.11%	-5.16%	-1.44%	-1.58%	-1.41%	-1.87%
blind	-8.25%	-4.58%	-12.8%	-6.61%	-13.5%	-3.90%	-9.59%

The first line is the expected value of the clairvoyant. Not surprisingly, it increases when the demand is higher and when costs are lower. The second line is the expected value of *Amsaa*, nr=4, tree 4-2. Then other lines report the gap between the considered algorithms and *Amsaa*, nr=4, tree 4-2. In order to have a meaningful comparison, *1s-AA* was used with the same number of scenarios, that is, 8.

A few comments are to be made. The first is that this is actually a fairly easy problem: online values are surprisingly close to the expected value of the clairvoyant. As expected, the number of roundings considered has an influence of the solution value: the more decisions are being considered, the higher the reward.

A somewhat disappointing observation, however, is that *Amsaa* did not improve decision quality compared to *1s-AA*. On most instances, the gap between *Amsaa*, nr=4, tree 4-2 and *1s-AA*, nr=8, iid-8 is not statistically significant. However, *1s-AA* is significantly better on `highDem`, `moreJIT`, and `lessJIT`. This is a very counterintuitive result, that could be due to the heuristic used to limit the considered decisions: maybe the imperfections of this heuristic get amplified by exploration.

7.4.6 Conclusion

Compared to the three other problems considered earlier, the batch scheduling problem has a much higher number of possible decisions per state, and the offline is more difficult, taking several tens of second to solve optimally while it took less than a millisecond for the S-RCPSP. As a result, much fewer scenarios can reasonably be used, and *Amsaa*'s exploration is necessarily very shallow. This shallowness, in addition to the fact that *1s-AA* is very close to the clairvoyant (and so a fortiori to the online optimum) on these instance make that *Amsaa* does not provide an improvement in expected value.

It therefore seem that on this problem, like for others, the importance of non-anticipativity constraints is low, even though the importance of stochastic information is significant: taking into account the distribution of possible future scenarios provide significantly higher rewards than using a point forecast, and the gap is even higher compared with using a blind scenario.

Chapter 8

Search Algorithms for Amsaa

8.1 A Brief Overview of *Amsaa*

This section gives a high-level description of *Amsaa*, an Anytime MultiStep Anticipatory Algorithm. A detailed presentation can be found in [38].

Scenarios and Offline Problems Exogenous uncertainty can be modelled by a distribution of scenarios. The actual scenario is a random variable denoted $\boldsymbol{\xi}$. A *state* is the whole information available at a given time to the decision maker. This contains information about the actual scenario: in the S-RCPSP, it contains the realizations of the completed tasks, and in the OSRS, it contains the history of requests. We assume that the decision maker can sample replications of $\boldsymbol{\xi}$ conditionally on any state s. We denote by $\mathcal{C}(s)$ the set of scenarios *compatible* with a state s, i.e., those scenarios which are not inconsistent with the information in s. For a given scenario ξ compatible with the state s, there is an *offline problem* which consists of finding the best way to act from s when the realization ξ of $\boldsymbol{\xi}$ is known. Such an offline problem is deterministic and, for our applications, combinatorial: it is a cumulative scheduling problem for the S-RCPSP and a multiknapsack problem for the OSRS. We denote by $\mathcal{O}(s, \xi)$ the optimal value of this offline problem. Although these offline problems are NP-hard, we assume an effective black-box is available to compute $\mathcal{O}(s, \xi)$, which is the case for the motivating applications (branch and bound for the S-RCPSP; integer programming for the OSRS).

The MultiStep Anticipatory Algorithm The core of *Amsaa* is the Multi Step Anticipatory Algorithm `Msaa`, whose pseudo-code is as follows.

The first step consists of approximating the problem by sampling a number of scenarios. This is known as the Sampling Average Approximation (SAA) method, which was initially developed for purely exogenous problems [32] and later generalized to Stoxuno problems [38]. Sampling scenarios is an appealing approximation of a problem; it is superior to interior sampling methods for MDPs [31] since many exogenous problems exhibit positive correlations for the decision values with respect to scenarios. In other words, since there are *good* and *bad* scenarios independently of the decisions

Function `Msaa`

```
Approximate the problem: replace $\boldsymbol{\xi}$ by a random scenario $\boldsymbol{\xi}'$ whose distribution is the empirical distribution on a sample of $\boldsymbol{\xi}$;
Express the approximated problem as an MDP;
Solve the resulting MDP with a search algorithm using $h_{\mathbb{E},\max}(s) = \mathbb{E}\left[\mathcal{O}(s, \boldsymbol{\xi}') \mid \boldsymbol{\xi}' \in \mathcal{C}(s)\right]$ as an upper bound;
return the greedy decision at the root node of the MDP.
```

(scenarios with more requests in the OSRS and with more successful projects in the S-RCPSP), sampling scenarios leads to faster convergence than sampling outcomes of state-decision pairs [38].

The approximated problem is then modeled as an MDP which is solved optimally. Most real-world stochastic optimization problems cannot be solved optimally if expressed as an MDP because of the doubly exponential explosion due to (1) the outcomes and (2) the decisions. Sampling scenarios eliminates the former explosion: *for a given deterministic policy*, the number of reachable states is bounded by $T \times n$, where T is the time horizon and n is the number of sampled scenarios. The second combinatorial explosion must be tackled by effective search algorithms which can find the optimal policy by exploring only a small fraction of the state space. Fortunately, MDPs produced by `Msaa` enjoy an excellent upper bound to guide this search: for a state s, the *offline upper bound* $h_{\mathbb{E},\max}(s)$ is the expected value of a clairvoyant from state s, i.e., $h_{\mathbb{E},\max}(s) = \mathbb{E}\left[\mathcal{O}(s, \boldsymbol{\xi}') \mid \boldsymbol{\xi}' \in \mathcal{C}(s)\right]$.

Amsaa *Amsaa* is an anytime algorithm based on `Msaa`. It iteratively applies `Msaa` to increasingly finer approximations of the original problem until some termination condition is met, typically a time constraint (e.g., "make a decision within 30s"). This iterative refinement is efficient thanks to *incrementality*: Calls to `Msaa` reuse earlier computations, so that resolving the MDP is fast after a small change in the approximation.

Performance and Discussion *Amsaa* exhibits remarkable performance on the S-RCPSP. It outperforms other available algorithms on a number of instances and under various time constraints. [38] reported results with 12 instances varying the decision times from 31 ms to 32s. Each instance has 2 labs, 5 projects, and a total of 17 tasks. *Amsaa* outperformed the one-step anticipatory algorithm (1s-AA), even with gap reduction techniques [21], Bounded Real-Time Dynamic Programming (B-RTDP), a mathematical programming approach, and the Heuristically Confined Dynamic Programming (HC-DP) algorithm [17]. It is useful to spell out the strengths of *Amsaa* for Stoxuno problems.

- By using search algorithms, *Amsaa* avoids the intractability of mathematical-programming models which require a huge number of binary variables for Stoxuno problems because they cannot precompute when scenarios will become distinguishable. See [24] for such a model and [38] for a comparison with *Amsaa*.
- *Amsaa* applies the SAA method for sampling, leveraging positive correlations of the decisions contrary to interior sampling methods;

- *Amsaa* exploits a good upper bound to guide the search efficiently, leveraging the combinatorial structure of the application; For instance, on instance Reg of the S-RCPSP, the gap of the traditional $h_{\max,\max}$bound for MDPs is more than 6 times larger than $h_{\mathbb{E},\max}$'s one.

On OSRS, *Amsaa* performs as well, not better, than other state-of-the-art algorithms.

8.2 Stochastic Dynamic Programming

This section gives a brief overview of Stochastic Dynamic Programming, which aims at solving Markov Decision Processes (MDPs). MDPs can model purely endogenous problems, purely exogenous, and Stoxuno problems. We only consider finite horizon MDPs with no reward discounting and no transition costs, because MDPs generated by `Msaa` are such.

Markov Decision Processes An MDP $(S, s_0, F, X, \perp, \mathcal{X}, f, \mathcal{P})$ consists of:

- a finite state space S, an initial state $s_0 \in S$, and a set of final states $F \subseteq S$.
- a decision space X containing a decision $\perp$ (denoting no action) and a function $\mathcal{X} : S \to X$ returning the set of feasible decisions in a given state such that $\forall s \in S, \mathcal{X}(s) \neq \varnothing$ and that $\forall s \in F, \mathcal{X}(s) = \{\perp\}$.
- a bounded reward function $f : F \to \mathbb{R}$.
- a transition function $\mathcal{P} : S \times X \to \text{prob}(S)$, where $\text{prob}(S)$ is the set of probability distributions over S, satisfying $\forall s \in F, \mathcal{P}(s, \perp)(\{s\}) = 1$.

For convenience, we write $\mathcal{P}(\cdot|s, x)$ instead of $\mathcal{P}(s, x)(\cdot)$. A run of an MDP $(S, s_0, F, X, \perp, \mathcal{X}, f, \mathcal{P})$ starts in the initial state s_0. In a given state s, the decision maker selects a decision $x \in \mathcal{X}(s)$ which initiates a transition to state $s' \in S$ with probability $\mathcal{P}(s'|s, x)$. The resulting sequence of states and decisions, i.e. $s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} \dots \xrightarrow{x_{t-1}} s_t \xrightarrow{x_t} \dots$, is called a trajectory. This is a Markovian process: conditionally on s_i and x_i, the distribution of s_{i+1} is independent of the past trajectory. We assume horizon finiteness: there exists an integer T such that all trajectories starting in s_0 are such that s_T is final. As a corollary, the state space graph has to be acyclic. The objective of the decision maker is to maximize $\mathbb{E}\left[f(\boldsymbol{s_T})\right]$.

Policies, Value Functions, and Optimality A (deterministic) Markovian policy $\pi : S \to X$ is a map from states to feasible decisions, i.e., that satisfies $\forall s \in S,\ \pi(s) \in \mathcal{X}(s)$. The value $v_\pi(s)$ of policy π in state s is the expected value obtained by running policy π from state s. A policy π is optimal if the value $v_\pi(s_0)$ is maximal among all policies.

A value function v is a map $S \to \mathbb{R}$. The Q-value function canonically associated to v is the mapping $S \times X \to \mathbb{R}$ defined by $Q(s, x) = \mathbb{E}_{\mathcal{P}}\left[v(\boldsymbol{s'})|s, x\right]$, which, in the case of finite state space, becomes $Q(s, x) = \sum_{s' \in S} v(s')\mathcal{P}(s'|s, x)$. Given a value function v and a state s, a decision $x \in \mathcal{X}(s)$ is *greedy* if $Q(s, x) = \max_{x' \in \mathcal{X}(s)} Q(s, x')$. We assume that there is a rule to break ties, so we can talk about "the" greedy decision even though it is not unique. The *greedy policy* π_v associated with a value function v is the policy defined by taking the greedy decision in every state. A value function is optimal if the associated greedy policy is optimal. A necessary and sufficient condition for v to be optimal is that, for all state s reachable under π_v, we have $v(s) = f(s)$ if s is final, and $Res_v(s) = 0$ otherwise, where $Res_v(s) = v(s) - \max Q(s, x)$ is called the *Bellman residual* of v at s. Under our assumptions, there is always an optimal value function $v^\star$.

Heuristic Search Algorithms for MDPs Once *Amsaa* has solved an approximation of the problem, it solves a finer one. Hence the MDP algorithm used as a subroutine needs to terminate. We focus on heuristic search algorithms, which solve MDPs optimally and terminate. These algorithms perform a partial exploration of the state space, using a — possibly monotone — upper bound to guide the search. A value function $h : S \to \mathbb{R}$ is an *upper bound* if $\forall s \in S,\ h(s) \geq v^\star(s)$, and is a *monotone upper bound* if, in addition, $Res_h(s) \geq 0$ for all state s. A monotone upper bound is an optimistic evaluation of a state that cannot become more optimistic if a Bellman update is performed.

```
Function heuristicSearch(MDP A, h upper bound)
  precondition: $h(s) = f(s)$ if $s$ is final
  foreach $s \in S$ do  $v(s) \leftarrow h(s)$
  repeat
      Pick a state $s$
      $v(s) \leftarrow \max_{x \in \mathcal{X}(s)} Q(s, x)$
  until no state $s$ reachable from $s_0$ and $\pi_v$ has $Res_v(s) \neq 0$
  return $v$
```

Function `heuristicSearch`, captures the general schema of heuristic search algorithm for MDPs and returns an optimal value function upon termination. At each step, the algorithm selects a state and performs a Bellman update. When h is monotone, only strictly positive (instead of non-zero) Bellman residuals must be considered. Different instantiations of this generic schema differ in the choice of the state to reconsider. They include, among others, HDP [13], Learning Depth-First Search (LDFS) [14], Real-Time Dynamic Programming (RTDP) [2], Bounded RTDP [36], and AO* [25]. These algorithms only manipulate partial value functions defined only on the states visited so far, performing the initialization $v(s) \leftarrow h(s)$ on demand. The first time a Bellman update needs to be performed on a state s, its immediate successors are created and initialized. We say that s is *expanded.*

Properties of the MDPs Solved in *Amsaa* The MDPs solved in *Amsaa* have a number of specific properties: (1) their state and decision spaces are finite; (2) the time horizon is finite and the state space is thus acyclic; (3) The number of outcomes of a transition from state s and decision x tend to decrease with the depth of the state; (4) the offline upper bound $h_{\mathbb{E},\max}$is available, is monotone, and can be computed by solving an offline problem for each scenario in $\mathcal{C}(s)$.

8.3 Heuristic Search Algorithms

This section classifies the heuristic search algorithms for acyclic MDPs we are aware of. We did not include RTDP, which cannot detect convergence, nor HDP and LAO* since they reduce respectively to acyclic-LDFS and to AO* in the acyclic case. We use LDFS to denote the acyclic version of LDFS. The following table summarizes the features of the considered algorithms.

	LRTDP	LDFS	BRTDP	AO*
expansion strategy	trials	all tips	trials	one tip
maintain zero residual	no	no	no	yes
convergence testing	labels	labels	lower bounds	implicit
randomized	yes	no	yes	N/A

One of the key observations is that these algorithms often make fundamentally different design decisions, which makes it hard to determine which features are critical for a class of applications. We examine these design decisions now, starting with some notations.

- E is the sets of explored states which have been initialized ($v(s) \leftarrow h(s)$) at some time in the algorithm;
- $Z \subseteq E$ is the sets of explored states $s \in E$ such that there exists a path $s_0, s_1, \ldots, s_n, s$ such that, for all $0 \leq i \leq n$, $s_i \in E$ and $Res(s_i) = 0$. Note that $Res(s)$ can be zero or non-zero.
- $R \subseteq E$ is the set of explored states $s \in E$ that can be reached using the current greedy policy, visiting only nodes of E;
- T is the set of *tips*: it is the sets of states $s \in (R \cap Z) \setminus F$ that either have not been expanded or have a non-zero residual. These are the states which should get prime attention by an algorithm.

Expansion Strategy Most of the algorithms differ in their expansion strategy, i.e., in which states $s \in E$ are selected to be expanded. The "trials" strategy consists of simulating executions of the stochastic problem following the current greedy policy π_v, starting at a root node, and ending in a state for which $v^\star(s)$ is known. At the beginning of execution, this generally means a final node. The nodes in the trajectory of the trial are expanded whenever necessary. In this strategy, expanded states are in E but may be outside of R and/or outside of Z. The trial strategy always explores deep in the state space, even at the very beginning of the exploration. In contrast, the two other strategies ("all tips" and "one tip") are *Iterative Deepening* (ID) strategies: they only expand or update states in T. When a state is expanded, its children can be only expanded next if they are tips. This is often not the case, because the state expansion may decrease its value and remove it from R. At each iteration of LDFS, all the tips are expanded or updated: a depth-first search explores the whole frontier of $Z \cap R$. In AO*, a single tip is chosen (using any appropriate heuristic) and this tip is expanded.

Maintaining Zero Residual Once a state has been expanded, the second design issue is whether to maintain zero Bellman residuals in the explored state space. For instance, AO* restores the invariant $E = Z$ each time a state s is expanded by performing appropriate Bellman updates bottom-up. Note that AO* could instead restore the invariant $R \subseteq Z$. This would not affect the set T and AO* would behave similarly, provided that the tip-selection heuristic is not influenced by unreachable nodes. None of the other algorithms maintain such a property, so typically $R \nsubseteq Z$. In LRTDP and BRTDP, this is because these algorithms were designed for cyclic MDPs and maintaining this invariant is expensive. In the acyclic case, it is possible to design variants of LRTDP, LDFS, and BRTDP that maintains one of these two invariants. For LDFS, maintaining $R \subseteq Z$ or $E = Z$ would lead to the same behavior.

Convergence Testing Detecting termination is of primary importance for our applications. AO* terminates when T becomes empty. LDFS and LRTDP use flags to label states as "solved", meaning that it has been proved that $v(s) = v^\star(s)$. AO* could also use flags to test whether T is empty. Finally, BRTDP here is quite different: It maintains two value functions v^+ and v^- satisfying the invariant that for each state s, $v^-(s) \leq v^\star(s) \leq v^+(s)$. Hence termination is detected by checking the gap between $v^-(s_0)$ and $v^+(s_0)$.

Randomization LDFS is a deterministic algorithm: it always expands/updates all tips. AO* may or may not be randomized, depending on the tip selection heuristic, which is left unspecified. LRTDP and BRTDP are randomized. LRTDP trials are pure Monte-Carlo simulation of the MDP with the current greedy policy. BRTDP trials are different: in state s with greedy decision x, the chosen outcome is s' with a probability proportional to $\mathcal{P}(s'|s,x) \cdot (v^+(s') - v^-(s'))$. Note that, for each s', this quantity is the maximum possible decrease of $Q(s,x)$ achievable by expanding s'. This remark suggests a derandomized variant of BRTDP, in which the explored outcome maximizes this quantity.

Discussion on the Design of Algorithms The four design decisions we identified are independent and the proposed algorithms only represent a subset of the space of all possible heuristic search algorithm. In particular, many new algorithms can be constructed by combining these design choices differently. Moreover, existing algorithms often make very different decisions so that a direct comparison between these algorithms may not reveal which design decisions are significant for explaining the performance of an algorithms. In the next section, we systematically explore this design space in the hope of revealing the salient features of effective heuristic search algorithms for our domain.

8.4 Empirical Evaluation

8.4.1 Experimental Setting

Methodology and Goals Our first version of *Amsaa* used LDFS to solve MDPs. The goal of this paper is to find whether *Amsaa*'s performance can be improved by using a more sophisticated algorithm. An important aspect of *Amsaa* is the upper bound $h_{\mathbb{E},\max}(s)$ for a state s, which solves an offline problem for each scenario in $\mathcal{C}(s)$. (See [38] for experimental results indicating why this upper bound is critical). These offline problems are typically NP-complete, so it is important to solve as few of them as possible. In a first approximation, we assume that a depth-first search on the explored spates is much cheaper than initializing a state. This assumption was validated experimentally: When solving the S-RCPSP instance `3L8J` with *Amsaa* and LDFS, 90% of the cpu time is spent solving offline problems and 5% in creating nodes in the search space (which includes computing sets $\mathcal{C}(s)$). Moreover, in a modified version of LDFS which performs a useless depth-first search of the currently reachable nodes between two iterations, only 0.8% of the time is spent in this useless DFS. It is even more striking on the OSRS, for which such a useless DFS takes 0.05% of the time.

Therefore, it makes sense to look for search heuristics that minimize the number of states created and/or the number of calls to the offline solver, even if it means spending more time exploring the state space. For this reason, implementations of the search algorithms here are not optimized in general and we only optimize those promising ones for which a significant amount of time appears is spent in exploration.

We examine several design choices which we compare on one instance of OSRS (b1hh, similar to the one in [3]) and three instances of S-RCPSP (Reg, the regular instance; Agr, the instance we have on which SAA problems are the harder to solve; and 3L8J, and instance with 3 labs, 8 jobs, that has harder offline problems than Reg and Agr). For each instance, we solve 10 SAA problems with 1,000 scenarios. For each run, we measure the number of explored states, the number of calls to the offline solver, and the cpu time of the run. We use the following abbreviations: Dfs for LDFS, Bdp for BRTDP, and Ldp for LRTDP. Variants of algorithms are denoted by attaching feature at the name, in a mixed-case fashion. For example, BdpIdMzr is a variant of Bdp featuring ID and MZR.

Designing Lower Bounds There are basically two ways to design lower bounds for MDPs solved in *Amsaa*. (recall we want to maximize the expected score).

One is to use the idea from Stochastic Programming. Readers not familiar with this area can safely skip this paragraph. The scenario decomposition algorithm [42, Ch. 4,p 244] uses heuristics to transform anticipative solutions into non-anticipative ones an obtain lower bounds. Similarly, the set of offline solutions for scenarios in $\mathcal{C}(s)$ can be heuristically turned into a non-anticipative policy starting in s, providing a lower bound on $v(s)$. However, in *Amsaa*, this would probably be a bad idea. Indeed, *Amsaa* is an efficient anytime algorithm because any upper bound can be made incremental [38] but similar incrementality is not available for lower bounds. Hence, using primal-rounding heuristics to compute lower bounds is possible, but is not incremental.

The other idea is to run an online heuristic for each scenario in $\mathcal{C}(s)$. The mean score obtained by the online heuristic is a lower bound of $v^\star(s)$. Moreover this is a perfectly incremental idea: adding new scenarios will never violate any non-anticipativity constraints. Note that, in the case of randomized online heuristic, it is important to use the same seed for all scenarios in a given state, since otherwise the mean score could not be a lower bound. In the experimental results, we use the following online heuristics. For the OSRS, in a given state, assign the incoming item to the best-fit knapsack. For S-RCPSP, we compared the three following heuristics:

1. Do Nothing (DN) is a very bad heuristics that simply terminates the currently running task, if any, and schedules nothing else. DN always gets a score of zero when used from the initial state;

2. Highest Best-Case Reward (HBCR) is a reasonable heuristics: on Reg, it is 15% below *Amsaa*-32s and, on Agr, it is 9% below *Amsaa*-32s. It is inspired by the problem-independent online heuristics consisting of choosing in state s the decision x maximizing $\max_{x \in \mathcal{X}(s)} \max_{\xi \in \mathcal{C}(s)} \mathcal{O}(s, \xi)$.

3. Highest Isolated Expected Profit (HIEP) is an excellent, but expensive, online heuristics. For each available task, it solves optimally the MDP with only the project of the given task. It then chooses the task maximizing this one-project expected profit, or no task if all these expected profits are negative. This heuristic gets an expected profit that sometimes beat one-step anticipatory algorithms: It is less than 5% below *Amsaa*-32s on Reg, and less than 9% below on Agr.

Unless indicated, the lower bound used in HBCR for S-RCPSP.

8.4.2 Comparison of the Existing Algorithms

Figure 8.1 reports on the performance of the four main algorithms. Here AO* has the simplest tip selection heuristic, choosing tips according to the distribution of them being reached following the greedy policy. The results indicates that the number of states is far greater for Ldp and Bdp, suggesting that an iterative deepening strategy is superior to a "trial" exploration. The runtimes of Bdp are much larger because the lower bound is not optimized.

8.4.3 Testing Single Features

The Importance of Iterative Deepening To test the hypothesis that iterative deepening explains the performance difference, we compare Ldp and Bdp with their ID variants. The following table reports the difference in percentage of the calls to the offline solver between the ID variant, the nonId variant, and Dfs.

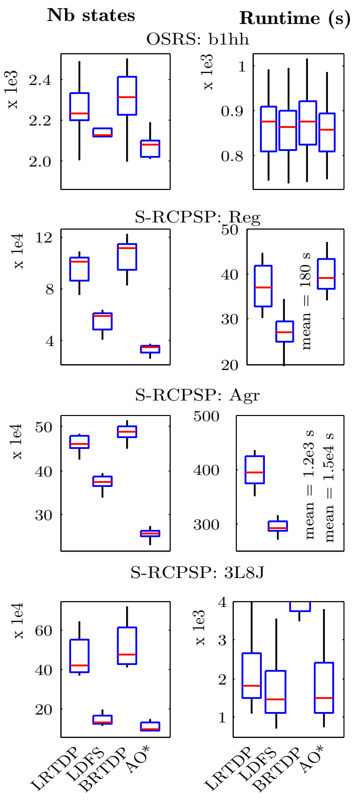


Figure 8.1: Number of States (E) (left) and Runtime (right) of the Main Algorithms

	LdpId		BdpId	
wrt.	Ldp	Dfs	Bdp	Dfs
b1hh	-3.9	-1.7	-4.4	-1.8
Reg	-35	-7.8	-36	-7.1
Agr	-25	-7.5	-27	-4.5
3L8J	-48	+3.3	-50	+2.4

These results provide conclusive evidence that an ID strategy is indeed fundamental, sometimes dividing the number of offline solver calls by 2 (on 3L8J), the number of states by 4 (Ldp, 3L8J), and reducing the cpu time by a third. Compared to Dfs, BdpId and LdpId make fewer calls to the offline solver, which suggests that a "one tip" strategy explores fewer states than an "all tips" strategy. In terms of cpu time however, Ldp is slightly dominated by Dfs, which ranges from 2% faster to 17% slower.

Maintaining Zero Residuals To evaluate the benefits of maintaining zero residuals, we compare DfsId, LdpId, and BdpId with their MZR variants. Since nodes are expensive to create and maintaining this property is cheap, we expected the MZR variants to be faster than nonMzr versions. The following table reports the variation in terms of calls to the offline solver between the MZR and nonMzr variants, and between the MZR variant and DfsMzr. All differences are in percentage.

	DfsMzr	LdpIdMzr		BdpIdMzr	
wrt.	Dfs	LdpId	DfsMzr	BdpId	DfsMzr
b1hh	-.05	-.12	-1.7	+.19	-1.5
Reg	-4.8	-1.3	-4.4	-1.4	-3.8
Agr	-7.1	-4.7	-5.1	-5.6	-2.9
3L8J	-.87	-.27	+3.9	-.32	+2.9

As expected, the number of calls to the offline solver decreases, although by a rather small (but consistent) quantity. Note that for technical reasons, we maintain $R \subseteq Z$ in LdfsMzr and $Z = E$ in LdpIdMzr and BdpIdMzr. In term of runtimes, DfsMzr is faster than Dfs by 0% to 8%. The two other algorithms have a similar runtime with and without MZR, except on Agr for which bottom-up updates take significant time and result in large slowdown. All in all, maintaining $R \subseteq Z$ is a good idea for Dfs, thanks to the consistency of gains. Maintaining $Z = E$ is not a good idea. It would be interesting to evaluate maintaining $R \subseteq Z$ on Ldp.

Testing for Convergence We observed that LdpIdMzr solves slightly fewer offline problems (up to -2.5%) but creates slightly more states (up to 4% more) than BdpIdMzr. We now try to explain this difference. BdpIdMzr uses lower bounds for two reasons: to test convergence and to guide the search by biasing the probability of selecting an outcome toward states with lage gaps between the upper and lower bounds. To separate both effects, we created a variant LdpIdMzrLb that maintains a lower bound v^- like Bdp but only uses it only for convergence testing. Hence LdpIdMzrLb and BdpIdMzr differ only by the probabilities of choosing a given outcome. We observed no statistically significant difference between LdpIdMzr and LdpIdMzrLb for both for the number of nodes and the number of calls to the offline solver. Therefore the difference in behavior between BdpIdMzr and LdpIdMzr is explained by the different outcome selection. A possible interpretation is that favoring "unknown" states tends to expand states with many compatible scenarios. We also tried to add a lower bound to test convergence to DfsMzr. It made virtually no difference, decreasing the number of offline problems by 0 to 0.3%, and increasing the runtime by up to 0.2%.

Tip Selection Heuristics for AO* Thanks to the one-tip strategy, AO* is the most reluctant algorithm to create a new state. Therefore, in the case initializing states is expensive, AO* would seem the best choice if a good tip-selecting strategy is available. We compare several tip selection heuristics for AO* or a variant of AO* maintaining two value functions v^+ and v^- like in Bdp. In addition to the two instances of AO* already presented (LdpMzrId and BdpMzrId), we developped four tip-selection heuristics, all based on the same idea: a tip is important if decreasing its v^+ can impact $v^+(s_0)$ significantly. We denote $LocalMaxDecr(s)$ the maximum decrease of $v^+(s)$ resulting from the expansion of one node. If s is a tip, this is $v^+(s) - v^-(s)$. If s is not a tip, this is the difference between $Q(s, x_1)$ and $Q(s, x_2)$ with x_1 and x_2 being the best and second-best decision in s. The three heuristics are:

- Deterministic BdpIdMzr (BdpIdMzr-D). To choose the tip, a Bdp-like trial is done, choosing at each step the outcome s' of (s, x) maximizing $\mathcal{P}(s'|s, x) \times (v^+(s') - v^-(s))$. The trial ends when a tip a reached, and this is the chosen tip.
- Globally Most Important Tip (GMIT). The chosen tip s maximizes $LocalMaxDecr(s) \times \mathcal{P}(s_0 \to s|\text{greedy})$, where $\mathcal{P}(s_0 \to s|\text{greedy})$ is the probability of reaching s in the current greedy policy;
- GMIT-Path (GMITP). For each tip s, there is a unique path $s_0, s_1, \ldots, s_n, s$ following the current greedy policy. Define $MaxDecr(s_0, s)$ as the minimum, among the states s of the path excluding s_0, of $LocalMaxDecr(s) \times \mathcal{P}(s_0 \to s|\text{greedy})$. The heuristic GMITP takes the tip maximizing $MaxDecr(s)$. The minimum of $MaxDecr(s)$ and $LocalMaxDecr(s_0)$ is the maximum possible decrease of $v^+(s_0)$ resulting from expanding s.
- GMIT-Lexicographic (GMITL). GMITP may have many ties if the limiting states are near the beginning of the paths. To address this problem, GMITL selects the tip s that is maximal for the lexicographic order over the vectors $(MaxDecr(s_0, s), MaxDecr(s_1, s), \ldots)$. In other words, the potential decrease of $v^+(s_0)$ is the most important criteria. If there are ties, the potential decrease of an immediate child of s_0 is considered, and so on for the other depths.

These heuristics are quite sophisticated, and complex data structures may be needed to find the tip to expand efficiently. In a first experiment, we took no care in optimizing this, and so we do not report cpu times. We decided to investigate efficiency only if these heuristics look promising with respect to the number of states created and calls to the offline solver.

The following table reports the mean differences on the number of calls to the offline solver when compared to BdpIdMzr. All lower bounds are intermediate.

	DfsIdMzr	LdpIdMzr	BdpIdMzrDet	GMIT	GMITP	GMITL
b1hh	+1.5	+.21	-.16	+.09	+.39	+1.2
Reg	+3.9	-.45	+83	+1.1	+22	-3.0
Agr	+2.7	-2.3	+12	+7.6	+13	+7.1
3L8J	-2.9	+.99	+198	+2.3	+57	-7.1

Most entries are positive: BdpIdMzr is, with LdpIdMzr, the heuristic that calls the offline solver the least. In particular, all the deterministic strategies are clearly dominated. Differences for the number of states created are even more favourable to randomized tip selection strategies.

Comparing the Lower Bounds We compare BdpIdMzr with these three bounds. In a first experiment, the computation of the lower bounds is not optimized: there are computed by running the above heuristics with the data structures developed for running simulations. In particular, a new state is physically created for each decision taken by these heuristics. Once again, the strategy was to evaluate the potential gain first before optimizing. The following table reports differences and ratios of quantity when using HBCR or HIEP instead of DN in BdpIdMzr. For the number of explored nodes and calls to the offline solver, it gives the mean difference (in percentage) and for the cputime, the mean ratio.

	Reg		Agr		3L8J	
	HBCR	HIEP	HBCR	HIEP	HBCR	HIEP
states(%)	-.38	-1.8	-5.3	-5.6	-1.3	-1.6
offline(%)	+0.58	+0.1	-.83	-1.1	-.83	-.98
cputime	×1.8	×7.3	×1.1	×3.0	×1.6	×5.1

As expected, the tighter the bound, the fewer states are explored. It is also somewhat true for the number of calls to the offline solver although, on Reg, there is an unexpected loss. The numbers on the call to the offline solvers show that a careful implementation of HBCR or HIEP would be a waste of time, because the potential for gain is very small: 1% at most without counting the time spent in the online heuristic. The small impact of the lower-bound quality on the performance suggest that, when solving MDPs optimally, lower bounds are not very useful to guide the search.

8.5 Conclusion

This paper investigated how different heuristic search algorithms impact the efficiency of *Amsaa*, an anytime multistep anticipatory algorithm for online stochastic combinatorial optimization. We evaluated existing algorithms, as well as novel approaches combining their orthogonal components on two applications: online project scheduling and online reservations. The main conclusion that trial-based methods are dominated by algorithms using iterative deepening. Other features have a much smaller impact, although maintaining zero residuals is a good idea with Dfs. With our current implementations, the best algorithms is DfsMzr, with Dfs and LdpId being very close. All three are within percents of each other by all measures. An efficient lower-bound implementation would bring BdpId to the same performance at best. The use of sophisticated lower bounds or tip-selection heuristics have little potential for gain, at least for solving MDPs optimally on our applications. However, lower bounds can certainly test if a given optimality tolerance is reached, which may be important in other contexts.

These small differences are actually rather reassuring: it means that there is some robustness in heuristic search algorithms. We have no example of an algorithm being the best on an instance and very bad on another. Therefore, we hope that little tuning will be needed to use *Amsaa* on new problems. Finally, although these experiments were concerned with *Amsaa* only, it is likely that our main conclusions will apply to any acyclic MDPs with expensive upper bounds.

Chapter 9

Conclusion

9.1 Summary of the Contributions

There were two goals to this doctoral work:

1. Advance the state-of-the-art in computing decisions in a limited amount of time for OSCO problems;

2. Show that online stochastic optimization can be applied to realistic problems.

I believe that these two goals have been reached to a satisfying extent. On computing decisions, we first made progresses on understanding when one-step anticipatory algorithms, that ignores the alternating character of decisions and observations. We showed that such a relaxation is satisfying when the global anticipatory gap is small. The core of this thesis, however, is to address the case where one-step algorithms are not quite satisfying enough. *Amsaa* was developed to address this case. *Amsaa* features three desirable properties:

- it is an *anytime* algorithm: its continuously refine the chosen decision until there is no time left;

- when the available runtime converges to infinity, *Amsaa*'s decision converges to the optimal one, provided that the stochastic distribution of the uncertain variables are discrete, or that scenario sampling is done in such a way that the solution sub-graph has out-degrees converging to infinity at every layers;

- it allows to derive a *probabilistic upper-bound* (when maximizing) that converges to the optimal online expected value.

Taking the non-anticipativity constraints into account was experimentally shown to lead to better decisions than ignoring them on two problems:

- the S-RCPSP, on which *Amsaa* largely outperforms all the other previously existing algorithms;

- the WRM, where a one-step approach in incapable of deciding to send several interceptors at once, which in some cases is disastrous.

Taking or not the non-anticipativity constraints into account did not seem to make a difference on the two other problems: the SKM and the BS problems.

The good experimental results for *Amsaa* on S-RCPSP and WRM can be explained by the following arguments:

- using exterior sampling makes it possible to *rank* decisions with a low variance, even though the estimations of their value might be high;
- the upper bound $h_{\mathbb{E},\max}$ considers the stochastic and the combinatorial aspects altogether, which explains why it is tight of some problems, and thus why it is well-adapted to do *guidance* and *pruning* in the state-space;
- using a black-box offline solver enables the modeler to use the best technique available — be it Mixed Integer Programming, Constraint Programming, Dynamic Programming — to solve the offline problems.

The second goal of this thesis was to show that stochastic optimization is not just a mere academic exercise. Although we have not implemented the solutions developed in this work at a client, I believe the breadth of the problems tackled, and their tight connection to real problems, makes the point. None of the problems considered has enough mathematical structure so that a specialized solution is known. The problems presented diverse challenges. The stochastic project scheduling problem featured endogenous observations, something that is known to be difficult to manage in the stochastic programming community. On weapon resource management, we had to introduce the concept of decision decomposition, and to deal with decision symmetries. The number of decisions was also a challenge on batch scheduling. There, no symmetry breaking could mitigate the problem, and we had to introduce the offline-driven decisions because the number we showed that *Amsaa* could accommodate heuristics to limit the number of considered decisions.

9.2 Research Perspectives

Along this thesis, we raised a few questions that would be interesting to investigate for the future. Concerning one-step algorithms, the techniques we used to bound the global anticipatory gap were very primitive, and advances on that matter would be theoretically interesting.

9.2.1 Applicability of *Amsaa*

Concerning *Amsaa*, a few directions for future research emerged. First, we only considered problems with discrete decision space. A generalisation to mixed discrete-continuous spaces would have a lot of value, as many problems have continuous decisions. An example of online stochastic optimization problem with continuous decision variables is the banner ad auction clearing problem introduced in [15]. There has been recent work on search algorithms for deterministic planning problems with continuous actions space [29]; maybe the same ideas can be applied to *Amsaa*.

The problems we studied also all had discrete opportunities in time were decisions could be made. In some problems, like the problems described in the section on control theory (section 3.1), it is continuously possible to adjust some parameters. There is of course a research opportunity here: can anticipatory algorithms, an *Amsaa* in particular, be modified to make decision continuously?

9.2.2 Performance of *Amsaa*

While the two above research directions are concerned with the applicability of *Amsaa*, there are also possible lines of research concerned with its efficiency. Here the most important issue is whether one can do better that sampling independent and identically distributed scenarios for Stoxuno problems. The answer is known to be yes for purely exogenous problems [22]; and it is known that iid sampling does not converge to the optimal decision is presence of continuous distributions. However, the techniques to generate scenario trees sound difficult to adapt to the case of endogenous observations.

Another potential way of improving *Amsaa*'s performance would be to hybridize it with gap-reduction techniques [21]. Indeed, we have seen how, under time constraints, *Amsaa*'s solution tree can be limited in depth and thus not take into account non-anticipativity constraints that occur after the solution trees leaves. Gap reduction techniques are heuristic ways of taking non-anticipativity constraints into account that requires negligible computation time.

Bibliography

[1] SA Ateji. The economics of optimj.

[2] Andrew G. Barto, S. J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995. rtdp.

[3] T. Benoist, E. Bourreau, Y. Caseau, and B. Rottembourg. Towards stochastic constraint programming: A study of online multi-choice knapsack with deadlines. In Toby Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 61–76. Springer, 2001.

[4] R. Bent and P. Van Hentenryck. Dynamic Vehicle Routing with Stochastic requests. *International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

[5] R. Bent and P. Van Hentenryck. Regrets Only. Online Stochastic Optimization under Time Constraints. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, San Jose, CA, July 2004.

[6] R. Bent and P. Van Hentenryck. Scenario Based Planning for Partially Dynamic Vehicle Routing Problems with Stochastic Customers. *Operations Research*, 52(6), 2004.

[7] R. Bent and P. Van Hentenryck. Online Stochastic Optimization without Distributions . In *Proceedings of the 15th International Conference on Automated Planning & Scheduling (ICAPS 2004)*, Monterey, CA, 2005.

[8] R. Bent and P. Van Hentenryck. Waiting and Relocation Strategies in Online Stochastic Vehicle Routing. *Proceedings of the 20th International Joint Conference on Artificial Intelligence, (IJCAI'07)*, January 2007.

[9] Russell Bent, Irit Katriel, and Pascal Van Hentenryck. Sub-optimality approximations. In Peter van Beek, editor, *CP*, volume 3709 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2005.

[10] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Comptutation: Numerical Methods.* Prentice Hall, 1989.

[11] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. 1.* Athena Scientific, 2005.

[12] J.R. Birge and F. Louveaux. *Introduction to Stochastic Programming.* Springer Verlag Series in Operations Research. Springer Verlag, New York, 1997.

[13] Blai Bonet and Hector Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 1233–1238. Morgan Kaufmann, 2003.

[14] Blai Bonet and Hctor Geffner. Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to mdps. In *ICAPS*, 2006.

[15] Craig Boutilier, David Parkes, Tuomas Sandholm, and William Walsh. Expressive banner ad auctions and model-based online optimization for clearing. In *Proceedings of AAAI'2008*, 2008.

[16] H. Chang, R. Givan, and E. Chong. On-line Scheduling Via Sampling. *Artificial Intelligence Planning and Scheduling (AIPS'00)*, pages 62–71, 2000.

[17] Jaein Choi, Matthew J. Realff, and Jay H. Lee. Dynamic programming in a heuristically confined state space: A stochastic resource-constrained project scheduling appplication. *Computers and Chemical Engineering*, 28(6–7), 2004.

[18] Edwin K. P. Chong, Robert L. Givan, and Hyeong Soo Chang. A framework for simulation-based network control via hindsight optimization, September 15 2000.

[19] Csirik, Johnson, Kenyon, Orlin, Shor, and Weber. On the sum-of-squares algorithm for bin packing. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2000.

[20] M. A. H. Dempster. Sequential importance sampling algorithms for dynamic stochastic programming. *Annals of Operations Research*, 84:153–184, 1998.

[21] G. Dooms and P. Van Hentenryck. Gap reduction techniques for online stochastic project scheduling. In *CPAIOR'08*, 2008.

[22] Jitka Dupacova, Giorgio Consigli, and Stein W. Wallace. Scenarios for multistage stochastic programs. *Annals of Operations Research*, 100(1 - 4):25–53, December 2000.

[23] Markus P. J. Fromherz, Lara S. Crawford, and Haitham A. Hindi. Coordinated control for highly reconfigurable systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, volume 3414 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2005.

[24] Vikas Goel and Ignacio E. Grossmann. A class of stochastic programs with decision dependent uncertainty. *Math. Program*, 108(2-3):355–394, 2006.

[25] Eric A. Hansen and Shlomo Zilberstein. LAO: A heuristic-search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1–2):35–62, 2001.

[26] P. Van Hentenryck, R. Bent, and Y. Vergados. Online stochastic reservation systems. In J. Christopher Beck and Barbara M. Smith, editors, *CPAIOR*, volume 3990 of *Lecture Notes in Computer Science*, pages 212–227. Springer, 2006.

[27] Ronald Hochreiter. Stochastic optimization in pension fund management. http://www.univie.ac.at/spxi/tutorial/TutPres_Hochreiter.pdf, 2007. Tutorial at the 11th Conference on Stochastic Programming (SPXI).

[28] P.A. Hosein and M. Athans. An asymptotic result for the multi-stage weapon-target allocation problem. *Decision and Control, 1990., Proceedings of the 29th IEEE Conference on*, pages 240–245 vol.1, Dec 1990.

[29] Samuel Ieong, Nicholas Lambert, Yoav Shoham, and Ronen I. Brafman. Near-optimal search in continuous domains. In *AAAI*, pages 1158–1163. AAAI Press, 2007.

[30] M. Kearns, Y. Mansour, and A. Ng. A Sparse Sampling Alogorithm for Near-Optimal Planning in Large Markov Decision Processes. *International Joint Conference on Artificial Intelligence (IJCAI'99)*, 1999.

[31] M. Kearns, Y. Mansour, and A. Ng. Approximate Planning in Large POMDPs via Reusable Tragectories. *Advances in Neural Information Processing Systems*, 1999.

[32] A. Kleywegt and A. Shapiro. The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM Journal on Optimization*, 15(1-2):1–30, January 1999.

[33] Jennifer L. Loveland, Susan K. Monkman, and Douglas J. Morrice. Dell uses a new production-scheduling algorithm to accommodate increased product variety. *Interfaces*, 37(3), 2007.

[34] W. K. Mak, D. P. Morton, and R. K. Wood. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24:47–56, 1999.

[35] B. Martin. Combinatorial aspects of yield management, a reinforcement learning approach. In *Proceedings of The 2004 European Simulation and Modelling Conference ESM2004*, Magdeburg, Germany, June 2004.

[36] H. Brendan McMahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In Luc De Raedt and Stefan Wrobel, editors, *ICML*, pages 569–576. ACM, 2005.

[37] L. Mercier and P. Van Hentenryck. Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In Manuela Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 07)*, volume 2, pages 1979–1984, January 2007.

[38] Luc Mercier and Pascal Van Hentenryck. Amsaa: A multistep anticipatory algorithm for online stochastic combinatorial optimization. In *Proceedings of CPAIOR*, 2008.

[39] D. Parkes and A Duong. An Ironing-Based Approach to Adaptive Online Mechanism Design in Single-Valued Domains. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 94–101, Vancouver, Canada, 2007.

[40] Andy Philpott. Stochastic optimization in electricity systems. http://www.univie.ac.at/spxi/tutorial/TutPres_Philpott.pps, 2007. Tutorial at the 11th Conference on Stochastic Programming (SPXI).

[41] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality.* Wiley Series in Probability and Statistics. Wiley-Interscience, 2007.

[42] Ruszczynski and Shapiro, editors. *Stochastic Programming.* Handbook in operations research and management science. Elsevier, 2003.

[43] A. Ruszczynski and A. Shapiro, editors. *Stochastic Programming*, volume 10 of *Hanbooks in Operations Research and Management Series.* Elsevier, 2003.

[44] G. Sand and S. Engell. Modeling and solving real-time scheduling problems by stochastic integer programming. *Computers & Chemical Engineering*, 28(6-7), 2004.

[45] Steven S. Seiden. On the online bin packing problem. *J. ACM*, 49(5):640–671, 2002.

[46] A. Shapiro. On complexity of multistage stochastic programs. *Oper. Res. Lett*, 34(1):1–8, 2006.

[47] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[48] M. Thomas and H. Szczerbicka. Evaluating Online Scheduling Techniques in Uncertain Environments. In *Proceedings of the 3rd Multidisciplinary International Scheduling Conference (MISTA'07)*, Paris, France, 2007.

[49] Jochen Till, Guido Sand, Maren Urselmann, and Sebastian Engell. A hybrid evolutionary algorithm for solving two-stage stochastic integer programs in chemical batch scheduling. *Computers & Chemical Engineering*, 31(5-6), 2007.

[50] P. Van Hentenryck and R. Bent. *Online Stochastic Combinatorial Optimization.* The MIT Press, Cambridge, Mass., 2006.

[51] Ignacio E. Grossmann Vikas Goel. A stochastic programming approach to planning of offshore gas field developments under uncertainty in reserves. *Computers & Chemical Engineering*, 28(8), 2004.

[52] Hercules Vladimirou. Stochastic programming applications in finance. http://www.univie.ac.at/spxi/tutorial/TutPres_Vladimirou.ppt, 2007. Tutorial at the 11th Conference on Stochastic Programming (SPXI).

[53] Stein W. Wallace. Stochastic optimization in logistics. http://www.univie.ac.at/spxi/tutorial/TutPres_Wallace.ppt, 2007. Tutorial at the 11th Conference on Stochastic Programming (SPXI).

[54] Emilie Winter and Philippe Baptiste. On scheduling a multifunction radar. *Aerospace Science and Technology*, 11(4):289–294, 2007.