POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Computational experiments with stochastic models for the assembly-to-order system under demand uncertainty.

Relatori
prof. Paolo Brandimarte
prof. Edoardo Fadda

Candidato
Alberto Gennaro

Anno Accademico 2019-2020

Acknowledgements

I would like to extend my sincere thanks to Professor Paolo Brandimarte, who has guided me through this project with invaluable expertise and to Professor Edoardo Fadda: we had long confrontations for every tiny detail of the models and he helped me so much in the computational aspects of this work.

I want to thank every friend I have met during my experience at Politecnico, colleagues which have stimulated me to become a better student and a better person, thank you for every laugh we had, every joke and every hour spent together.
I dedicate this work to all my friends from the tennis centre and to my coaches: without intense training hours and competitions, wins and losses I would not be the same person.
To all my lifelong friends, and in particular, to the Amitici and to the "Fai Come il Capo Bottega" group, we shared so many years together, I could not have been here without you. I cannot wait to celebrate with all of you when the pandemic will let us do.

To my family and to my girlfriend, you have inspired in so many ways, I have learned so much from you and I can simply say I love you, you have always been the reason for not giving up in every difficult moment. Thanks to my mum Luisa and to my dad Vittorio, for the unconditional love and support, thanks to my brother Carlo for every joyful moment, every fight we have had in our room together (you are hundred times wiser than me), thanks to my cousin Lorenzo for every carefree summer, thanks to my grandparents Paola, Carlo, Marika and Franco for having shown me how powerful the willingness of human beings can be and thanks to my girlfriend Federica, who more than anyone has shown me how the hard-work always pays-off. I would be nothing without you all. The best is yet to come.

Contents

List of Figures

List of Tables

1 Introduction

Uncertainty is pervasive in the world where we live and it becomes essential to take it into account properly in business decision. Among different approaches, stochastic programming is one of the most used. In this work, the focus is on an assembly-to-order system, a business situation where the firm does not store final products, but only the components necessary to make these products. Then, when an order arrives, components are assembled together and the product is sold to the client. It is obvious that such system may be a smart solution whenever the firm has some difficulties in producing or ordering the components, so that this first step is slow and need to be done before orders arrive, but it is extremely fast to assembly components and to deliver the final product; an example of such case may be the automotive sector.

In this business problem, there can be a lot of risk factors, like for example demand uncertainty for final products, machines delays or breakdowns, quality control on the production, assembly problems or changeover costs and lead times in production. This work assumes a fairly simple setting, where we ignore all the aforementioned risk factors but the demand uncertainty. Basically, we want to optimize our production decisions without knowing the demand. The classic approach to tackle this problem is to build a stochastic program with two different types of decision:

1. the *first stage* decisions, which are in our case the production plan for the components and must be chosen before demand realization;

2. the *second stage* decisions, which are sales (or assembly) decisions and can be taken while discovering the demand for the products. These decisions are conditioned on the first stage ones because we have limited components to assembly final products.

Production has a cost, which we assume to be linearly dependent on the amount of components made, and has a limit, since we need to allocate a finite number of resources. We further assume that unsatisfied demand is lost (no backlogging) and we have limits on sales based on the demand realization and the amount of components available. Our aim is to maximize profit.

We build several models to formalize the problem, both considering a single period and multi-period sales (demand as a random variable or stochastic process), trying to understand which performs better in terms of profits generated and return on investment, a popular measure for firm performance. In particular, within the same demand framework, we have two families of models: classical recourse models and decision rules ones. In the former, second stage decisions are an implicitly defined non-linear function with respect to the risk factors involved, while, in the latter, decisions are a parametric function with respect to the same risk factors (in our case we explore linear and piece-wise linear functions). So, the difference among the models is in the second stage decisions structure, while the production plan, which is the implemented part of the solution, has the same

structure. Their common characteristic is the fact they are built on montecarlo approximation of the expected value presented before, which is a common procedure in the literature of the sample average approximation technique. Our primary interest is the comparison of such models in different condition coming from possibly different economic situations. In particular, we analyse the impact of

- the *demand uncertainty*, so we try to optimize under different demand distributions, with differences in mean, volatility, skewness and finiteness;
- the number of common and specific components. It can be logic to expect that the more common components, the easier is to manage the components' production because of the risk-pooling effect on the variance of the demand for end products;
- the profitability of end products, i.e the margin over the cost of making an end-item;
- the capacity limit.

We are interested in gaining some insights about the quality of the solutions under different settings: the questions we want to answer are, for example, if there is a single model that, no matter how the aforementioned features are chosen, is always better than the other, or if there are some settings in which some or all the models have strange behaviours or lead to very bad outcomes. The work also wants to stress the importance of tractability of the models, so we put a focus on computational issues when models have heavy requirements with respect to both hardware and software. This will be tremendously important in the multi-period sales framework.

The technological support of a commercial solver was needed and we adopt the GUROBI 9.0.3 solver for every experiment carried out; all results displayed are obtained using a MacBook Pro 2018 with 2,3 GHz Intel Core i5 quad-core and 8 GB of RAM.

The work has been organized as follow: after the introduction (Chapter 1), there is an overview (Chapter 2) of the business problem, the related literature and the optimization machinery used to build meaningful models to tackle the problem.
Subsequently, in Chapter 3, the first simple models are described in details: these are the models where uncertainty does not span over different time instants but it is concentrated in a single moment. Precisely, we build four models, one where uncertainty is not treated, the classical two-stage approach and two decision rules models. We then exploit the results of the models described before, defining some measures to evaluate them, putting some lights on their behaviour and stability when changing the principal features of the problem.
After that, in Chapter 5, the robust counterparts of the previous models are presented and compared, gaining an insight into the price we pay in mean performance to build more stable solutions. In Chapter 6 the attention shifts to a new

framework, where uncertainty is modelled as a stochastic process. New models are developed, which are the extension of models we have built in the third chapter, with a new important state variable, the inventory: now we can keep pieces and assembly them in a future period, where demand is high. This machinery requires a greater computational burden: these are the multi-stage stochastic programs which are known for their issues of tractability (the curse of dimensionality) and so we show how, with decision rules models, we can avoid intractability. Specifically, we show that computational requirements for the decision rules approach scale quadratically with respect to the number of time periods, while for classical models the scaling is exponential. These models are evaluated both from a managerial point of view and in computational terms.
Finally, before the conclusions, there is an attempt (Chapter 8) to put all the models presented before under the same light, exploring some tricks coming from the dynamic programming literature to avoid computational burdens without losing too much performance. Results seem to suggest that these approaches worth the effort, since they lead to sensible solutions.

2 Literature Review and code structure

2.1 ATO

The assembly to order (ATO) is a very well known business problem, from both an academic point of view, but also from an industry perspective. An assembly-to-order system consists of multiple components, which in turn can be thought as end products of a precedent assembly, and end products to be sold to clients which are made by a suitable mix of the components. The idea behind this kind of systems is quite simple: in certain sectors, as for example car manufacturing or high-tech, the supply chain has its bottleneck in the making or ordering of the components, whereas the assembly part can be carried out in a very efficient way, so that the company can maintain low the response time to clients' orders. We will refer to the components used by all the end products as the *common components*, while the term *specific components* will identify the components used by only one end product. In the literature, sometimes the common adjective is used with a less harsh meaning, so that a common component may be part of at least a certain percentage of the final items. Common components play a significant role in the risk management perspective: if demand has a really high variability for different products, this may not be the case for the common components, which are shared across all the final products. So, in some sense they allow pooling the demands of various final products, leading to a reduction in the risks and therefore in the cost of offering a big amount of different products to the clients. Remaining in the tech sector, one can think of an ATO system for a producer of computers: there are a certain number of hardware components which can be customized (graphics card, computer data storage, keyboard) and other which are common (like the motherboard) and it is natural to store more motherboard, which is necessary to build all the computers, with respect to a specific keyboard, which may be used only by graphics experts. In the ATO problem there can be multiple sources of uncertainty, like for example:

- demand uncertainty, which is often the most studied one;
- lead time uncertainty both for components and final products;
- reliability of machines used to make the components or to assembly the final products.

Moreover, in the literature, depending on the sources of uncertainty used, there may be solutions based on optimising the (expected) profit or risk measures based on profit, but also based on client satisfaction, meaning that the objective becomes to be able to meet the client order in the smallest time possible. From this brief inspection, it is clear that a comprehensive model to tackle the problem will likely result computationally infeasible, so our choice is to study the problem taking into account only the demand uncertainty. In particular, the scope of the study will be the investigation of the robustness of our solution with respect to certain factors characterizing the instance of the problem, like:

- the skewness of the chosen distribution and demands' distributions;
- the number of common components with respect to the overall number of components;
- the percentage of items with high margin.

A quite comprehensive literature review can be found in Atan et al. (2017).

2.2 Optimization methods

Uncertainty is pervasive in our modern society and therefore, in order to survive in the increasingly competitive world of business, it is necessary to make consistently good decisions without perfect information. This is for sure one of the reason why the last few decades were devoted to developing new powerful frameworks to take uncertainty into account in the decision making process. In this context, two very popular settings were dynamic programming (DP) and multi-stage stochastic programming (MSP). We will briefly discuss these two approaches, pointing at their advantages and pitfalls, which will justify the decision rule approach that will be used in this work.

2.2.1 Dynamic Programming

Dynamic Programming is a very powerful and general paradigm based on a simple dependency among different stage of information. It is possible to cast in these settings a lot of problems, both with finite and infinite horizons. In this brief presentation, we will concentrate in a setting where the objective is to maximize the expected value of the sum of some discounted rewards in time, with a finite horizon T, but we stress that this is not meant to be a complete description of all possible problems that can be tackled by DP. Interested readers may consult Denardo (2013) or Bertsekas (2017) for a comprehensive treatment of the subject.

In the naive DP, there is a *markovian* flow of information (in the computer science field processes with this feature are known as *Markov Decision Processes* (MDPs)), meaning that the current state is the only thing that matters in order to make a decision for the next period.
Actions to perform are chosen in order to maximize the (expected value of the) objective function (also known as reward function), which depends on both action and current state. Based on current state and action chosen, there is a state transition and the agent receives its reward, as highlighted in Fig. 2.2.1. The very strong assumption of Markov behaviour leads to a nested principle known as the Bellman equation which in turn allows finding an optimal policy (with respect to the cumulative reward function) that maps the current state of the world to an optimal action to execute. Basically, we are solving the following optimization problem:

$$\arg\max_{\pi\in\Pi} \mathbb{E}\Big[\sum_{t=0}^{T-1} \gamma^t f(s_t, A_t^\pi(s_t)) + \gamma^T f(s_T)\Big] \tag{1}$$

Stated as such the problem seems to be unfeasible, but here markovianity assumption comes into play. In order to be able to apply this paradigm, it is necessary to have a value function V_t for every time instant t which maps every state s_t in a real value that can be thought as the monetary value of the state itself. Moreover, we suppose that $V_T(s_T) = f(s_T)$ is known.
Now, we can recourse backward in time, stating that

$$\begin{aligned} V_{T-1}(s_{T-1}) = &\quad \max_{x_{T-1} \in \mathcal{X}_{T-1}(s_{T-1})} \{ f_{T-1}(s_{T-1}, x_{T-1} + \\ + &\quad \gamma \mathbb{E}\Big[V_T(g_T(s_{T-1}, x_{T-1}, \xi_T) | s_{T-1}, x_{T-1} \Big] \} \end{aligned} \tag{2}$$

where the function g_T is responsible for the change of the state, possibly random given the presence of risk factors ξ_T, while $\mathcal{X}_{T-1}(s_{T-1})$ is the set of feasible actions to perform given the current state.

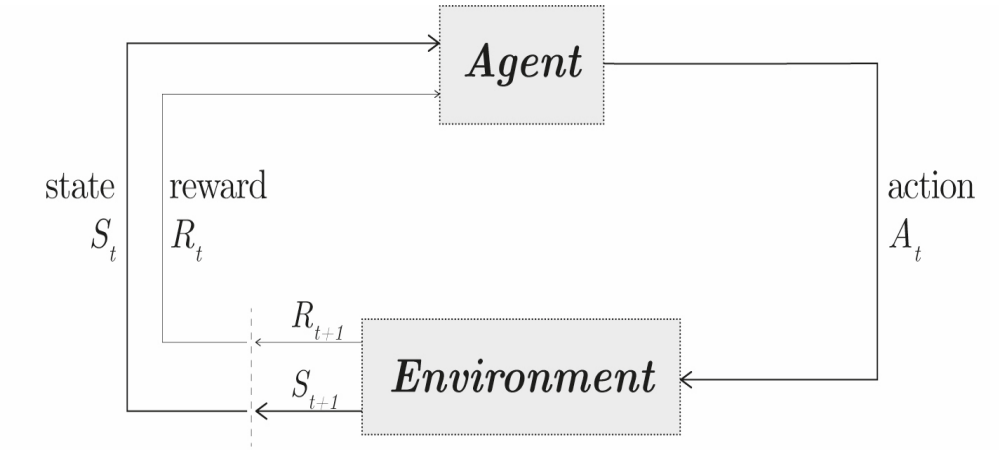

Figure 1: DP scheme: an agent interacts by means of an action with the environment which in turn provides a reward to the agent and changes its state.

The Markov property allows not to have concerns regarding how the state s_{T-1} has been reached, giving the chance to solve a simpler one stage problem. Recurring back up to the time $t = 0$ we obtain

$$V_0(s_0) = \max_{x_0 \in \mathcal{X}_0(s_0)} \{ f_0(s_0, x_0 + \gamma \mathbb{E}\Big[V_1(g_1(s_0, x_0, \xi_1) | s_0, x_0 \Big] \} \tag{3}$$

Once we have solved this last problem, we have implicitly found the best decisions x_t depending on the state s_t, i.e we have found the best policy π. Unfortunately, DP is plagued by the curse of dimensionality because we need to evaluate the value function at each time in each possible state. We have assumed that state space is discrete, but this is not often the case (and this can lead to solutions such as approximate dynamic programming or interpolation methods) and also

if discrete, the problem becomes computationally unfeasible if the dimension of the state space becomes larger and larger, as we need to evaluate each possible state-action combination in time and the number of possible combinations grows exponentially, posing problem not only in the evaluation but also in the saving of all of them. So we can conclude that, despite it allows for implicit but optimal policies, in its naive setting, DP is not really suitable for real world (and size) applications.

2.2.2 Multistage Stochastic Programming

Multistage Stochastic Programming offers other advantages and has some other pitfalls with respect to DP. First of all, it does not assume any specific characteristic about the way uncertainty is revealed through time, but only a non-anticipativity requirement must be satisfied (which is very reasonable), giving a very elastic setting to work with. Following closely the notation adopted in Georghiou et al. (2019), we can cast a MSP problem as:

$$\min_{x_1,\dots,x_T} \quad \mathbb{E}_\xi\big[\sum_{t=1}^{T} c_t(\xi_t)\cdot x_t(\xi_t)\big] \tag{4}$$

$$\text{s.t.} \quad \sum_{s=1}^{t} A_{ts}x_s(\xi^s) \geq b_t(\xi_t) \qquad \forall \xi \in \Xi, t = 1,\dots,T \tag{5}$$

$$x_t(\xi^t) \geq 0 \qquad \forall \xi \in \Xi, t = 1,\dots,T \tag{6}$$

where it is important to explicitly state the difference between ξ_t which is a realization of the risk factors at time t, and $\xi^t = (\xi_1,\dots,\xi_t)$ which is a realization of the risk factors from the beginning up to time t.
It is noteworthy to mention that non-anticipative constraints are implicitly written because of the dependency of x_t only from ξ^t and not from $\xi_{t+1},\dots,\xi_T$. In the notation, we can also see a ξ, which we will refer as a scenario (or trajectory) and it has the same meaning of ξ^T.
The problem stated in this way accommodates an infinite number of constraints, as eq. (5) and (6) must be satisfied for each possible scenario ξ in the set of uncertainty Ξ, so it is generally intractable.

To overcome this problem, the Sample Average Approximation (SAA) is often used: some techniques are adopted to generate scenarios ξ from the set Ξ and non-anticipativity constraints must be enforced for decisions which are indistinguishable at the moment they are taken. The problem becomes computationally feasible, but with some care: in fact, the more scenarios one can generate, the better the solution, but it has been shown that the complexity of the problem scale exponentially with the number of stages with respect to generated scenarios. This is the main drawback of this method. It is critical to understand the trade-off between solution accuracy and computational requirements. Small size sample solutions are strongly dependent on the sample itself and therefore perform poorly out of sample, while a big size sample leads to intractability.

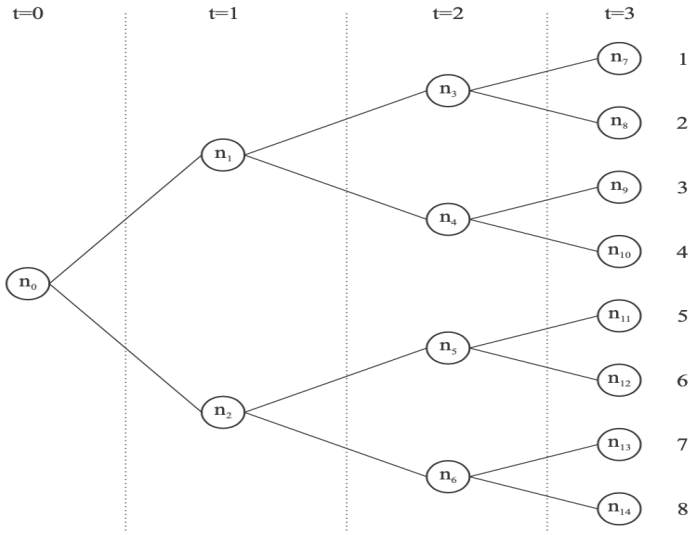

Figure 2: Example of a scenario tree: 2 realization of the risk factor and 4-time intervals leads to 2^{4-1} possible scenarios.

2.2.3 Linear Decision rules

Improvements can be made from the naive description of these two frameworks (from approximate dynamic programming to smart scenario generation and variance reduction techniques), or a third road may be taken: it is exactly in the middle of the two aforementioned approaches that decision rules find their place. The idea behind decision rules is to look for the best policy possible among a parametrized subspace of the infinite space of all real-valued functions. In order to better understand what that means, it is worthy to set the environment in which we are going to deploy these decision rules. In their description, we will again follow closely the work of Georghiou et al. (2019).
The general problem we want to solve is

$$\min_{x_1,\dots,x_T} \quad \mathbb{E}_\xi\big[\sum_{t=1}^{T} c_t(\xi_t)\cdot x_t(\xi_t)\big] \tag{7}$$

$$\text{s.t.} \quad \mathbb{E}_\xi\big[\sum_{s=1}^{T} A_{ts}x_s(\xi^s)|\xi^t\big] \geq b_t(\xi_t) \qquad \forall \xi\in\Xi, t=1,\dots,T \tag{8}$$

$$x_t(\xi^t)\geq 0 \qquad \forall \xi\in\Xi, t=1,\dots,T. \tag{9}$$

This problem is a generalization of Problem 4, which can be reconstructed by setting $A_{ts}=0 \quad \forall t<s$: in this way we meet non-anticipativity constraints and, since now we only deal with no-future variables, conditional means with respect to ξ^t

are treated as constants, so we recover Problem 4.
First of all, without any loss of generality, we can always assume that only constraints are subject to uncertainty, possibly using some additional variables to write the problem in its "epigraph" form, therefore we can suppose that cost coefficients c_t are non random. Now, it is clear from this problem that we are trying to optimize some functions $x_t(\cdot), \quad \forall t = 1, \dots, T$ under some constraints, but we are not requiring any form of regularity about these functions: the space of all possible functions from an infinite set to another one is non-countable and this means the problem is intractable. But what if we impose that $x_t(\xi^t) = X_t\xi^t$ with X_t matrix? Then, we are basically shifting our problem to a stochastic linear program, looking for the best matrix coefficient (which are finite in number) under some constraints. So, our focus is now on the linearized version with non-anticipativity constraints (no conditional expectation), that we can write as

$$\min_{X_1,\dots,X_T} \quad \mathbb{E}_\xi\big[\sum_{t=1}^{T} c_t X_t \xi^t\big] \tag{10}$$

$$\text{s.t.} \quad \sum_{s=1}^{T} A_{ts} X_s \xi^s \geq b_t(\xi_t) \qquad \forall \xi \in \Xi, t = 1, \dots, T \tag{11}$$

$$X_t \xi^t \geq 0 \qquad \forall \xi \in \Xi, t = 1, \dots, T. \tag{12}$$

Now, under suitable condition on the set Ξ (basically Ξ has to be a polyhedron that can be separable) it is possible to show (see Georghiou et al. (2014) or Georghiou et al. (2019)) that the problem, which accommodates an infinite number of constraints, can be reduced to a finite linear programming problem (using concept of duality). This is the most technical approach and solve exactly the problem, but requires the knowledge of the space in which the risk factors lie. Another possible approach, the one we will use in our work, simply operates a sample average approximation to the approximated linear program generated by the linearization of the decision: the advantage of this approach is due to the fact that rules are scenario independent and so there is no need in the construction of a scenario tree for the non anticipativity requirements. This allows representing uncertainty well without an explosion in the number of scenarios, so that the trade-off between the accuracy of the solution and computational burden may be the one desired. The drawback is obviously a loss of optimality, since this is an approximation and it is quite common to have optimal policies which are non-linear. Nevertheless, linear policies are simple to implement in real-life applications (and are generally more interpretable) and "locally" approximately correct (if we think of the linearized version like a Taylor expansion truncated at the first order). This was a simple introduction on the topic, where the focus was on the intuitive idea and the mathematical framework has been kept simple. For the interested reader, a more comprehensive treatment can be found in Ben-Tal et al. (2004), which is the seminal work for the renaissance of linear decision rules in the XXI century. In Chen et al. (2007) new deviation measure (forward and backward) are defined to

overcome some difficulties with infinite domains for the uncertainties, to set probabilistic constraints which are tighter than usual. Decision rules can be extended to obtain greater model's flexibility, so in the literature, some advances have been done from Linear Decision Rules (LDRs). In particular, in Chen et al. (2007) and Georghiou et al. (2014) some piece-wise linear decision rules are treated and in Georghiou et al. (2019) some non-linear decision rules are studied by means of a lifting operator, an operator that takes the uncertainty factor from a space to another, preserving essentially the convexity of the set (or finding a tight inner approximation for the set Ξ). A great work in assessing the sub-optimality of this approach can be found in Georghiou et al. (2014) where a posteriori bounds on the optimality gap can be found by solving the primal approximate linear program and its dual. Finally, this framework may be used in different contexts: in Ben-Tal et al. (2004) an inventory management problem is discussed, in Georghiou et al. (2014) a production planning problem is tackled, but also in portfolio management, we can see applications of decision rules, as for example in Calafiore (2008) or in Moallemi and Sağlam (2017).

2.3 General code architecture

In the following sections of this work, we propose different models with which to tackle the ATO problem and we have stated them in mathematical terms.
To use these models and solve the problem we needed an optimization toolbox and we found it in the `Python` version of the `Gurobi` optimizer. The choice is related both to the great performances a commercial solver can achieve and to the fact that model generation and maintenance is made really easy in the `gurobipy` (Gurobi toolbox for Python) framework. We have built an Object-Oriented architecture to handle model creation, instance generation and evaluation, which will be available on GitHub (upon request): We now discuss briefly all the folders and

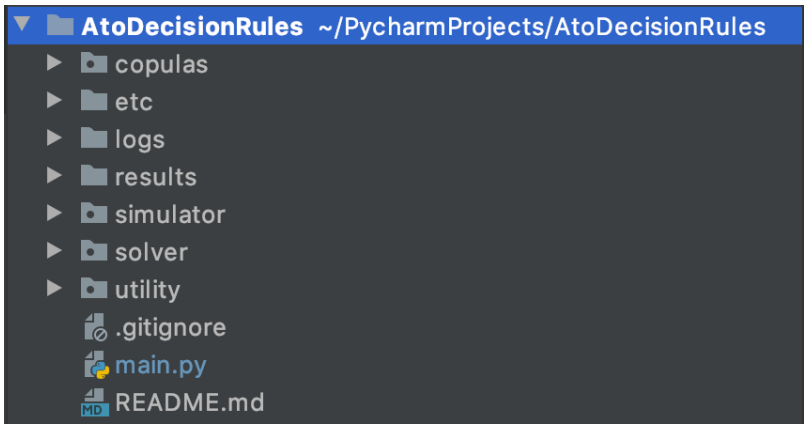

Figure 3: Package architecture.

then, when needed, our focus will be on some specific classes.

1. the `copulas` folder is a future work add-in for allowing the demand to be related in different ways from independence or linear correlation in case of jointly normal demands and will no be further discussed in here;

2. in the `etc` folder there are the `.json` file in which the user may specify some features related to the instance of the ATO problem he wants to solve (see ch. 4 for an in-depth explanation on what can be found in such files);

3. the `logs` and `results` are two folders in which are stored information about model creation, solving and performances, plus some graphics interpretation of the solutions if needed;

4. the `simulator` is the package in which there are the classes related to the instance generation and for the solutions evaluations, which are two of the three phases (and classes) we are going to cover in-depth in the rest of the work;

5. the `solver` folder is where models are created and therefore it is where the most important part of the code is situated.

6. the `utility` is the folder in which we store useful functions which compare models and write down results in `.csv` format that are then processed to obtain graphs.

3 Two-stage models

In this section, we build several models describing the ATO production planning problem over two periods of time. In the first period, components are made without any information on future demands of end products with constraints on the capacity of our machines, while in the second stage demands realize and we assembly our components into the final items, trying to maximize our profit under constraints related to the total components we have made and to the demands outcomes. Each of these models has some common and specific features, so, for the sake of parsimony, we introduce the common notation and sets for every model.

- $\mathcal{I} = \{1, \dots, I\}$ is the set for the components;
- $\mathcal{J} = \{1, \dots, J\}$ is the set for the end items;
- $\mathcal{M} = \{1, \dots, M\}$ is the set for production machines (resources);
- $\mathcal{S} = \{1, \dots, S\}$ is the set for scenarios that we use to discretize the distribution of random demand;
- d_j^s, the demand for end item $j \in \mathcal{J}$ in scenario $s \in \mathcal{S}$;
- π^s, the probability of scenario s (if not otherwise specified we are going to use Montecarlo sampling, so that $\pi^s = \frac{1}{|\mathcal{S}|}$ and scenarios are equiprobable);
- C_i, the cost of component $i \in \mathcal{I}$;
- P_j, the price (or revenue) of the end item $j \in \mathcal{J}$;
- L_m, the time availability (in the chosen temporal unit) of the machine $m \in \mathcal{M}$;
- T_{im}, the processing time for the component $i \in \mathcal{I}$ on machine $m \in \mathcal{M}$;
- G_{ij}, the number of components of type $i \in \mathcal{I}$ needed to assemble one end item of type $j \in \mathcal{J}$; in manufacturing parlance, these coefficients are called *gozinto factors* and are grouped into the *gozinto matrix* G;
- $I_{0,i}$ is the initial inventory for the component $i \in \mathcal{I}$ and it is set to zero if not explicitly stated otherwise;
- x_i is the quantity of component i made before the realization of the end products' demands and they are the only variables common to all models.

3.1 The basic problem: Expected Value formulation

The Expected Value (EV) model takes a stochastic problem and transforms it into a simple deterministic one, by simply taking expected value every time a risk factor is encountered. In this way we eliminate all the uncertainty of the problem, we do not really take care of it. This may not be the best approach, but it is for sure the simpler one and produces a small (or medium) size linear program. So the model has only J additional variables which are:

- y_j, the quantity of end item j produced.

The model then it is given by

$$\max \quad -\sum_{i=1}^{I} C_i x_i + \sum_{j=1}^{J} P_j y_j \tag{13}$$

$$\text{s.t.} \quad \sum_{i=1}^{I} T_{im} x_i \leq L_m \qquad \forall m \tag{14}$$

$$y_j \leq \bar{d}_j \qquad \forall j \tag{15}$$

$$\sum_{j=1}^{J} G_{ij} y_j \leq x_i + I_{0,i} \qquad \forall i \tag{16}$$

$$y_j, x_i \geq 0$$

where $\bar{d}_j$ is the mean demand for end item j.
We are optimizing over the decision variables $x_i,\ i \in \mathcal{I}$ and $y_j,\ j \in \mathcal{J}$. The objective function in eq. (13) is simply composed of the total revenues given by the sales of end items minus the total costs of the component we have made to meet the demand of our products. We have three types of constraints:

1. *capacity constraints*, which are written in form of eq. (14): they state that the time of completion of our components must not be greater of the time available on the machines we have. We emphasize that the capacity of our machines is not aggregated, so that the system represented may be seen as a real supply chain and may be affected by a bottleneck (i.e a machine that under-produces with respect to all the others);

2. *demand constraints*, which are in eq. (15) and tell that we cannot sell more end-products than the demand (in this simplistic case the demand coincide with the mean of the distribution)

3. *assembly constraints*, which are in eq. (16) and limit the assembly of final items: we cannot produce them if there are not enough components to build them, both just made or present in the inventory.

Finally, we have some lower bounds on the variable, which basically state that we cannot produce negative amounts of components nor assembly negative amounts of end items.
Talking about the dimension of the problem, we have $I+J$ variable to optimize and $M + J + I$ constraints, so that the problem grows linearly with respect to items, components and resources. Again we want to stress that there are other models to tackle the ATO problem which are harder to write and solve: for example, in our model, there are no fixed costs on machines use, nor the possibility of malfunctions of resources or quality checks on productions; we have kept deliberately simple our model to be able to carry out extensive comparisons in demand uncertainty and in how components' type (common or specific) act in changing the solution.

3.2 Recourse Second Stage version

This is the classical two-stage recourse problem with sample average approximation. In this model we need more second-stage variables with respect to the EV problem; in particular, we define

- y_j^s is the quantity of end-item j produced under the demand scenario s

and the problem becomes

$$\max \quad -\sum_{i=1}^{I} C_i x_i + \sum_{s=1}^{S}\sum_{j=1}^{J} \pi^s P_j y_j^s \tag{17}$$

$$\text{s.t.} \quad \sum_{i=1}^{I} T_{im} x_i \leq L_m \qquad \forall m \tag{18}$$

$$y_j^s \leq d_j^s \qquad \forall j,\ s \tag{19}$$

$$\sum_{j=1}^{J} G_{ij} y_j^s \leq x_i \qquad \forall i,\ s \tag{20}$$

$$y_j^s, x_i \geq 0$$

Basically, the main difference is that we now have a second stage variable for every end item and every possible scenario (therefore they are scenario dependent). This implies that the number of constraints of the type of eq. (19) grows from $\mathcal{J}$ up to $\mathcal{J} \times \mathcal{S}$, exactly the same number of second-stage variables, while the number of assembly constraints becomes $\mathcal{I} \times \mathcal{S}$. This is because market and assembly constraints must be satisfied in each scenario.
In this case, the number of decision variables is $\mathcal{O}(J \times S)$. What we dislike is the fact that this number is linear in S and this is not a good news, as the number of scenarios is usually quite big if we want to generate stable solutions. Luckily there are some algorithms already implemented in the `Gurobi` solver that maintain the computational burden low in this particular case of two-stage setting, deploying the particular structure of the constraint matrix. This will not be possible when multi-stage models are built.

3.3 Decision Rules models

We now shift the attention to the decision rules models: we present two models based on a linear and piece-wise linear characterization of the decisions, which still lead to linear programs of moderate size.

3.3.1 Model I: Linear Decision Rules version

We have the following second-stage variables in addition to the common first-stage x_i:

- $\bar{y}_j$ is the number of end item j produced in the standard condition;
- $H_{j,k}$ is the rate of variation of production of item j per one item variation in demand of item k;

Again, we use $\bar{d}_j$ as the mean demand for item j. The model then becomes

$$\begin{aligned}
\max \quad & -\sum_{i=1}^{I} C_i x_i + \sum_{j=1}^{J} P_j \bar{y}_j + \\
& + \sum_{s=1}^{S} \sum_{j=1}^{J} \pi^s P_j \big(\sum_{k=1}^{J} H_{jk} [d_k^s - \bar{d}_k] \big) && (21) \\
\text{s.t.} \quad & \sum_{i=1}^{I} T_{im} x_i \leq L_m \qquad \forall m && (22) \\
& \bar{y}_j + \sum_{k=1}^{J} H_{jk} [d_k^s - \bar{d}_k] \leq d_j^s \qquad \forall j,\ s && (23) \\
& \sum_{j=1}^{J} G_{ij} \big(\bar{y}_j + \sum_{k=1}^{J} H_{jk} [d_k^s - \bar{d}_k] \big) \leq x_i \qquad \forall i,\ s && (24) \\
& \bar{y}_j + \sum_{k=1}^{J} H_{jk} [d_k^s - \bar{d}_k]) \geq 0 \qquad \forall j,\ s \\
& x_i \geq 0 \qquad \forall i
\end{aligned}$$

The differences with respect to the previous model are concentrated in the second-stage decisions y. In this model we separate the standard (or better the mean) behaviour, which is represented by $\bar{y}_j$, the decision we adopt in case of a scenario in which the demand is equal to its mean, from the adjustments $H_{j,k}$ which are the best linear responses in the assembly of item j if demand for item k moves from its mean value.
This intuition has been emphasized in the objective function: first-stage costs and mean second-stage behaviours are independent of scenarios, while in eq. (21) we can see the profit coming from an adjustment in a specific scenario, multiplied by the probability of the scenario.
The constraints about capacity (eq. (22)) are equivalent to the previous model, since they concern only first stage solution, while eq. (24) and (23) are respectively the linear equivalent of eq. (20) and (19). So, basically, the idea behind this model is to substitute y_j^s of the Recourse Second Stage version (which in the standard model of the chapter 2 was $x_t(\xi^t)$ with $t = 1$) with $\bar{y}_j + \sum_{k=1}^{J} H_{jk}[d_k^s - \bar{d}_k]$ (which is affine with respect to our risk factors). In this way, we have linearized the recourse model and we have (linear) decision rules which are scenario independent in the sense that the rate of adjustment does not depend on the scenario considered. This may be a good feature in the out of sample performance, since highly non-linear scenario dependent adjustments may lead to first-stage decisions which perform

badly on a never seen scenario. In computational term the model presented is quite different with respect to the previous one: the number of parameters here is no more dependent on the number of scenarios, but the number of constraints still is. In fact, now we are optimizing over

- I first stage variables x_i,
- J second stage mean variables $\bar{y}_j$,
- $J \times J$ second stage adjusting variables $H_{j,k}$,

so that the total number of variables is $\mathcal{O}(J^2)$. The number of constraints remains equal to the recourse model just presented, since constraints must still be met in every scenario.
Giving the difference in the total number of variables we expect a better performance of the decision rules in terms of computational time when the number of scenarios is high enough. Unfortunately, this cannot be appreciated: as we pointed out before the structure of the recourse matrix in the recourse problem is suitable for faster methods (this is basically due to the fact that we have variables for each scenario), while the structure here is more complicated, since the rules in some sense aggregate all the scenarios together, so the solver is not able to spot a particular useful matrix structure.

3.3.2 Model II: Deflected Linear Decision Rules version

For this model we introduce the additional notation of

$$d^+ := \max(d, 0), \quad d^- := \max(-d, 0) \tag{25}$$

which is used in defining

$$d_j^{+s} = \max(d_j^s - \bar{d}_j, 0), \quad d_j^{-s} = \max(\bar{d}_j - d_j^s, 0) \qquad \forall s \in \mathcal{S}, j \in \mathcal{J}$$

because we have different coefficients for variations upward or downward of the demands from the expected value.
Therefore, with respect to the previous model, we have the following variables in addition to the first stage x_i:

- $\bar{y}_j$ is again the number of end item j produced in the standard condition;
- $H_{j,k}^+$ is the rate of variation of production of item j per one item positive variation in demand of item k;
- $H_{j,k}^-$ is the rate of variation of production of item j per one item negative variation in demand of item k

Then the deflected (or piece-wise) linear decision rules model is

$$
\begin{aligned}
\max \quad & -\sum_{i=1}^{I} C_i x_i + \sum_{j=1}^{J} P_j \bar{y}_j + \\
& + \sum_{s=1}^{S} \sum_{j=1}^{J} \pi^s P_j \Big(\sum_{k=1}^{J} H_{jk}^{+} d_k^{+s} + H_{jk}^{-} d_k^{-s} \Big) && (26) \\
\text{s.t.} \quad & \sum_{i=1}^{I} T_{im} x_i \leq L_m \qquad \forall m && (27) \\
& \bar{y}_j + \sum_{k=1}^{J} H_{jk}^{+} d_k^{+s} + H_{jk}^{-} d_k^{-s} \leq d_j^s \qquad \forall j,\ s && (28) \\
& \sum_{j=1}^{J} G_{ij} \big(\bar{y}_j + \sum_{k=1}^{J} H_{jk}^{+} d_k^{+s} + H_{jk}^{-} d_k^{-s} \big) \leq x_i \qquad \forall i,\ s && (29) \\
& \bar{y}_j + \sum_{k=1}^{J} H_{jk}^{+} d_k^{+s} + H_{jk}^{-} d_k^{-s} \geq 0 \qquad \forall\, j,\ s \\
& x_i \geq 0 \qquad \forall i
\end{aligned}
$$

As stated before, the only difference with respect to the previous model is due to the different coefficient for opposed moves from the mean: if the demand falls short we adjust following H^- matrix, while if demand goes over we adjust following the H^+ matrix.
Computationally speaking, this doubles the number of variables in the model, but the model has greater elasticity, so that an increase of the performances has to be expected.
Finally, at least for in-sample tests, this model outperforms the previous one, since the optimal choice of H coefficient in the precedent section is amongst the solution of this model, imposing $H^+ = H^-$ entry-wise.

3.4 A short dive in the code: model classes

In this chapter, we have presented 4 different models to tackle the ATO problem. From a software point of view, these models are all created in the `solver` folder of the repository. Specifically, expected value problem and elementary recourse share a class called `ato.py`, while two different classes contain respectively the linear and piece-wise linear decision models.

In snippet 1, it can be seen the basic structure of expected value and simple recourse problem. As stated before they share the same class and differ only in one initialisation parameter, the `stoch_type`, a string which can take values `"ev"` for expected value, `"st"` for the (stochastic simple recourse) or `"qn"` for the quantile version (the choice of the mean as a possible realization is subjective). Then this parameter is central in the `solve` method, where a conditional statement based

Listing 1 ato.py class skeleton

```
class Ato():
    def __init__(self, stoch_type, var_type="dis"):

    def solve(
        self, dict_data, cont_value=None, time_limit=None,
        gap=None, qnt_approx=0.8, verbose=False
    ):
```

on this value build the model in the desired way. Another important parameter of this function, which is common to all models classes is the `dict_data`, where there is all relevant information of the instance we want to solve. While we will talk about the `cont_value` parameter when referring to multistage evaluation, the other parameters are less important for the following discussion: `qnt_approx` is the quantile level in case of the quantile version of the problem, and all other parameters are settings for the gurobi solver.

Listing 2 atoDLDR.py class skeleton

```
class AtoDeflectedLinearRules():
    def __init__(self, a_priori_check=False, threshold=0, var_type="dis"):

    def solve(
        self, dict_data, cont_value=None, time_limit=None,
        gap=None, verbose=False
    ):
```

The Linear and Deflected Linear Decision Rules models are in two different classes for user ease, but share the same class attributes and so we now analyse only one of the two, the Deflected one. Contrary to the class before, in the `__init__()` method there is no information about the stochastic type, since in these models uncertainty is always taken into account. There is the same `var_type` parameter, which can be `"dis"` (integer variables) or `"con"` (continuous variables): even if the default is to treat integer variables, we strongly recommend to put this value to `"con"` for computational reasons. The two new parameters are deeply related and try to explicitly avoid over-fitting of the model. The parameter `a_priori_check`, if set to `True`, puts to 0 all the adjustment coefficients for items j with respect to demand movements of every item k for each item k which is dissimilar to item j. The similarity (or dissimilarity) is calculated as the percentage of common components between items j and k over their total number of components. The obvious question is how much dissimilar two objects need to be in

Listing 3 Adjustments creation and over-fitting check

```
H_plus = model.addVars(
            dict_data['n_items'], dict_data['n_items'],
            vtype=self.var_type,
            name='H_plus',
            lb=-GRB.INFINITY
        )
H_minus = model.addVars(
            dict_data['n_items'], dict_data['n_items'],
            vtype=self.var_type,
            name='H_minus',
            lb=-GRB.INFINITY
        )
if self.a_priori_check:
    for j in items:
        for k in items:
            if dict_data['items_metric'][j, k] <= self.threshold:
                model.addConstr(H_minus[j, k] == 0)
                model.addConstr(H_plus[j, k] == 0)
```

order to activate this check? The `threshold` parameter takes care of the answer and adjustments are set to 0 if the similarity of two objects is less than this value. In the listing 3 we can see an example of adjustment definition, by means of the `addVars()` method for a gurobi model object, which requires dimensions, type of the variable, a name, a lower bound (if different from the default value 0) and an upper bound (not present in this case since the default value $+\infty$ is the right one for our problem) and how the control for dissimilar products is carried out, with the use of the `addConstr()` gurobi method, which has a self-explanatory name.

4 Comparison and results: two-stage

Up to this point, we have presented 4 different models, trying to explain a priori, by means of intuition and number of variables and constraints, what are the pros and drawbacks of each one intuitively.
In this chapter, we solve different instances of the ATO problem with all the models and we end up with 4 different solutions that we want to test and compare. It is important to note that the global solutions coming from the 4 problems are structurally different. In fact, the part of the solution related to the components are equally dimensioned: $x^* = (x_1^*, \dots, x_I^*)$ are the optimal amounts of components to be made; the part related to the assembly of final products is really different:

- in the Expected Value solution, it is composed of J values, which represents for each final product the number of pieces to be sold;
- in the Recourse Problem solution, it is composed of $J \times S$ values, which represents for each final product the number of pieces to be sold in each scenario;
- in the Linear Decision Rules model there are $J + J \times J$ values, so we have $\bar{y}_j^*$ and H_{jk}^* which are respectively what to do in case of demand equal to the mean of the distribution from which the demand is sampled and how to react from the movement of the demand of the final product k;
- in the Linear Decision Rules model there are $J + 2J \times J$ values, so we have $\bar{y}_j^*$, H_{jk}^{+*}, H_{jk}^{-*} which are respectively what to do in case of demand equal to the mean of the distribution from which the demand is sampled and how to react from the movement of the demand of the final product k, with positive and negative deviation treated separately.

But are the second stage, assembly-related, part of the solutions really important in the evaluation of the solutions themselves? The answer is luckily no, so that one type of evaluation can be carried out for the 4 solutions.
In fact, second stage solutions depend on the particular scenarios used in the optimization routine (for the EV there a single scenario consisting in the mean of each demand), but after we implement first stage decisions for production, the real-life demand arrives and we try to do our best to obtain the largest possible revenues from what we have available, so that what really matters is just the components' production decisions.

4.1 How to understand the importance of accounting for uncertainty?

4.1.1 Value of Stochastic Solution

The last remark we have done is particularly important for the difference between the solutions coming from the recourse version of the problem and the expected value ones. This is because in the expected value we get rid of the uncertainty by

replacing the risk factors with their means or another fixed value.
This approach is sometimes the only viable option, but let arise an important question: how much are we losing by not considering the uncertainty information?
A popular measure to capture this fact is the Value of the Stochastic Solution (VSS).
To define it, let us introduce a new light notation for a stochastic program

$$\min_{x \in \mathcal{X}} \; z(x, \xi) = \mathbb{E}_\xi \big[c^T x + \mathcal{Q}(x, \xi) \big] \tag{30}$$

where $Q(x, \xi)$ is the so called recourse function and comes from the following minimization:

$$\begin{aligned} \mathcal{Q}(x, \xi) = \min_y \; & q_\xi^T y \\ \text{s.t. } & W y = h_\xi - T_\xi x \\ & y \geq 0 \end{aligned}$$

In this description, x is the first stage decision which does not depend on uncertainty and y are successive stage decisions depending on the risk factors ξ.
This implies that now we the recourse problem is simply

$$RP = \min_x \mathbb{E}_\xi \big[z(x, \xi) \big] \tag{31}$$

while the expected value problem can be written as

$$EV = \min_x z(x, \bar{\xi}) \tag{32}$$

where the bar over ξ signals the fact that we are considering the mean of each risk factor. While RP theoretically represents the mean value of the cost (we are in a minimization framework), EV is the cost under a particular scenario, so the two values are not fairly compared. A fair comparison can be achieved by plugging the first stage solution $\bar{x}_{EV}$ of the EV in the function z and minimize the mean under all scenarios: the logic is to check the performance over all the possible scenarios of a solution found discarding the uncertainty. In formulas, we calculate

$$EEV = \mathbb{E}_\xi \big[z(x_{EV}, \xi) \big] \tag{33}$$

and compare it with the RP.

Definition 4.0.1. We define the Value of Stochatic Solution as

$$VSS = EEV - RP.$$

Proposition 4.1. VSS is non negative.

Proof. Proof is straightforward: RP comes form the minimization over x of a certain quantity, while EEV it is the same quantity with a fixed x. In formulas, we have

$$RP := \min_x \mathbb{E}_\xi \big[z(x, \xi) \big] \leq \mathbb{E}_\xi \big[z(x, \xi) \big] \qquad \forall x.$$

In particular this is true for $\bar{x}^*$ so that

$$RP := \min_x \mathbb{E}_\xi\big[z(x,\xi)\big] \leq \mathbb{E}_\xi\big[z(\bar{x}^*,\xi)\big] =: EEV.$$

□

VSS is a very useful measure because let the modeller understand if the computational burden of employing all the machinery of the stochastic programming approach is worth or not: a high value of the VSS means that including the uncertainty in the model we are saving a good amount of money, if the value is small it may be the case that a more simplistic approach may lead to good enough solution for our purposes. It is important to understand two main points when carrying out the calculation for VSS:

1. for risk factors like the one present in the ATO problem, which come from a continuous distribution the VSS practically calculated is a sample average and not the exact value;
2. the recourse first stage solution is calculated on other samples of the distribution and not over the same ones used to find the result, otherwise the comparison would be too biased in favour of the latter.

4.1.2 Expected Value of Perfect Information

With the VSS we can characterize the importance of taking uncertainty into account. This measure is very useful because operatively tells the modeller if the extra effort to build a more sophisticated model generates a larger profit (or a thinner overall cost). Another precious information for the decision-maker can sometimes be about the role uncertainty has in the problem he faces, in terms of how much he is losing for not having perfect knowledge. Perfect knowledge of the future is obviously impossible in case of risk factors which are exogenously random, but sometimes more money can be invested to gain a better perspective on the business. With this idea in mind, we introduce the concept of the Expected Value of Perfect Information (EVPI). The idea is basically to compare the profit generated by the RP model with the one generated by a decision-maker that can decide the production after having seen the demand, known as the wait and see solution. Switching again into the standard minimization formalisation, we define the wait and see profit as

$$WS = \mathbb{E}_\xi\big[min_{x(\xi)}(z(x(\xi),\xi))\big] \tag{34}$$

so that we can first choose x and then we carry out the average process.

Definition 4.0.2. The Expected Value of Perfect Information is defined as

$$EVPI = RP - WS.$$

Proposition 4.2. EVPI is non negative.

Proof. Proof is again straightforward: WS comes from an expected value of a minimization while the RP is the minimization of the expected value. If we take x_{RP} as the RP solution of the RP model, we have that

$$z(x_{RP}, \xi) \geq z^*(\xi) = \min_x(z(x, \xi)) \qquad \forall \xi \in \Xi,$$

so we have that

$$WS := \mathbb{E}_\xi\big[z^*(\xi)\big] \leq \mathbb{E}_\xi\big[z(x_{RP}, \xi)\big] =: RP.$$

□

Unfortunately, EVPI is not as useful as VSS because it simply tells how much are we losing for our lack of information, but it is uninformative of how to improve. It is a good measure to understand how much uncertainty is deteriorating the profit and how much we are willing to pay at maximum to eliminate uncertainty from our problem.

4.2 Instance generation and important parameters

Once created the skeleton of the problems, the successive step is to build an instance of the problems to be solved. In the companion code of this work, the instance is a separate class from the problems classes (and an input for the latter) and it is initialized mainly through a `.json` file containing critical information that we now analyze with some detail.
Specifically, the file contains a dictionary with

- the number of end-items, the number of components and the number of machines (or resources);
- the number of common (to all items) and specific (single item) components;
- a dictionary containing the stochastic information about the demands, naming the dependency between demands, distribution from where to sample (in particular normal, left-skewed beta, right-skewed beta and uniform) and relevant info about these distributions (mean and variance for the normal, a, b and an interval for betas, the interval for the uniform);
- three intervals for the margin of three different categories of end-products: low, medium and high margin product;
- for the same categories, the percentage of items that are into one of the categories (percentage of high-profit products is implicitly defined);
- a range for the processing time of all the components, another one for the number of components (*processing_time_interval*) in each final product (*gozinto_factor*) and still one for the components' costs;
- a parameter called *tightness* that we are discussing deeply in a while.

Listing 4 An example of .json file

```
{
    "n_items": 35,
    "n_components": 50,
    "n_machines": 5,
    "n_scenarios": 1000,
    "n_common_components": 0,
    "n_specific_components": 10,
    "components_per_item": [5, 10],
    "dict_stoch":{
        "dependency": {
            "copula": "ind"
        },
        "marginal": {
            "distr": "beta",
            "a": 2,
            "b": 4,
            "low": 100,
            "high": 500
        }
    },
    "profit_margin_low": [0.05, 0.2],
    "profit_margin_medium": [0.2, 0.4],
    "profit_margin_high": [0.4, 0.6],
    "perc_low_margin_item": 0.4,
    "perc_medium_margin_item": 0.3,
    "processing_time_interval": [0, 6],
    "gozinto_factor": [0, 6],
    "component_cost": [1, 5],
    "tightness": 0.8
}
```

An example of such information is the code snippet 4:

From the data settings, we further generate:

1. scenario demands, which are built using stochastic information and the number of end-products (in the file) by means of the method `generate_demand()` of the `instance` class;

2. the gozinto matrix, choosing randomly from the gozinto_factor with some constraints related to the number of common and specific components;

3. the time processing matrix, which is built by choosing randomly again from the processing_time_interval;

4. the limits to the capacity are set in a way that tries to stress the production and it is worth to be deeply explained. Indeed, the limits are found by solving a simplified version of the expected value problem, without the capacity constraints (14). Then, the optimal production solution x^* is left-multiplied by the time processing matrix to understand how much time-machine is used in each resource. Finally, this quantity for each machine is multiplied by the tightness factor, so if this parameter is less than one we are in a condition in which we cannot produce enough to satisfy mean demands. We have done this because we were unable to obtain real data, so this was a meaningful way to create synthetic ones.

4.2.1 A short dive in the code: two stage instance class

As stated before, we have created a separate class to consider the instance in order to fill the skeleton of our models. In this way, we can keep separate the structure of the model from the data with which we need to solve the problem. In the listing 5 it can be seen the method declaration of the `Instance` class, that we now discuss. Beginning with the `__init__()` method, the `sim_setting` attribute is simply

Listing 5 Instance initialisation

```
class Instance():
    def __init__(
        self, sim_setting, init_inventory=None,
        plot_gozinto_matrix=False, gozinto_style=None, sampling_type=None
    ):
    def _simulate_gozinto(self, gozinto_style):
    def get_data(self, n_scenarios=-1):
    def generate_demand(self, n_scenarios=-1, sampling_type=None):
    def _simulate_marginal_distribution(self, dict_distr, size):
    def create_metric(self):
    def update_info(self, sol, assembly, cv_coeff):
```

the json file similar to the one in the listing 4, loaded in python, `init_inventoy` is an array which (if not null) tells what is the initial inventory, while the other parameters are related respectively to plots, particular shape in the gozinto matrix and to demand generation. The method performs what we have described in the preceding section, so basically generate all data necessary to fill a model. Inside it other methods of the class are called, in particular:

- the `_simulate_gozinto()` to generate the gozinto factors. This method requires information about the maximum number of components of the same

type as well as the maximum number of the different type of components, the number of common and specific components. There is an optional parameter for the style of the gozinto matrix; we know from the literature three very well known and studied cases:

1. the M-shaped version;
2. the W-shaped version;
3. the nested version.

A graphical representation of such a system is given by fig. 4.2.1. These special cases are interesting because they allow, under some assumption on the distribution of the demands, to find analytically the optimum for low dimensional instances. Since our interest is for medium-sized instances, we do not experiments with them, but the interested reader can refer to Nadar et al. (2014), Elhafsi et al. (2015) or Reiman and Wang (2015).

- the `generate_demand()` to generate the demand on which the models are optimized. This method is quite self-explanatory; when the default value is used for the first attributes, this implies that scenarios are generated in the number read in the json file, otherwise in the parameter-specified number. This function call also `_simulate_marginal_distribution()` to sample from same known distributions of the random library (when needed), and characteristics useful in describing the distribution are used.
- the `create_metric()` to create a dissimilarity matrix to use to control interactions among different products in decision rules models. This method takes the gozinto matrix as input and returns the dissimilarity matrix between final products.

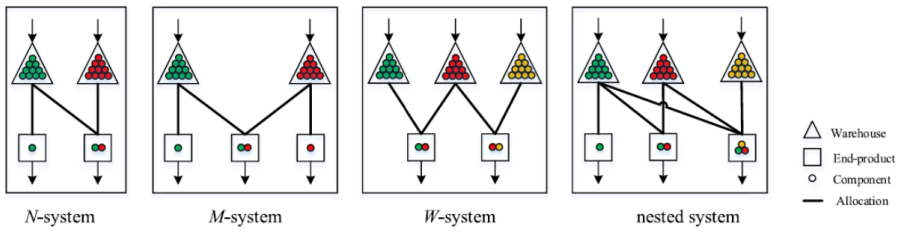

Figure 4: Graphical representation of the classical gozinto schemes.

The last two methods are `get_data()`, which creates the dictionary with all the information that will be passed to the problem class, and `update_info()`, which is a useful method when we will talk about multistage evaluation, since it allow

to change the values of initial inventory (and continuation values, which we will discuss deeply later).

4.3 In sample and Out of sample stability

In order to evaluate a solution, the first property to be checked is its stability, hence how much the objective function or the solution itself changes when different samples are available. In our case, we are interested in the stability of the objective function, which is often the case when the problem is business-related (there is a keen interest on profits or costs and not on how the result is achieved). The first basic request, which is a sort of minimum requirement to believe in the goodness of the solution is *in sample stability*. The idea behind in sample stability is quite simple. The scope of the stochastic program is to find a solution which yields the best possible mean profit (or cost). In formulas, using a simplified notation, we want to find

$$x^* := \arg\min_{x \in \mathcal{X}} f(x, \xi),$$

where ξ here represent the true probability distribution, but, because of computational limitations, we can only obtain

$$x_i^* := \arg\min_{x \in \mathcal{X}} f(x, \mathcal{T}_i),$$

where $\mathcal{T}_i$ is a generic sample of the real distribution.
In sample stability is basically asking that, given two different samples $\mathcal{T}_i$ and $\mathcal{T}_j$ and their respective optimal solutions

$$x_i^* := \arg\min_{x \in \mathcal{X}} f(x, \mathcal{T}_i) \tag{35}$$

$$x_j^* := \arg\min_{x \in \mathcal{X}} f(x, \mathcal{T}_j), \tag{36}$$

it must hold that

$$f(x_i^*, \mathcal{T}_i) \approx f(x_j^*, \mathcal{T}_j).$$

In sample stability is, therefore, a request of similar results when evaluating a solution against the set of scenarios on which the solution has been obtained, but what can we expect when evaluating the solution over a new set of scenarios? The short answer is that, with only in sample stability, there are no guarantees about performance on a never seen scenario. To overcome this issue the stronger notion of out of sample stability comes into play. Out of sample stability requires to evaluate different solutions (coming from optimization over different samples of the distribution) with respect to a set of scenarios which is different from the one used to obtain the solution. It can be thought as the validation step in a machine learning paradigm, where the scope is to have an insight into the true performance of the algorithm.
Out of sample evaluation can be computationally expensive for multistage stochastic programs, as a costly rolling horizon simulation procedure is needed. In our

two stages case, this check is not hard, as it requires solving many second stage problems. In fact, giving two solutions x_i^* and x_j^* defined as eq. (35) and (36) and three sets of scenarios $\mathcal{T}_i$, $\mathcal{T}_j$ and $\mathcal{T}_k$, two possibilities are available to check if solutions are stable out of sample:

1. with the *cross check* we use only $\mathcal{T}_i$ and $\mathcal{T}_j$ and check if

$$f(x_i^*, \mathcal{T}_j) \approx f(x_j^*, \mathcal{T}_i)$$

2. with an *independent check* we want

$$f(x_i^*, \mathcal{T}_k) \approx f(x_j^*, \mathcal{T}_k)$$

The cross-check is quite useful if there are some difficulties in finding new data and may be considered the equivalent of the cross-validation approach in machine learning, while the independent check is more similar to the classical validation set approach.

It is important to stress that stability is a relative concept as it depends on the type of the problem: if the problem requires a very precise result maybe there is satisfaction only with solutions that range in an interval very tight (for example an interval with length less than 1% of the mean value of the objective function), other times we can be satisfied with a broader range for our solutions.

A priori, since LDR and DLDR models are a simplistic approximation of the recourse model, we expect to find that stability of the LDR and DLDR solutions is achieved with a shorter amount of scenarios than the stability of the two-stage recourse model.

4.4 Comparison between stable solutions

Given that for all models it seems that 500 scenarios are enough to generate stable solutions, we have set the number of scenarios to this value, with a seed for exact replication of our results, and we have run all the models, so at this stage, we basically have 4 vectors x_{RP}, x_{LDR}, x_{DLDR} and x_{EV}.
In particular, for the solutions we are analysing, we have generated the instance with setting 35 end-items, 50 components, 5 machines and then we have created all other parameters. The aforementioned settings are kept fixed for every experiment we have carried out, while now we describe which are the parameters we have varied to gain an insight on possible critical behaviours about the different models we propose. Specifically, we are interested in

- the *role of uncertainty* of the demands, so we have tried 4 different distribution settings, with differences in mean, volatility, skewness and finiteness;
- the *role of specificity* of components, how much common and specific components are critical for a solution to perform well. It can be logic to expect that

the more common components, the easier is to manage the components' production because of the mitigation effect on the overall variance of the total demand for risk pooling. On the contrary the more specific components, the riskier to produce them in case of an adversary demand situation;

- the *role of profit margins*, how a solution change if there are more low margin items or vice versa if there are more medium (or even high) margin products.
- the *role of capacity* by means of the tightness parameter, which can give an insight about where to put additional efforts or resources in order to avoid bottlenecks in production.

We are obviously interested in these analyses for all the models we have just presented, to understand if some of the models may be discarded, which are the most remunerative ones, how risky they are in terms of variability of the results. An obvious question is which metric to use in order to evaluate the 4 solutions. The first obvious metric, since we are solving a business problem, is related to the profits the firm made using the proposed solutions. We indicate profits with letter p from now on, with the subscript of the type of problem if needed for clarity. Another popular evaluation of the performance of a firm is the *return on investment*, which is usually abbreviated in ROI.
ROI is defined as the profits made over the total costs afforded to generate them, so, indicating by C the global costs, we define the ROI

$$\rho = \frac{p}{C} \tag{37}$$

and again we will use the subscript of the problem when needed.

With these solutions, we have solved 100 second stage problems with fixed first stage solution (equal to the ones we have previously found) and we have registered for each sub-problem both the ROI (return on investment) and the overall profit (or loss). By observing the differences between the profit and ROI of the recourse problem and the expected value, we can also assess the importance of the uncertainty in the problem, i.e the Value of Stochastic Solution (VSS). From both the following figures and tables in section A, it is clear that the VSS is quite high no matter the distribution used. The difference is less evident in the favourable settings where a lot of common components are present or where profitability is higher.

Talking about the EVPI, it is clear from the table A that uncertainty has a really deep impact, with a high value of the information: we can see a structural behaviour similar to the VSS insight, with the profits possible with the recourse version of the problem that is generally around the 55% of the profits generated with the perfect information in the more averse case of no common components and low profitability, while the gap reduces in a more convenient situation, varying from 90 to 95%. This suggests that it is crucial for a firm operating with an ATO system to collect as much as possible in terms of data and information about demand. Tables from which the following figures have been created are available

in the Appendix. After these premises, we are ready to take a look at the plots (fig. 5, fig. 6 and fig. 7). In the first four plots of Fig. 5, we can appreciate, from

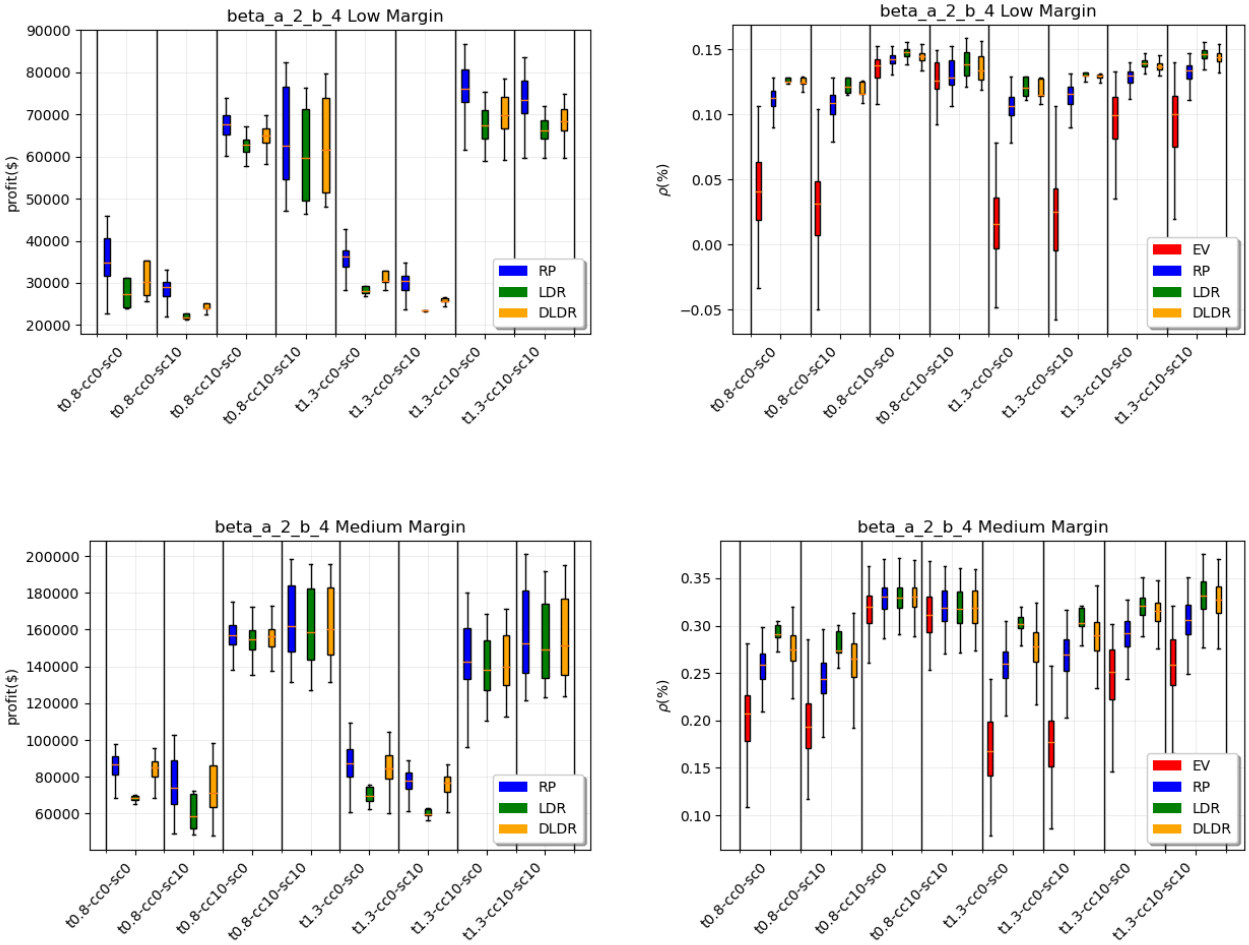

Figure 5: profits (left) and rho (right) for different margin level with a right skewed beta distribution.

both a profit and a ROI perspective, how the profitability is crucial for LDR model to be comparable with respect to both DLDR and the recourse model when the demand is distributed as a right-skewed beta. DLDR suffers less from this feature, staying almost at the same level of the recourse version both when margins are medium and low. Amongst the two decision rules, LDR shows a significantly worse expected profit and so a major grade of sub-optimality, with a gain in terms of less uncertainty in the profits, as it can be seen from the very tight box-plot.

The behaviour of these decision rules models can be explained by a simple reasoning: the two approaches differ because of the possibility of having different adjustments in case of negative or positive variation from mean demand leads to an higher mean sales' plan, with positive adjustment which are smaller. The fact that LDR cannot do that implies a smaller mean sales' plan and therefore a more conservative production plan, which is able to sell almost everything that is produced, but creates more unsatisfied demand which would carry on more costs and more profits too.

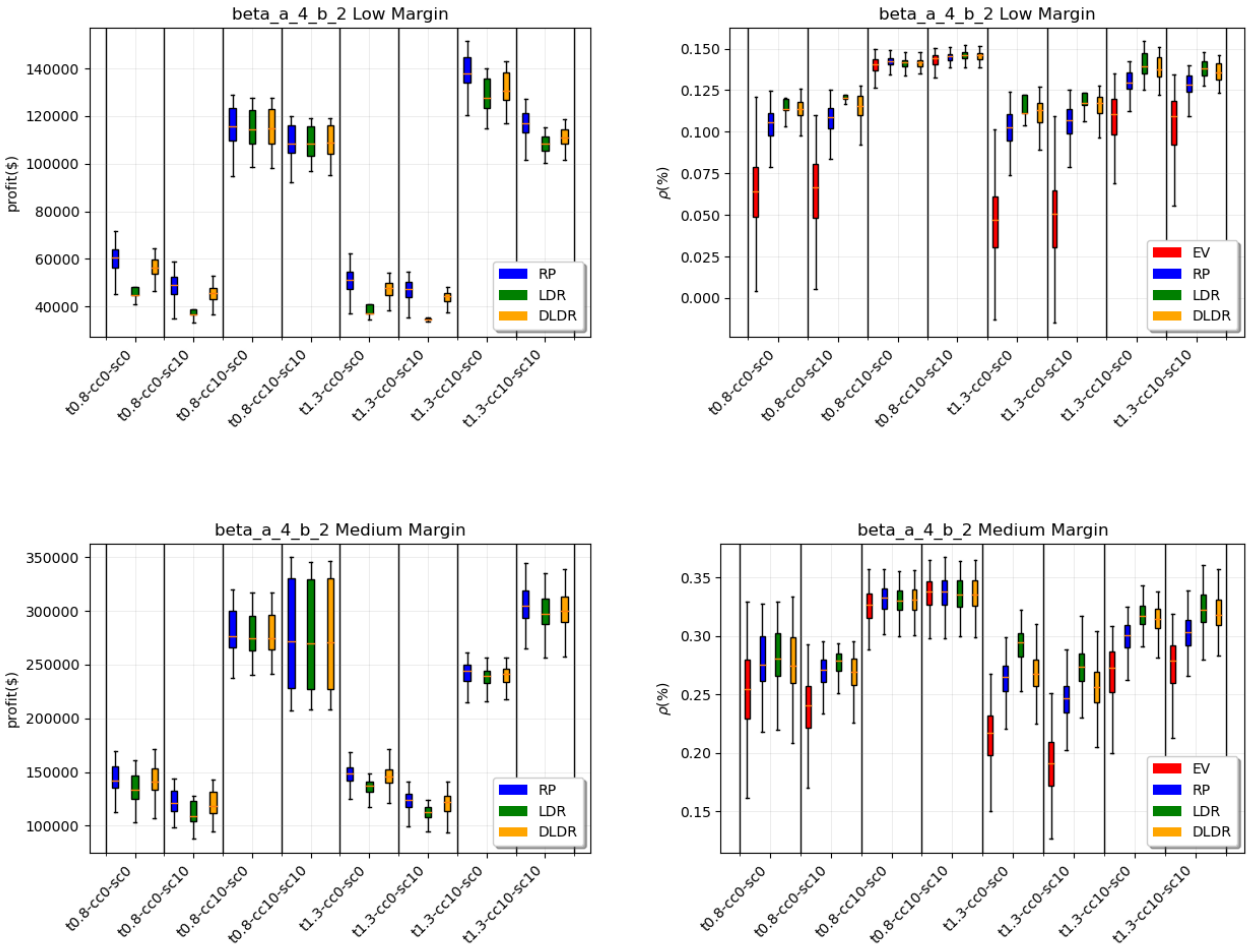

Figure 6: profits (left) and rho (right) for different margin level with a left skewed beta distribution.

So, making few components with respect to the other approaches guarantees that the majority of the items assembled is sold and therefore with low costs it is not surprising that the ROI is high and the overall profit is very low. This also reflects on the variances of both profits and ROI and, since very few items are assembled and sold, it is quite uncommon to not sell everything and so profit (and ROI) tend to be stable and low (high).
As stated before, this pattern is particularly evident for LDR, while DLDR seems to suffer from this problem in critical situations, like for example when there are no common components and 10 specific components; nevertheless in this situation, maybe, a conservative plan is what a firm should implement as there is no risk pooling effect and the demand risk is exacerbated by very specific items.

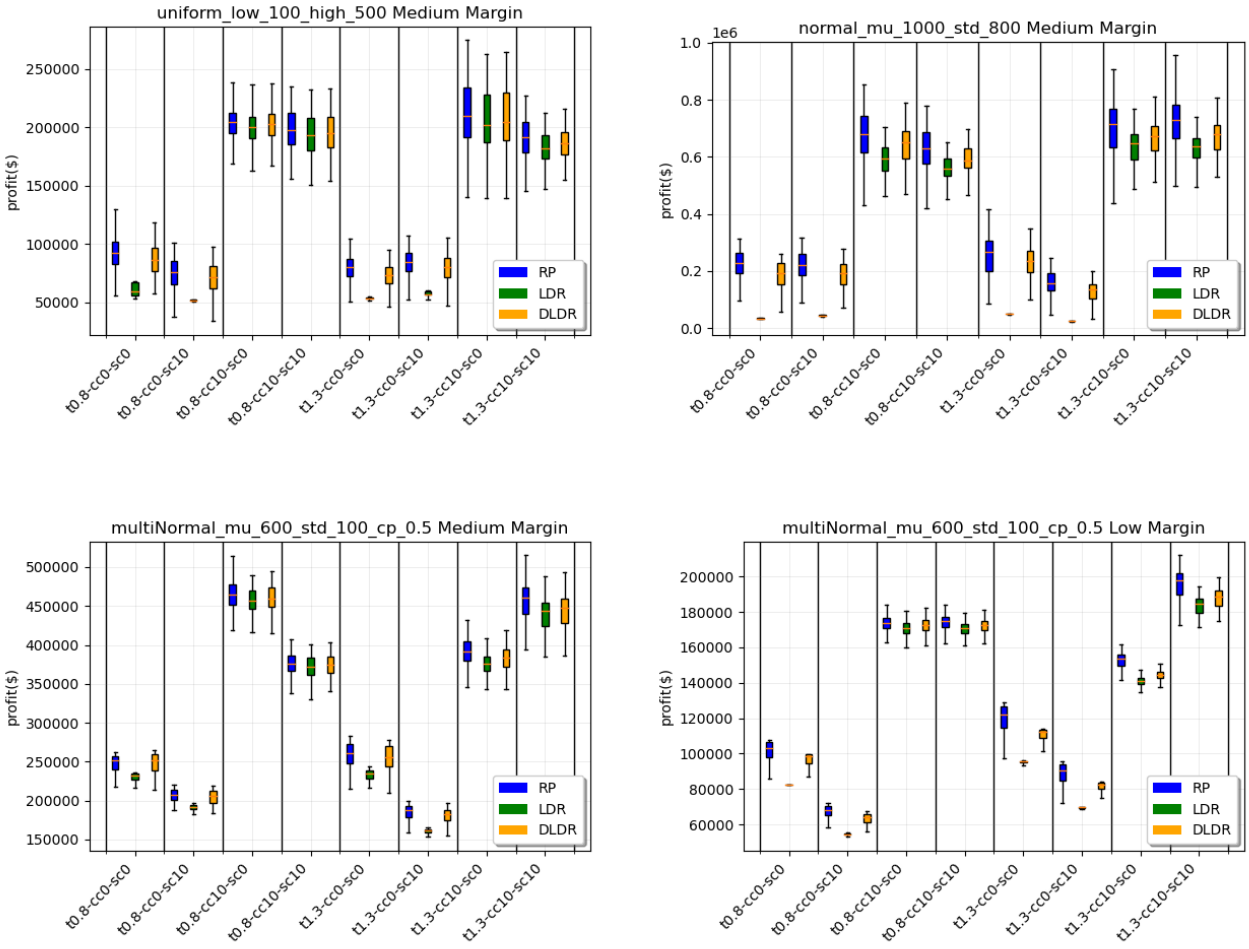

Figure 7: Profits for some symmetric distributions.

In Fig. 6, we can appreciate what happens when the distribution is a left-skewed beta. The pattern is similar, but it is slightly better for the LDR solution, with results which are tighter to the other two models more often than before. This is because now the mean behaviour is towards left and there is room for a more aggressive mean sales' plan without worries about satisfying the demand constraints. Anyway, exactly as before, the low margin case is still very difficult to manage for the LDR model, especially when there are no common components, which is another quite risky situation for business in any case.

The poor behaviour of LDR in these situations is even more clear for medium margin and symmetric distribution, like uniform distribution when there are no common components or in every case of a normal distribution, as we can see from Fig. 7.

Finally, before jumping into the last section, we briefly summarize the overall evaluation done over these 4 methods. In a trade-off between mean value of profits and ROI, the recourse problem is probably the best approach to be used, as one can suppose. A good competitor is the DLDR approach, which is quite close to the recourse formulation, with a little more conservative solutions which in some cases, like for example in the normal distribution low margin case of Fig. 8. A possible explanation is that in this case adjustment are difficult because of the unlimited nature of the demand, so that big movements are possible and this is combined with very low profitability that discourages large production, so a super conservative plan is adopted. LDR are poorer with respect to the DLDR, which

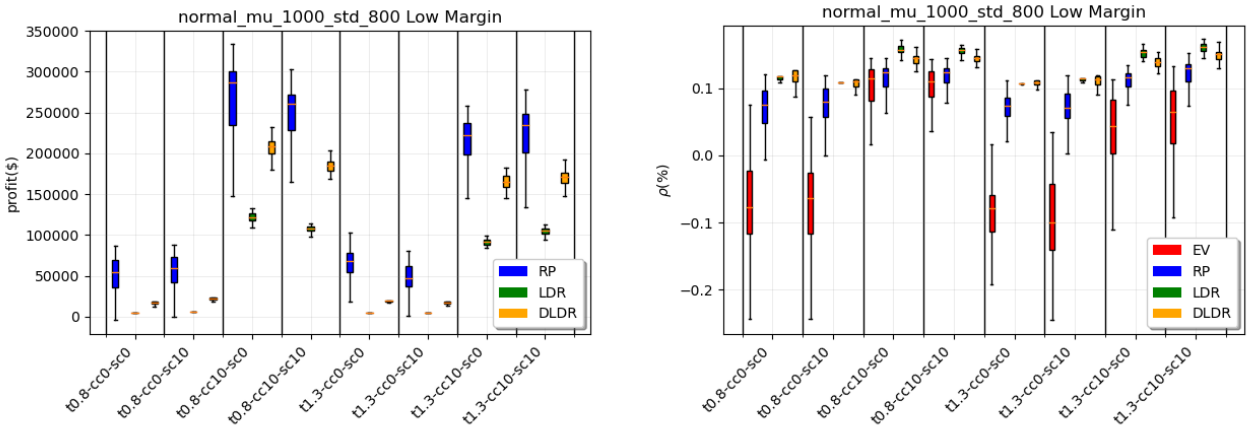

Figure 8: Normal distribution low margin is a critical situation for all models expect the recourse one.

in some sense justifies the major computational burden of the latter method. In general, with LDR, solutions are very conservative no matter the settings if there

is a limited number of common components, and this leads to a quite lower overall profit, very stable, but to a surprisingly high ROI: this is because the profit is lower but also costs are lower. It is noteworthy to mention that in case of a good number of common components (which implies a risk-pooling effect on the end-items demands) LDR solutions are quite similar to the other two methods. Lastly, limited support of the distribution (which is an obvious requirement for real-life problems) seems to be a relevant factor concerning the demand distribution, left and right skewness are an important factor for decision rules approaches: in particular, right skewness leads to a conservative LDR solution. The combination of number of the common components and profitability is instead crucial for all the models and in particular for decision rules approaches to generate aggressive solutions which are comparable to the recourse problem plan in terms of absolute profit.

4.4.1 A short dive in the code: two stage evaluation class

The two stage evaluation is another class. The `compare_sols_lst()` method re-

Listing 6 Two stages evaluation skeleton

```
class EvaluateTwostage():
    def compare_sols_lst(
        self, inst: Type[Instance], sols: List, labels: List, n_scenarios: int,
        get_rho=False, get_both=False,
        file_path=None, verbose=False,
        get_raw_data=False, show_graph=False
    ):
    def solve_second_stages(
        self, inst: : Type[Instance], sol: List, n_scenarios: int,
        demands: np.array, get_rho: bool, get_both: bool, var_type="cont"
    ):
```

quires an `Instance` class, which is the one on which a list of solutions (`sols` in the code) have been found. Parameter `labels` has the same length as the previous one and it contains the name of the problems of the related solutions. The `n_scenarios` simply tells the evaluator on how many scenarios to carry out the evaluation. `get_rho` indicates if we want to store ROIs (`True`) or profits (`False`) and it is active in the function only if `get_both` is set to `False`, otherwise both metrics are kept. Last important parameter is `get_raw_data`: if set to `True` the function returns a dictionary with, for each label, a list of result for ROIs and (or) profits. If set to `False` some key facts about ROIs and (or) profits are returned (mean, standard deviation, min and max). The `solve_second_stage()` method is called inside the `compare_sols_lst()` and, as the name may indicate, solves the second stage problems with a fixed first stage solution (`sol` in the parameters).

5 Robust models

As the actual pandemic has pointed out, sometimes the goal of a firm should be to make decisions which allow surviving even in very complicated (and potentially unexpected) situations. The idea is to find solutions that perform well in stressed conditions (in other words "robust"), at the expenses of possible conservativeness behaviour in good times. Robust optimization is a popular branch of the optimization and in its naive approach, it lays down to a max min (for profit) or min max (cost) problems. The interested reader may find a great detail of algorithms and robust formulation in Ben-Tal et al. (2009). In this work, we focus on this max min approach, where the min is taken with respect to the scenarios used in the optimization routine and the max is related to the decisions to take. We propose a robust formulation for the recourse problem and then we shift our attention to the decision rules formulation. Some alternatives of the classical robust version and its decision rules counterparts may be found in Brandimarte et al. (2020), where models are coupled with some financial-related risk measure such as Value at Risk (VaR) or Conditional Value at Risk (CVaR) because of the strong links with the ATO problem with portfolio optimization (assets are the components and products are portfolios which generate random returns).

5.1 Max Min formulation

The model coming from a max min problem, where the maximization is respect the decision variables x_i, y_j^s and the minimization is over the set of scenarios.

$$\max \quad -\sum_{i=1}^{I} C_i x_i + \min_{s\in\mathcal{S}} \sum_{j=1}^{J} P_j y_j^s \tag{38}$$

$$\text{s.t.} \quad \sum_{i=1}^{I} T_{im} x_i \leq L_m \qquad \forall m \tag{39}$$

$$y_j^s \leq d_j^s \qquad \forall j,\ s \tag{40}$$

$$\sum_{j=1}^{J} G_{ij} y_j^s \leq x_i \qquad \forall i,\ s \tag{41}$$

$$y_j^s, x_i \geq 0$$

The problem in this version, despite the sample approximation, is still not suitable for implementation because it is in a non linear form.

5.2 Robust version of the recourse model

The problem can be readily cast into the LP framework by adding an auxiliary variable t in the following way:

$$\max \quad -\sum_{i=1}^{I} C_i x_i + t \tag{42}$$

$$\text{s.t.} \quad t \leq \sum_{j=1}^{J} P_j y_j^s \qquad \forall s \tag{43}$$

$$\sum_{i=1}^{I} T_{im} x_i \leq L_m \qquad \forall m \tag{44}$$

$$y_j^s \leq d_j^s \qquad \forall j,\ s \tag{45}$$

$$\sum_{j=1}^{J} G_{ij} y_j^s \leq x_i \qquad \forall i,\ s \tag{46}$$

$$y_j^s, x_i \geq 0$$

5.3 Robust version of DLDR model

Since also in the previous model were present some recourse variables, it may be the case to transform these variables in a decision rules fashion. To maintain a certain consistency among the different parts of the work, we have developed both the LDR and DLDR version of the problems, but for the sake of parsimony (and also because of the extremely conservativeness of LDR without even asking for robustness), we only present the former. Then the model becomes

$$\max \quad -\sum_{i=1}^{I} C_i x_i + t \tag{47}$$

$$\text{s.t.} \quad t \leq \sum_{j=1}^{J} \pi^s P_j \Big(\sum_{k=1}^{J} H_{jk}^{+} (d^{+})_k^s + H_{jk}^{-s} (d^{-})_k^s \Big) \qquad \forall s \tag{48}$$

$$\sum_{i=1}^{I} T_{im} x_i \leq L_m \qquad \forall m \tag{49}$$

$$\bar{y}_j + \sum_{k=1}^{J} H_{jk}^{+} (d^{+})_k^s + H_{jk}^{-} (d^{-})_k^s \leq d_j^s \qquad \forall j,\ s \tag{50}$$

$$\sum_{j=1}^{J} G_{ij} \big(\bar{y}_j + \sum_{k=1}^{J} H_{jk}^{+} (d^{+})_k^s + H_{jk}^{-} (d^{-})_k^s \big) \leq x_i \qquad \forall i,\ s \tag{51}$$

$$\bar{y}_j + \sum_{k=1}^{J} H_{jk}^{+} (d^{+})_k^s + H_{jk}^{-} (d^{-})_k^s \geq 0 \qquad \forall j,\ s$$

$$x_i \geq 0 \qquad \forall i$$

5.4 Robust results

From robust models, we expected solutions which lead to results less volatile in terms of profit, but also less profitable (in the mean case). We have carried out experiments proposing a comparison with all previous models.
Since the Robust version of the recourse problem was already present in Brandimarte et al. (2020), we focused only on the Decision Rules version. This results will be denoted by the acronym RDLDR (Robust Deflected Linear Decision Rules).
The behaviour, in this case, lies between the pure DLDR and the LDR: solutions are generally more conservative than the DLDR and so profits are lower as it is its variance and sometimes a very concentrated box-plot signals the presence of a really conservative plan which reminds us about a plain LDR solution.
We present here (Fig. 5.4 and 5.4) some of the most significant plots regarding the comparison between our robust model and precedent ones. As one can see from

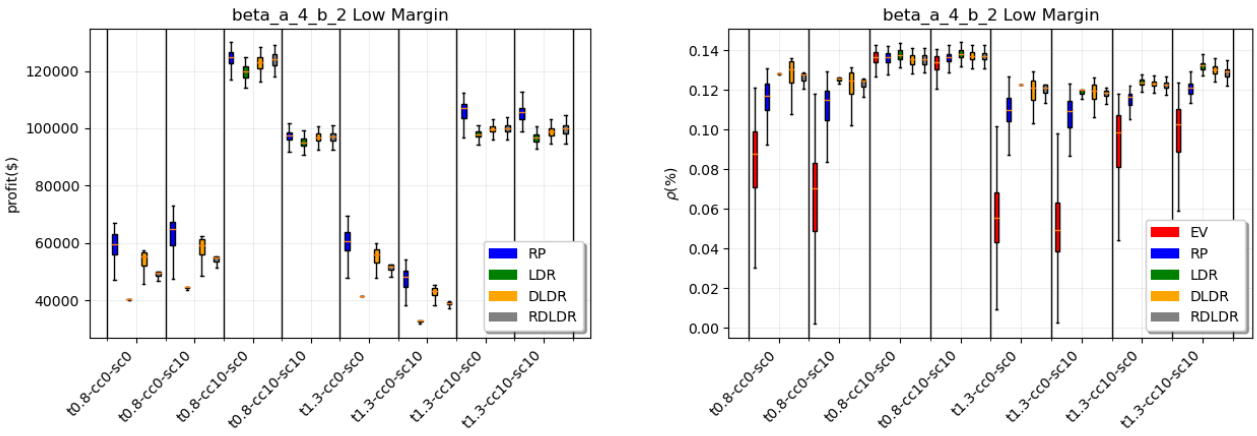

Figure 9: Extreme behaviour of Decision Rules approaches caused by a mix of low margins and no common components.

Fig. 5.4 with medium margin it is noticeable that the behaviour is exactly as expected: less variance with respect to DLDR and a lower mean profit, but the price of the robustness (the difference between mean profits) seems overall acceptable. Obviously, there are some pitfalls in the model, as the low margin situation exacerbate, where the robust model behaves almost exactly as the LDR, but as we have already discussed, this situation is amongst the worst possible for a real business, so a very conservative plan may be a good solution in this case. An example with the same distributions may be the figures in Fig. 5.4.

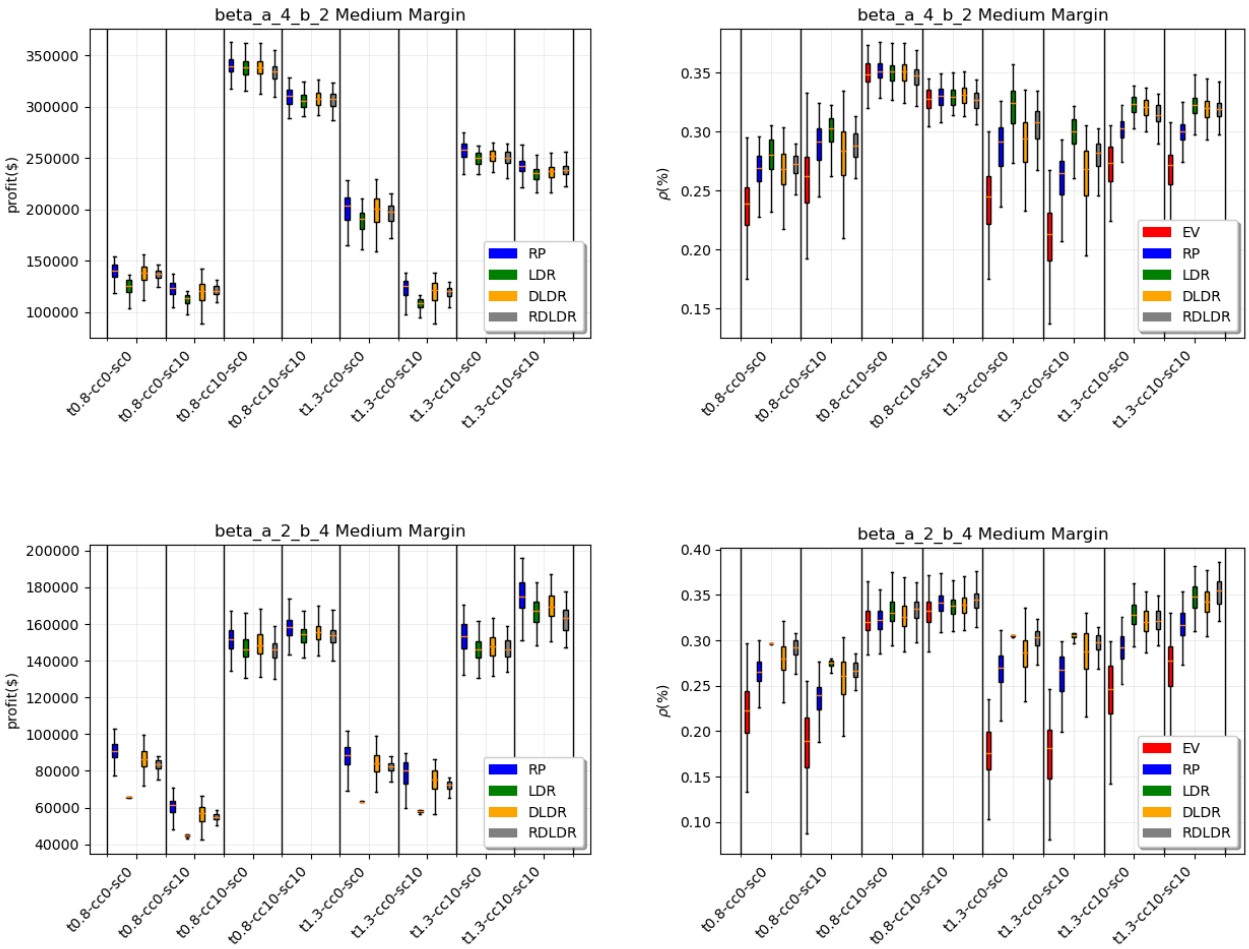

Figure 10: profits (left) and rho (right) for different skewness of a beta distribution.

6 Multistage models

Two-stage models in section 3 give an insight into what a firm should do now, looking at what may happen in a subsequent period of time. Despite the useful information these models may provide, a firm's cycles of production usually last for more than two periods and therefore multistage models come in handy. The decision they output is less myopic: they consider a larger time horizon and the benefits of far-in-to-the-future events. From a practical point of view, in the problem we are analysing, the multistage perspective must take into account a new state variable: the inventory. We consider a pure ATO system, so that inventory is kept only for components and not for the end products. Furthermore, we keep our models relatively simple and we restrict our analysis for non-perishable components, so that we can keep a component in the inventory as long as we want. A possible real-life industry case in which a similar system is involved may be the automotive case: in an industrial plant manufactured cars are not stored, but only components, which are non-perishable and the technological development is not as fast as in the high-tech sector, so that it is uncommon to discard some pieces of the inventory due to obsolescence. We think that this setting is not too restrictive for almost every ATO systems if time intervals considered are wisely chosen and therefore our decisions are widely applicable in real-life instances.

6.1 Multistage recourse model

As we pointed in Chapter 2, multistage stochastic optimization programs suffer from the curse of dimensionality, and the following model, unfortunately, is not an exception. Variance reduction techniques are a clever approach to reduce the variance of the distribution from which we sample out, but since our experiments are based on the difference between some synthetic distributions, it makes little sense to use them. Anyway, variance reduction is very useful in a more data-driven approach and the interested reader may check some details on these techniques in Brandimarte (2014). Another popular technique is to be more precise in sample out of a distribution for the first time periods and to sample less the more time goes by. This is because, in real-life applications, we may rely on good models for demand prediction over a short term, while models become less and less accurate the more we augment the elapsed time and so we are unsure about the true distribution of far-away-in-time demands. This leads to different branching factors in different time periods: by doing so, we change the number of scenarios from $|S|^T$ to $|S_1| \times \cdots \times |S_T|$, where S is the (common) number of scenario generated in an equally branched fashion and $|S_1| \dots |S_T|$ are the number of scenarios at each time instant considered. For example, with $T = 6$ we can have a very similar number of scenarios by setting $|S| = 10$ or $|S_1| = 50$, $|S_2| = 35$ and $|S_t| = 5$ $t = 3, \dots, 6$: it is clear that, if we plan to rerun the model as the time goes by, it is better to characterize well the first two instants and have a less precise view of the further future. In general, we can observe that there is a trade-off between initial and

final precision and it is part of the job of a good modeller to choose the right uncertainty sampling model. In the companion code of this work, we have implemented the following model with a scheme where the branching factor was decreasing in time, since business models like the ATO are usually re-optimized at each new time instant. Now, we present the multistage recourse version of the problem, an extension of the second model of section 3.

$$\max \quad -\sum_{t=0}^{T-1}\sum_{i=1}^{I} C_{i,t}\sum_{s=1}^{S}\pi^s x_{i,t}^s + \sum_{t=1}^{T}\sum_{j=1}^{J} P_{j,t}\sum_{s=1}^{S}\pi^s y_{j,t}^s \tag{52}$$

$$\text{s.t.} \quad \sum_{i=1}^{I} T_{i,m} x_{i,t}^s \leq L_{m,t} \qquad \forall m,\ s,\ t \geq 0 \tag{53}$$

$$y_{j,t}^s \leq d_{j,t}^s \qquad \forall j,\ s,\ t \geq 1 \tag{54}$$

$$\sum_{j=1}^{J} G_{i,j} y_{j,t}^s = x_{i,t-1}^s + I_{i,t-1}^s - I_{i,t}^s \qquad \forall i,\ s,\ t \geq 1 \tag{55}$$

$$x_{i,t}^s = x_{i,t}^{s'} \qquad \forall s' \in \{s\}_{t-1}, \qquad \forall t \tag{56}$$

$$y_{i,t}^s = y_{i,t}^{s'} \qquad \forall s' \in \{s\}_t, \qquad \forall t \tag{57}$$

$$y_{j,t}^s, x_{i,t}^s, I_{i,t}^s \geq 0 \qquad \forall i,\ j,\ s,\ t$$

Let's analyse in detail the model.
The most critical part of the model is perhaps the constraints in Eq. (56) and (57). They are respectively the non-anticipativity constraints for x variables and y. We have used the very common notation of $\{s\}_t$ to indicate all the scenarios s' which are indistinguishable to s up to the time t. It is important to note that the variables related to the component making are in the objective function now there is a sum over scenarios also for the components' making variables, which are a decision process predictable with respect to the information flow (i.e filtration) generated by the end-items' demands. We can appreciate this fact from the constraint in Eq. (56), where, at the stage t, $x_{i,t}^s$ must be the same of the decision taken under scenarios indistinguishable at the precedent time $t-1$. We avoided to write down in a separate way the special case of time $t = 0$, for which decisions are the same in every possible scenario. This is also consistent with the interpretation in the working paper of Brandimarte et al. (2020), where the ATO problem is seen as a portfolio allocation problem, with components which play the role of assets: in fact, it is a well-known fact in discrete-time finance that portfolio weights must be predictable. The variables related to the assembly of final items (y's) are instead adapted to the information of demands for end items, so that at the stage t, $y_{i,t}^s$ must be the same of the decision taken under scenarios indistinguishable at the time t itself. The rest of the model is quite standard:

- the objective function in Eq. (52) is the sample expected value of the revenues generated by the sales of end items minus the costs of ordering (or making) of components and the costs of keeping an inventory;

- constraints in Eq. (53) and (54) are almost identical to the recourse model of section 3, but must be satisfied for all possible time instants;
- constraints in Eq. (55) is the constraint that links end items sold with components' production and inventory of the beginning and end of the period; it can also be read as the evolution of the inventory, which is the only state variable of the model (x's and y's are proper decision variables).

This model is presented in its split-variable formulation. This formulation is the compact formulation. What are the differences between the two approaches?

1. in the *split-variable* formulation, the tree structure is maintained only in the scenario generation, while we have, for each time step, a number of variables equal to the number of leaves of the tree. Then, non-anticipativity constraints are written such that decisions at stage t must be equal for scenarios which have the same history up to stage t (for adapted decisions) or $t-1$ (for predictable decisions). This model has, therefore, a huge amount of repeated variables which are forced to be equal and this is why this model is really slow; however, the advantage is its simplicity in the implementation and probably its interpretation ease.
2. in the *compact* formulation, the tree structure is maintained for both demands and decisions, so at each node corresponds a decision (in our case the decision concerns both sales for all items and production plan for subsequent periods). Here, non-anticipativity constraints are incorporated in the structure of the decisions themselves, so that a greater effort in the variables' definition leads to a lesser amount of the number of both variables and constraints, which in turn usually means a faster solving step. The difference between adaptation and predictability can be noticed (and we will stress this fact when analysing the code) because predictable decisions start at the root of the tree, while adapted ones begin at the first branch.

In order to introduce formally the compact formulation, we will denote:

1. with A we denote the entire set of all nodes. It can be expressed as the union between the pairwise disjoint sets $N_0, \ldots, N_T$ which contains the nodes related to stages $0, \ldots, T-1$ respectively. $N_0 = n_0$ contains the root of the tree, corresponding to the time instant 0;
2. with the subscript $a(n)$ we will identify the (unique) predecessor at node n;
3. finally, the vector π becomes now a vector of length $T+1$ where we record the probability at each time step to arrive in a node (we are under monte-carlo sampling, so that within an information stage all nodes have the same probability).

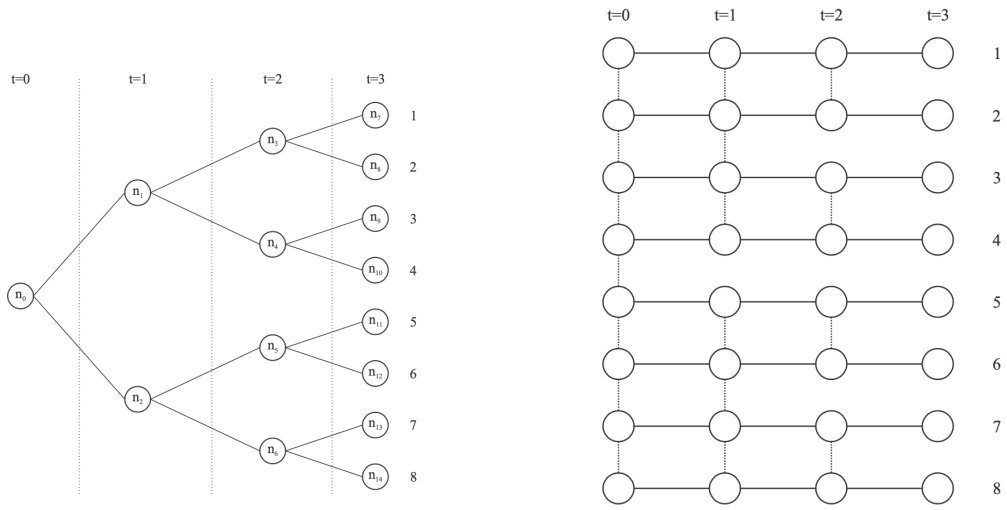

Figure 11: Compact variable formulation: at each node corresponds a variable.

So we have that $a(n_0) = \emptyset$. Then the model can be written as:

$$\max \quad -\sum_{t=0}^{T-1}\sum_{i=1}^{I} C_{i,t} \sum_{n\in N_t} \pi^t x_{i,n} + \sum_{t=1}^{T}\sum_{j=1}^{J} P_{j,t} \sum_{n\in}^{S} \pi^t y_{j,n} \tag{58}$$

$$\text{s.t.} \quad \sum_{i=1}^{I} T_{i,m} x_{i,n} \leq L_{m,t} \qquad \forall m,\ 0 \leq t \leq T-1,\ n \in N_t \tag{59}$$

$$y_{j,n} \leq d_{j,n} \qquad \forall j,\ t \geq 1,\ n \in N_t \tag{60}$$

$$\sum_{j=1}^{J} G_{i,j} y_{j,n} = x_{i,a(n)} + I_{i,a(n)} - I_{i,n} \qquad \forall i,\ t \geq 1,\ n \in N_t \tag{61}$$

$$y_{j,n}, I_{i,n} \qquad \forall j,\ t \geq 1,\ n \in N_t \tag{62}$$

$$x_{i,n} \geq 0 \qquad \forall i,\ 0 \leq t \leq T-1,\ n \in N_t \tag{63}$$

Another possibility is to write down a model without explicitly defining an inventory: this state variable is dependent on a recursion on time of the difference between what has been produced and what have been assembled. In formulas, if we start from $t = 0$ with an inventory I_0, and for the first time we need to assembly it must be the case that

$$\sum_{j=1}^{J} G_{i,j} y_{j,1} \leq x_{i,0} + I_{i,0} \tag{64}$$

We truncated the apex for the scenario for the sake of simplicity of the reasoning, but there is one of such constraint for every scenario. Basically, this is the constraint in Eq. (55), where we have omitted the positive variable with negative sign

$I_{i,1}$, so we can treat

$$x_{i,0} + I_{i,0} - \sum_{j=1}^{J} G_{i,j} y_{j,1}$$

as the inventory kept for the $t = 1$ stage and use this in the same way as I_0 in Eq. (64). So we have

$$\sum_{j=1}^{J} G_{i,j} y_{j,2} \leq x_{i,1} + x_{i,0} + I_{i,0} - \sum_{j=1}^{J} G_{i,j} y_{j,1}$$

which can be rewritten, putting similar terms together, as

$$\sum_{j=1}^{J} G_{i,j} \sum_{t=1}^{2} y_{j,t} \leq I_{i,0} + \sum_{t=0}^{1} x_{i,t} \tag{65}$$

Repeating inductively the same reasoning for a generic time t, we end up with the following constraint in substitution of constraint in Eq. (55) of the multistage problem:

$$\sum_{j=1}^{J} G_{i,j} \sum_{u=1}^{t} y_{j,u}^{s} \leq I_{i,0} + \sum_{u=0}^{t-1} x_{i,u}^{s} \quad \forall\, t,\ s,\ i \tag{66}$$

This implicit scheme for the inventory is slightly more complicated in the compact formulation

$$\sum_{j=1}^{J} G_{i,j} \sum_{u=1}^{t} y_{j,a^{t-u}(n)} \leq I_{i,0} + \sum_{u=0}^{t-1} x_{i,a^{t-u}(n)} \quad \forall\, t \geq 1,\ i, n \in N_t \tag{67}$$

where we used the notation of $a^i(n)$ to indicate the i-th predecessor of node n. In this way we can highlight where the computational burden came from: in fact it is the inventory which links all the stages together and does not allow for a decomposition in simpler two-stage problems.

6.2 Multistage Decision Rules models

In the Chapter 3 we have presented the decision rules approach for the two-stage problem, in which we have found that computational times were worse than classical two-stage recourse model. The advantage of such models came when classical recourse formulation begins its burden: the multistage setting is exactly the case. The approach to build the models is exactly as the one has already taken for the two-stage case: we take every recourse variable and we linearize (possibly into two pieces) with respect to movements in the demands. There are two main interesting points to discuss:

1. the fact that now we have also components making as recourse variable after the initial time, so we linearize also this kind of decision;

2. in the two-stage models there was a demand in the second stage ($t = 1$), while now we have different demands at each time stage, so which one is the risk factor we want to linearize about? The answer is that the general model will consider as risk factors all the demands up to the time considered included (for adapted decisions) or excluded (predictable decisions), but we also explore a version in which only the last relevant demand is taken into account as a risk factor.

6.2.1 Linear Decision Rules

Even if we have seen a greater degree of sub-optimality, to be consistent with the previous work on the two-stage models, we have built the linear decision rules multistage model. For clarity we present a version with explicit inventory, but, exactly as stated for the classical recourse multistage version, there is the possibility to work without it. Before showing the formal model, we state the new notation:

- with $\bar{x}_{it}$ we denote how many pieces of component i the firm produces at time t in the mean case of demand;
- with $\bar{y}_{jt}$ we denote how many pieces of item j the firm sells at time t in the mean case of demand;
- with X_{itu} we denote how many pieces of component i the firm produces at time t if demand for it at time u differs of a piece with respect to the mean;
- with Y_{jktu} we denote how many pieces of item j the firm sells at time t if demand for item k at time u differs of a piece with respect to the mean;
- with I^s_{it} we denote the number of pieces of component i which remains in the inventory after demand occurrence at time t. Please note that this is a state variable and not a decision one, so it has not been linearized;
- for the sake of brevity, in this model, with a little abuse of notation, we use the symbol d^s_{jt} not for the demand for item j at time t in scenario s, but for the deviation from its mean value (as if we have centred demands data in 0).

We have mentioned demand for a component, but the firm only sells items. This is due to the fact that, since it is involved a linear transformation (guided by the gozinto matrix) from the items to the components, we can calculate the effective demand for a component by left-multiply a demand realization for the products by the gozinto matrix. So, for a generic demand realization $d^s = (d^s_1, \ldots, d^s_J)$, we can find the corresponding demand realization for the components

$$c^s = G' d^s,$$

where the $'$ is used to consider the transpose of matrix G. In the same way we can calculate the mean demand for each component and then having the shifts with respect to the mean, so we will use, with a slight abuse of notation c^s_t for

general movements with respect to the mean $\bar{c}_t = G'\bar{d}_t$ and $c^{+s}_{i,t}$ and $c^{-s}_{i,t}$ for positive and negative movements. Finally, all the models presented have a maximum of 4 indexes. This is because, in the most general setting, we allow for adjustments with respect to earlier instants movements in the demand (as it can be seen in the fact that we have $Y_{j,k,t,u}$ with $u \leq t$), but we propose also a simpler version where adjustment are possible only for the same period for sales and one-step-preceding period for the production plan of the components. In this way, the dimension of the problem in terms of the number of variables, scales linearly with respect to time instants, while in the more general setting, the scaling was quadratic. The impact of the elimination of the second time index seems to be insignificant in our naive setting of inter-stage independent demands. An interesting analysis for a future work could be to analyse the impact of this elimination when demands follow an auto-regressive process or a process with some memory; in our simple case, the inventory seems to be a sufficient information to pass from one stage to the following one.

The linear decision rules model is:

$$\begin{aligned}
\max \quad & -\sum_{t=0}^{T-1}\sum_{i=1}^{I}\Big[C_{it}\big(\bar{x}_{it}+\sum_{u=0}^{t}\sum_{s=1}^{S}\pi^s X_{itu}c^s_{iu}\big)\Big] \\
& +\sum_{t=1}^{T}\sum_{i=1}^{J}\Big[P_{jt}\big(\bar{y}_{jt}+\sum_{u=0}^{t}\sum_{s=1}^{S}\pi^s\sum_{k=1}^{J}Y_{jktu}d^s_{ku}\big)\Big] \qquad (68)\\
& +\sum_{i=1}^{I}\sum_{t=1}^{T}\sum_{s=1}^{S}\pi^s h_{it}I^s_{it} \\
\text{s.t.} \quad & X_{itu}=0, \qquad \forall i,\ j,\ t \quad \forall u \geq t \qquad (69)\\
& Y_{ijtu}=0, \qquad \forall i,\ j,\ t \quad \forall u > t \qquad (70)\\
& \sum_{i=1}^{I}T^t_{im}\Big(\bar{x}_{it}+\sum_{u=0}^{t-1}X_{itu}c^s_{iu}\Big)\leq L^t_m \quad \forall m,\ s,\ t<T \qquad (71)\\
& \bar{y}_{jt}+\Big(\sum_{u=1}^{t}\sum_{k=1}^{J}Y_{jktu}d^s_{ju}\Big)\leq d^s_{jt} \qquad \forall j,\ s\ t\geq 1 \qquad (72)\\
& \sum_{j=1}^{J}G_{ij}\Big[(\bar{y}_{jt}+\big(\sum_{u=1}^{t}\sum_{j=1}^{J}Y_{ijtu}d^s_{ju}\big)\Big]=\bar{x}_{i,t-1}+ \\
& +\sum_{u=0}^{t-1}X_{itu}c^s_{iu}+I^s_{i,t-1}-I^s_{i,t} \qquad \forall i,\ s,\ t\geq 1 \qquad (73)\\
& \bar{y}_{jt}+\sum_{u=1}^{t}\sum_{k=1}^{J}Y_{jktu}d^s_{ku}\geq 0 \qquad \forall j,\ s,\ t\geq 1 \\
& \bar{x}_{i,t}+\sum_{u=0}^{t-1}X_{itu}c^s_{iu}\geq 0 \qquad \forall i,\ s,\ t<T \\
& \bar{y}_{jt},\bar{x}_{i,t}\geq 0 \qquad \forall y,\ i,\ t
\end{aligned}$$

Since the model is quite complicated, we analyze it in detail:

- Eq. (68), divided in 3 lines, is the overall profit, divided respectively in total costs due to components making, revenues generated by the sales of and items and cost of inventory.
- Eq. (69) and (70) are non-anticipativity constraints and are worth a deeper look. With these two lines we simply state that adjustment decisions must not depend on future events, i.e demands which are still unknown at the time of the decision. Here is again evident the difference between the nature of

components and items decisions: components decisions are predictable and in fact adjustments are not possible for the same time ($t > u$), while items decisions are adapted ($t \geq u$);

- Eq. (71), (72) and (73) are the multistage equivalent of Eq. (22), (23) and (24), with obviously the time subscript as extra dimensional for which constraints must be satisfied.

- last three lines are boundary requirement: they simply imply that components made at each time must be a non negative number such as items sold (first two lines) and also mean case must respect non negativity conditions without adjustments.

We stress again that, repeating exactly the same inductive reasoning as in the multistage recourse case, we can get rid of the inventory state variable by writing the generic I_{it}^s as

$$I_{i,0} + \sum_{h=0}^{t-1}(\bar{x}_{i,h} + \sum_{u=1}^{h}\sum_{j=1}^{J} X_{iuh}c_{ih}^s) - \sum_{j=1}^{J} G_{i,j} \sum_{h=1}^{t}(\bar{y}_{j,h} + \sum_{u=1}^{h}\sum_{k=1}^{J} Y_{jkuh}d_{kh}^s)$$

6.2.2 Deflected Linear Decision Rules

In this subsection, we briefly see another decision rules models, where there is a greater elasticity and therefore (at least) in sample better performances with respect to the model we have just presented. The differences in notation between this model and the previous one concern only the adjustments and relative movements of the demand. In particular

- we have now X_{itu}^+ and X_{itu}^- and they represent how many additional pieces (with respect to the mean case) of component i the firm produces at time t if demand for the same component i at time u differs of one piece respectively positively and negatively with respect to the mean;

- we have now Y_{jktu}^+ and Y_{jktu}^- and they represent how many additional pieces (with respect to the mean case) of item j the firm sells at time t if demand for item k at time u differs of one piece respectively positively and negatively with respect to the mean;

- finally we came back to the aforementioned notation of d_{jt}^{+s}, d_{jt}^{-s}, c_{it}^{+s} and c_{it}^{-s} for positive and negative deviations with respect to mean demand for item j or component i at time t.

The model then becomes

$$
\begin{aligned}
\max \quad & -\sum_{t=0}^{T-1}\sum_{i=1}^{I}\Big[C_{it}\big(\bar{x}_{it}+\sum_{u=0}^{t}\sum_{s=1}^{S}\pi^s X^{+}_{itu}c^{+s}_{iu}+X^{-}_{itu}c^{-s}_{iu}\big)\Big] \\
& +\sum_{t=1}^{T}\sum_{i=1}^{J}\Big[P_{jt}\big(\bar{y}_{jt}+\sum_{u=0}^{t}\sum_{s=1}^{S}\pi^s\sum_{k=1}^{J}Y^{+}_{jktu}d^{+s}_{ku}+Y^{-}_{jktu}d^{-s}_{ku}\big)\Big] \qquad (74)\\
& +\sum_{i=1}^{I}\sum_{t=1}^{T}\sum_{s=1}^{S}\pi^s h_{it}I^{s}_{it} \\
\text{s.t.} \quad & X^{+}_{itu}=0,\ X^{-}_{itu}=0 \qquad \forall i,\ t \quad \forall u\geq t \qquad (75)\\
& Y^{+}_{jktu}=0,\ Y^{-}_{jktu}=0 \qquad \forall j,\ k,\ t \quad \forall u>t \qquad (76)\\
& \sum_{i=1}^{I}T^{t}_{im}\Big(\bar{x}_{it}+\sum_{u=0}^{t-1}X^{+}_{itu}c^{+s}_{iu}+X^{-}_{itu}c^{-s}_{iu}\Big)\leq L^{t}_{m} \quad \forall m,\ s,\ t \qquad (77)\\
& \bar{y}_{jt}+\Big(\sum_{u=1}^{t}\sum_{k=1}^{J}Y^{+}_{jktu}d^{+s}_{ku}+Y^{-}_{jktu}d^{-s}_{ku}\Big)\leq d^{s}_{jt} \qquad \forall j,\ s\ t \qquad (78)\\
& \sum_{j=1}^{J}G_{ij}\Big[(\bar{y}_{jt}+\big(\sum_{u=1}^{t}\sum_{k=1}^{J}Y^{+}_{jktu}d^{+s}_{ku}+Y^{-}_{jktu}d^{-s}_{ku}\big)\Big]=\bar{x}_{i,t-1}+ \\
& +\sum_{u=0}^{t-1}X^{+}_{itu}c^{+s}_{iu}+X^{-}_{itu}c^{-s}_{iu}+I^{s}_{i,t-1}-I^{s}_{i,t} \qquad \forall i,\ s,\ t\geq 1 \qquad (79)\\
& \bar{y}_{jt}+\Big(\sum_{u=1}^{t}\sum_{k=1}^{J}Y^{+}_{jktu}d^{+s}_{ku}+Y^{-}_{jktu}d^{-s}_{ku}\Big)\geq 0 \qquad \forall j,\ s,\ t \\
& \bar{x}_{i,t}+\Big(\sum_{u=0}^{t-1}X^{+}_{itu}c^{+s}_{iu}+X^{-}_{itu}c^{-s}_{iu}\Big)\geq 0 \qquad \forall i,\ s,\ t \\
& \bar{y}_{jt},\bar{x}_{i,t}\geq 0 \qquad \forall y,\ i,\ t
\end{aligned}
$$

All comments we have done to the previous model are still valid for this one, where the only difference is a possibly better adjustment due to the double number of adjustments coefficients. The last remark we do is that, exactly as in the two-stage problems we have already discussed, by defining the same metric for the dissimilarity of the end-items, it is possible to force the adjustment to be set to 0 for items which pairwise do not have common components (or have a few) for every time $u \leq t$. The regularization for the adjustments of production plans after the first time is easier: since we are aggregating using the gozinto matrix, we consider only movements of demand for items which are built using the component we are considering.

6.3 A short dive in the code: multi-stage models

Exactly as the two-stage models, we have built one class for each new model in the multistage setting. We have built, for the classical recourse version of the problem 4 classes, the split-variable formulation, with explicit and implicit inventory, and the compact formulation, again with explicit and implicit inventory. In this way, we have been able to assess the importance of carefully design the mathematical model in order to avoid useless computational overload. The difference between the compact and the split-variable formulations are clearly present in the initialisation of the adjustment variables As it can be seen in the Listing 7, with the help of the tree structure for demand, we calculate with `n_scen_y` and `n_scen_x` the correct number of decisions at each information stage and then we use two dictionaries to save the variables. Again we stress the difference between the nature of

Listing 7 Compact formulation variables init

```
class AtoMultistageCompactNoInv():
    def solve(
        self, dict_data, time_limit=None,
        gap=None, verbose=False
    ):
        ...
        array_scenarios = dict_data['scenario_tree']
        n_scen_y = np.cumprod(dict_data['scenario_tree'])
        n_scen_x = np.ones(T)
        n_scen_x[1:T] = np.cumprod(dict_data['scenario_tree'][0:T - 1])
        ...
        X_var = {}
        for t in times:
            X_var[t] = model.addVars(
                dict_data['n_components'], int(n_scen_x[t]),
                vtype=self.var_type,
                name=f'X_var_{t}',
                lb=0.0
            )
        Y_var = {}
        for t in range(T):
            Y_var[t+1] = model.addVars(
                dict_data['n_items'], int(n_scen_y[t]),
                vtype=self.var_type,
                name=f'Y_var_{t+1}',
                lb=0.0
            )
```

making and sales decisions: the former begins with a single decision to be taken whichever scenario, the latter begins with a number equal to the first branching factor. The alternative of split-variable in Listing 8 is to obtain a big matrix of `n_items x n_scenarios x n_times` and then apply non-anticipativity constraints by equalizing decisions which are non-distinguishable up to the time considered.

Listing 8 Split-variable formulation variables init

```
class AtoMultistageNoInv():
    def solve(
        self, dict_data, time_limit=None,
        gap=None, verbose=False
    ):
        ...
        array_scenarios = dict_data['scenario_tree']
        n_scenarios = np.prod(array_scenarios)
        ...
        X = model.addVars(
            dict_data['n_components'], n_scenarios, T,
            vtype=self.var_type,
            name='X',
            lb=0.0
        )

        Y = model.addVars(
            dict_data['n_items'], n_scenarios, T,
            vtype=self.var_type,
            name='Y',
            lb=0.0
        )
        ...
        # Non anticipative constraints for making of components
        for i in components:
            n_groups = 1
            for (t, t_scen) in enumerate(array_scenarios):
                n_equal_dec = int(n_scenarios / n_groups)
                for h in range(n_groups):
                    for k in range(n_equal_dec):
                        model.addConstr(
                            X[i, k + h*n_equal_dec, t]==X[i, h*n_equal_dec, t]
                        )
                n_groups = int(n_groups * t_scen)
        # Non anticipative constraints for assembly final products
        for j in items:
            n_groups = 1
            for (t, t_scen) in enumerate(array_scenarios):
                n_groups = int(n_groups*t_scen)
                n_equal_dec = int(n_scenarios / n_groups)
                for h in range(n_groups):
                    for k in range(n_equal_dec):
                        model.addConstr(
                            Y[j, k + h*n_equal_dec, t]==Y[j, h*n_equal_dec, t]
                        )
```

This is achieved by calculating the number of different decisions at each time stage (`n_groups`) and for each one of these, how many scenarios share the same decision (`n_equal_dec`), i.e how many scenarios are equal up to the point considered. The difference between making and sales can be seen in the delayed update (for the making decisions) of the number of different decision to be made. So, for example, for a medium size tree of depth 4, with decreasing branching factor like `array_scenarios = [30, 10, 5, 5]` , we have for the compact formulation, for each end-item we have a total of $30 + 300 + 1500 + 7500 = 9330$ variables, against the $7500 \times 4 = 30000$ of the split-variable formulation. Similarly, for the components making, the difference is between $1 + 30 + 300 + 1500 = 1831$ variables for each component of the compact formulation against the $1500 \times 4 = 6000$ variables of the split-variable model.

Listing 9 Example of difference between formulations of multistage-DLDR.

```
class AtoMultistageDLDRNoInv():
    X_plus = model.addVars(
        dict_data['n_components'], dict_data['n_items'], T, T,
        vtype=self.var_type,
        name='X_plus',
        lb=-GRB.INFINITY
    )
    for t in times:
        for j in items:
            for u in range(t, T):
                for i in components:
                    model.addConstr(
                        X_plus[i, j, t, u] == 0,
                        "Non_antic_X_plus_{}_{}_t_{}_u_{}".format(i, j, t, u)
                    )
    ...
class AtoMultistageDLDRCompactNoInv():
    X_plus = {}
    for t in times:
        if t > 0:
            X_plus[t] = model.addVars(
                dict_data['n_components'], dict_data['n_items'], t,
                vtype=self.var_type,
                name=f'X_plus_{t}',
                lb=-GRB.INFINITY
            )
```

This behaviour becomes huge when time intervals increase in number and when longer horizon branching factor becomes larger and even in this moderate size example, the difference in computational speed is about a magnitude order in

the advantage of the compact formulation. Finally, the explicit or implicit definition of the inventory, in this case, is really important: with the explicit version, we double the number of our variable (the inventory has the same order of magnitude of sales decisions), so for very deep trees or for trees with moderate deep and big branching factors, this really slows down the solver. As we can see in a while, for decision rules models this may not be the case.

Concerning the decision rules models, exactly as in the two-stage case, we present the more complex deflected (or piece-wise) linear rules model. For this model also we have a version with explicit and implicit inventory, and we can talk, with a little abuse of notation, about normal and compact formulation. In fact, we can smartly avoid to produce a good amount of useless variables and not using the same amount of (simple) constraints, as we can see from the Listing 9.

In fact, the choice is between producing 4-d fixed-dimension adjustment decisions (with subscript for the components, the risk factor causing the movements, the time of the adjustment and to which time is related the risk factor) or to produce a 3-d dynamically dimensioned variables which will be located in a dictionary where the time of adjustment is the key (other 3 subscript remains in the variable). By dynamical dimensioning, it is possible to avoid to define the adjustments related to future demands and so we do not have to write explicitly non-anticipativity constraints which are present in the first formulation. Finally, concerning the explicit versus implicit inventory definition, in this case, there is a trade-off: if not explicitly built, model generation is quite slow, while model solving is relatively fast with respect to the explicit case. Overall, for medium-sized instances, there is a computational gain in solving the problem with explicit inventory and so this model is the one we adopt for our tests.

7 Comparison and results: multi-stage

The structure remains the same also for the multistage framework, so up to now, we have investigated the model formulations and their skeletons, which are populated by means of a multistage instance class and are evaluated by means of a multistage evaluation class as well. In this section we briefly see the difference in this new instance class, then we see how the evaluate class operates and finally we explore quantitative results and insights on the behaviour of our models.

7.1 A short dive in the code: multistage instance generation

We have already talked about how the two-stage instance is generated in great detail and almost the same process happens in this case too, with some differences we now remark. We will tacitly assume from now on that costs, profits, gozinto factors, capacity limitations are constant in time, so for this parameter the generation is exactly equal to the two-stage case and we remind to Chapter 4 for a refreshment. The great novelty with respect to the previous generation method is

Listing 10 Multistage instance: differences in the init method.

```
class InstanceMultistage():
    def __init__(
        self, sim_setting, T, array_scenarios_tree=None,
        init_inventory=None, plot_gozinto_matrix=False, gozinto_style=None
    ):
        self.time_instants = T
        if array_scenarios_tree is not None:
            if len(array_scenarios_tree) < T:
                compl_array=[array_scenarios_tree[len(array_scenarios_tree)-1]]*
                            (T-len(array_scenarios_tree))
                array_scenarios_tree = array_scenarios_tree + compl_array

        self.scenario_tree = None if array_scenarios_tree is None
                                  else array_scenarios_tree
        self.n_scenarios = sim_setting['n_scenarios'] if array_scenarios_tree
                               is None else np.prod(array_scenarios_tree)
        ...
        self.demand = self.generate_demand(
            self.time_instants,
            n_scenarios=self.n_scenarios,
        )
```

obviously related to the demand generation and this is the part we concentrate on, but there are also some minor changes that are worth some extra explanations. As

it is possible to see from the Listing 10, there are two new parameters with respect to the two-stage case:

1. T, which is the number of different time instants in which demands occur, so that the model has $T + 1$ stages;

2. the optional parameter `array_scenarios_tree`, which carries crucial information on the uncertainty nature (and give in some sense an a priori choice for the model to be used).

The second parameter is really important for the understanding of the dynamics of our two families of models: classical stochastic programming and decision rules. In fact, for decision rules, which are implicitly non-anticipative, we do not need to generate a scenario tree to assure a sensible meaning to the problem itself and therefore we can directly sample trajectories for the demands and this avoids the curse of dimensionality in terms of the explosion of the scenario tree. So, when solving the decision rules models, the instance we pass to populate the skeleton has the `array_scenarios_tree` parameter set to `None`. On the contrary, for building the classical recourse program we need to enforce this tree structure and so we pass to the initialisation method an array for the branching factor.
Note that we pass the number of time step separately to be able to generate demand scenarios even if the tree structure is not present and if present, we consider this structure to be possibly incomplete: in that case, last branching factor becomes the branching factor for all subsequent periods.

Listing 11 Multistage instance: demand generation.

```
    def generate_demand(self, T, n_scenarios=-1):
        ...
        size = (self.n_items, T, n_scenarios)
        ...
        if self.scenario_tree is None:
            demand = self._simulate_marginal_distribution(marginal, size)
        else:
            demand = np.zeros(size)
            n_groups = 1
            n_equal_dec = n_scenarios
            for (t, t_scen) in enumerate(self.scenario_tree):
                different_demand = self._simulate_marginal_distribution(marginal,
                                            (self.n_items, t_scen))
                n_equal_dec = int(n_equal_dec / t_scen)
                for j in range(self.n_items):
                    demand[j, t, :] = np.tile(different_demand[j,:].repeat(
                                                    n_equal_dec), n_groups)
                n_groups = int(n_groups * t_scen)
```

In Listing 11 we can appreciate the difference in the demand generation in the multi-stage framework. The first thing to note is that, no matter the scenario tree parameter, the demand is returned as a matrix of size ((`n_items, n_times, n_scenarios`)). This is a choice related to the fact that we wanted to keep separated the data information from the model to use and so we had to choose a uniform return format and this was the only viable option. When simulating trajectories, we operate like in the two-stage setting, with the `simulate_marginal_distribution` method called directly that receives a 3-d size. The most complex part is to transform the tree structure into a matrix format and we achieve this by using information about the tree structure and the combination of two functions coming from the `numpy` package: `tile`, which repeat the structure a desired number of times with `repeat`, which instead repeat each number contiguously a desired number of step. A simple example of how this combination works can be found in Listing 12. These highlighted differences are basically the only details that change from the

Listing 12 An example of combination of tile and repeat.

```
import numpy as np
np.tile(np.array([1,2,3,4]).repeat(2),3)
# result
array([1, 1, 2, 2, 3, 3, 4, 4, 1, 1, 2, 2, 3, 3, 4, 4, 1, 1, 2, 2, 3, 3, 4, 4])
```

two-stage models. Last tiny detail is that, in the treatment of parameters generation in Chapter 4, we have omitted a simple attribute from the `.json` file, which was already present, but we have preferred to introduce now since it is used only by the multistage class: this is the `perc_inventory` voices in the file. This attribute has been kept simple, but there is of course room for generalisation. In this simple form, it is the cost of the inventory for every component expressed as the percentage of the components cost. Our choice to use a percentage over the cost of making is related to the fact that also revenues are built in the same way and this allows to vary this parameter in a meaningful way. After this introduction to the instance class, we can suppose to have generated instances with the components costs, profits, inventory costs, gozinto factor for 10 final products generated by 30 components and we can now concentrate in the mechanism of evaluation and then in a comparison about the results achieved by the various models. We have built this relatively smaller instance because of the large number of tests we wanted to carry out and the difficulties in evaluations of the various solutions, which is the topic of the next section.

7.2 Evaluation of multistage solutions

Multistage stochastic programs are known to be difficult and slow to evaluate. This is because

1. all the decisions except the first stage ones are an adaptation with respect to the scenarios over which the solver optimizes, used to avoid myopic behaviour and so they are not implemented in practice;

2. nevertheless these decisions are hard to find because of non-anticipativity constraints which do not allow for clever scenario decompositions or at least good property of the so-called technological matrix (in terms of sparsity), so computational time in solving the original problem is high.

In these two bullets, there are the main reasons why evaluation is usually slow. To completely understand why this is the case, we now talk about two different types of evaluation:

- the `rolling horizon` evaluation. This is the method one should probably use in case of a problem for which the limited support of the horizon is used to let the model be tractable, but the real horizon is unknown (and possibly infinite). The general procedure in this case is, given a tree structure for demand representation and a model to solve, we optimize it at time $t = 0$ and find first-stage decision that we implement. Then, at $t = 1$ we see the real demand for this time instant and we do the best we can possibly do with sales decisions given the numbers of components we have in our inventories (what first stage decisions have told us) and we keep track of costs for making, revenues and final inventory. Then, at time $t = 1$ we have another sample which follows the initial tree structure, we optimize again and repeat the previous steps and we do this re-optimization until we reach time T. Then we sum all costs and revenues and calculate profits and, if needed, ROI, so at the end, for the evaluation, we have solved T instances of the same problem. The real problem in terms of computational resources is that, apart for the first problem to solve, in which every problem has the same conditions (i.e the same initial inventory), for subsequent times, we have to solve an instance of the problem for each trajectory, so, in the end, we solve a total number of $\texttt{n\_scenarios} \times (T-1) + T$ difficult problems, where the number of scenarios is the number of trajectories over which we want to carry out the evaluation.

- the `shrinking horizon` evaluation. This method is suitable for instances where the time T is the right time to stop and close the firm. In this case, the algorithm for the evaluation is similar to the previous one with a crucial exception. After the first optimization routine, we do not solve the $T + 1$-stages problem, but we decrease the tree depth of one step at the time up to a two-stage tree, so the difficulty scales down because in this setting time is really passing and it is not rolling like before. In the end, we solve the same exact number of problems, but their difficulty scales down rapidly (in some sense, we reverse the exponential growth and take advantage of it, solving problems that are way easier than the precedent ones).

These are the two main possibilities that we have implemented in our architecture and they are general in the sense that they can be applied to both the decision

rules approach as well as to the stochastic recourse one. We now see the exact implementation of the topics we have discussed. In the class (Listing 13) there are

Listing 13 Skeleton of the multi-stage evaluation class.

```
class EvaluateMultistage():
    def __init__(self, time_step, n_scenarios, evaluation_type="rolling"):
    def compare_problems(
        self, prb_inst_list, sim_settings, dldr_via_eur=False
        ):
    def _run_one_realization(
        self, prb_inst_list, data_realization, cv_coeff=None
        ):
    def heuristic_one_realization(self, sol, inst, data_realization):
    def _evaluate_sol(self, sol, data):
    def time_evaluation(self, prb_inst_list, max_t=5):
```

two head methods, one for evaluation purpose (which is `compare_problems`) and one for computational times comparison (the `time_evaluation`). The first method requires a list of tuples (instance, problem), a homogeneous file for settings and a boolean value to understand what to do for the decision rules problems (for further details, see next section). Then, it implements the kind of evaluation (rolling or shrinking) that it is passed into the initialization method. The way it has been done is by means of the `_run_one_realization` method, where problems are solved and most importantly, the instance is updated.

Listing 14 Some details of the evaluation procedure.

```
class InstanceMultistage():
    ...
    def update_info(self, sol=[0], assembly=[0], evaluation_type=None, T= -1):
    ...
        if evaluation_type is "rolling":
            pass
        elif evaluation_type is "shrinking":
            if self.scenario_tree is not None:
                self.scenario_tree = self.scenario_tree[0:self.time_instants-1]
                self.n_scenarios = int(np.prod(self.scenario_tree))
            self.time_instants = self.time_instants-1
    self.initial_inventory = self.initial_inventory + sol -
                        np.matmul(self.gozinto.T, assembly)
    self.demand = self.generate_demand(self.time_instants)
```

It can be seen that the inventory costs are calculated from the initial inven-

Listing 15 Evaluation over a single realziation over time.

```
class EvaluateMultistage():
    ...
    def _run_one_realization(
        self, prb_inst_list, data_realization, cv_coeff=None
        ):
        ...
        for (inst, prb) in prb_inst_list:
            cost_sol, cost_inv, profit = 0, 0, 0
            used_inst = copy.copy(inst)
            for ele in range(self.time_step):
                of, sol, comp_time = prb.solve(dict_data)
                I_0 = dict_data['initial_inventory']
                for i in range(len(sol)):
                    cost_sol += dict_data['costs'][i][0] * (sol[i] + 0.0)
                    cost_inv += dict_data['perc_inventory'] * \
                            dict_data['costs'][i][0] * (I_0[i] + 0.0)
                dict_data['demand'] = data_realization[:, ele]
                new_profit, assembly = self._evaluate_sol(sol, dict_data)
                profit += new_profit
                used_inst.update_info(sol,assembly[:,s],self.evaluation_type)
```

tory of the subsequent time instant: this is because this was the simplest way to separate this cost from the profits made. Then the `evaluate_sol` calculate over a realization what are the optimal sales and finally, with this information and info about first-stage solution, initial inventory is updated.

7.2.1 An alternative approach for Decision Rules

A sub-optimal alternative it is possible for the decision rules approach, which has some drawbacks, but also a great advantage. The idea behind this approach is that, in the decision rules settings we are finding the best policy among the ones in the family we are restricting our attention to, but this policy is, for its own nature, scenario-independent and, in the infinite limit of the sample from which we have generated it, it is optimal. So, one could use not only the first-stage solution, but a complete version of the solution we have from the optimization routine. Of course, it may have little sense to consider the linearized version of the sales, but we can take as a "complete solution" the deflected linear decision for the components making. The overall result is sub-optimal because we are not under the assumption of an infinite large scenario tree and so we have no assurance about the optimality of the rules (which is guaranteed for the scenarios over which we have optimized). Moreover, this is not even the major concern: in fact, we are assured to respect all constraints only in the scenarios we have used to build these rules, but not for

out-of-sample realizations, so we need to find a suitable way to always respect the limitations we have. In particular, for the model we are analysing, the problems are related to:

1. the *capacity constraints*;
2. the *non-negativeness of the components making*.

We have chosen a very simplistic way to assure these two conditions, correcting the decisions which are inconsistent by taking the maximum between each decision and 0, and by reducing all decision by the right multiplicative factor when a violation of the capacity constraints occurred. This is what Listing 16 shows.

Listing 16 Heuristic approach to evaluate a DLDR solution.

```
def heuristic_one_realization(self, sol, inst, data_realization):
    ...
    for i in range(dict_data['n_components']):
        tmp = sol[f'{t}']['X_bar'][i]
        for u in range(t):
            for j in range(dict_data['n_items']):
                tmp += sol[f'{t}']['X_plus'][i, j, u] * d_plus[j, u] + \
                       sol[f'{t}']['X_minus'][i, j, u] * d_minus[j, u]
        if tmp < 0:
            counter_adj += 1
            comp[i] = np.max([tmp, 0.0])
        respect, fraction = self.check_bound(inst, comp)
        if not respect:  # if we break the capacity constraints
            sol[f'{t}']['X_bar'] = sol[f'{t}']['X_bar']/(fraction+0.00001)
            sol[f'{t}']['X_plus'] = sol[f'{t}']['X_plus']/(fraction+0.00001)
            sol[f'{t}']['X_minus'] = sol[f'{t}']['X_minus']/(fraction+0.00001)
            counter_adj += 1
```

From the code, it is clear what we do when the solution imposes a negative production for a component, but the capacity constraints check is done in the `check_bound()` method, where we basically reconstruct how much resources we have used and check if the bound is respected. Please note that this passage needs to be done after the check on the positiveness because otherwise, it may be the case that it was the negative production of a component to let the constraints be met.

7.2.2 Computational times: a brief digression on fairness of comparisons

A subtle point in the discussion is how to evaluate solution coming from very different demand models: classical stochastic programming approaches need a demand tree for non-anticipativity requirements, while the decision rules approaches only

need trajectories. In the tree structure, there is an (exponential) growth of the number of scenarios when time flows and the number of the leaves can be enormous. For example, if we think of a normal distribution and a tree with $T = 6$ time steps and branching factor array of $bf = [40, 20, 5, 5, 5, 5]$ leads to

1. 40 indistinguishable scenarios at the first time;
2. 800 at the second time step;
3. $4000, 20000, 100000, 5000000$ respectively at time $t = 3, 4, 5, 6$

but if we concentrate in how uncertainty is explored within each single time instants, it is clear that there is a bad representation for time $3, 4, 5$ and 6, which may be a feature signaling a scarce interest in what happens in longer-horizon times (since in a multi-stage optimization setting we will re-optimize) or it can be related to the fact that there is a lot more uncertainty also in which will be the future distribution, but it remains that the uncertainty representation is poor. On the other hand, if we simply sample a certain number of trajectories, for example 200, we have a representation with 200 realization at each time instant, which may or may not be sufficient to capture sufficiently well the uncertainty, but it is a constant number. In our settings (10 products and 30 components), the ideal situation would be to have a tree structure with fixed branching factor over which to optimize, but this is not possible, and especially for a tree with depth over 5 time instants, it is really difficult computationally to ask more than a branching factor of 10 after the first 3 or 4 time instants. If this was possible, for sure the multi-stage recourse model, which is the "exact" version would be the logical choice against the decision rules models. Unfortunately, this becomes computationally intractable if we want something meaningful (like first branching factor greater than 50). Another possibility is saying that, since we have a certain number of scenarios for the decision rules, we need to structure the tree such that the number of the scenarios in a middle level of the tree is equal (or approximately equal) to the number of trajectories for the decision rules models. This may seem a good compromise, but, in the end, becomes too biased toward decision rules, and this is because of the exponential growth of the number of scenarios in a tree structure. So, a good decision is to fix a certain amount of time and find the tree structure and number of trajectories with which the models are able to find a solution in the required time (at least the mean time should be lower than the chosen one). This is what should be done when taking real decisions. Unfortunately, stable results can be obtained with configurations which require about 15 minutes, which is a very short time when deciding a firm strategy, but it is still too much to carry out extensive multi-setting evaluation experiments over hundreds of scenarios. For this reason, we have preferred to use a common trick for the recourse model and we have compared it with our decision rules heuristic, avoiding any kind of computational issue.

7.3 A computational trick: two-stage-multi-period model

As we have already pointed out, the multi-stage recourse model is quite slow because of the explosion in the number of scenarios that should be generated by means of a tree structure. This is crucial to guarantee that non-anticipativity constraints are met, but we can avoid a part of this computational burden applying a common trick in the literature. The idea is based on the fact that we build a complicated model to take far-from-now uncertainty into account, but then only first stage solution is kept and implemented, generating, in some sense, a waste in the computational resources we have deployed. So we do enforce non-anticipativity constraints, but only for the first stage decision plan which must be unique. In

Listing 17 Trick to transform the multi-stage model in a multi-period one.

```
class AtoMultistage():
    def solve(
        self, dict_data, multi_period=False, time_limit=None,
        gap=None, verbose=False
    ):
        ...
        if not multi_period:
            for i in components:
                n_groups = 1
                for (t, t_scen) in enumerate(array_scenarios):
                    n_equal_dec = int(n_scenarios / n_groups)
                    for h in range(n_groups):
                        for k in range(n_equal_dec):
                            model.addConstr(
                                X[i, k + h*n_equal_dec, t] == X[i, h*n_equal_dec, t]
                            )
                    n_groups = int(n_groups * t_scen)
            for j in items:
                n_groups = 1
                for (t, t_scen) in enumerate(array_scenarios):
                    n_groups = int(n_groups*t_scen)
                    n_equal_dec = int(n_scenarios / n_groups)
                    for h in range(n_groups):
                        for k in range(n_equal_dec):
                            model.addConstr(
                                Y[j, k + h*n_equal_dec, t] == Y[j, h*n_equal_dec, t]
                            )
        else:
            for i in components:
                for s in scenarios:
                    model.addConstr(X[i, s, 0] == X[i, 0, 0])
```

this way, in the model we are deciding what to do right and then having perfect demand information, the uncertainty revealed not one step at the time but all at once, the model knows everything just after having decided the production plan of the current time.

A possible drawback of this method is that obviously all decisions after time 0 are taken to perfectly match what happens in the trajectory where they are, which is a sort of extremely over-fitted behaviour, but the big benefit is a reduction in the computational times: in fact this model is a two-stage model and off-the-shelf algorithm work very fast with this kind of problems. The question we try to answer with this model is if the over-fitted behaviour means a bad inventory management or if, in the end, it is good enough to generate consistent returns. We stress also that, with this method, we are able to use a scenario generation method based on trajectories and not on the tree structure, which gives the possibility of a fair comparison with the decision rules approach based on the number of scenarios. In Listing 17 we can appreciate how the code for the split-variable formulation of the multi-stage model can be reused, by adding a single boolean value, to produce a two-stage-multi-period-model. If the boolean value is equal to true, we simply eliminate the non-anticipativity constraints for all the variables except for the first production decisions. It is important to note that in the compact formulation we are unable to do so because in that model variables were already dimensioned to consider stage uncertainty revealed one step at the time (`n_equal_dec` was the parameter responsible of the cited behaviour). We now show how are performances of the decision rules heuristic compared to the latter described two-stage-multi-period model. As pointed out before, the comparison is fair because we can use the same exact instance (in terms of demand generation) to solve the problems. For results shown in all the following, we have used $T = 5$ instant over which demand arrives (so 6 time steps in total), again 10 end items and 30 components and 5 machines and we have let the distribution, the number of common and specific components and the profitability varying, in order to gain some insights on the impact of these features. Another key characteristic, we have only run experiments with tightness 1.0: this is because, if capacity limits are tight, there is no need for use a complicated multi-instant framework to tackle the problem, as we hardly satisfy the demand and there is no need to control the inventory properly. If there is extremely high capacity, we do not need intelligent inventory management, as we can produce just in time. For the same reason, we apply an upper bound on inventory, forcing it to be less than a prespecified value for each component (in our case the value was 1500). In the evaluation process, since there is no guarantee to respect the bound after having decided the production, if a components violated the constraint, it was simply discarded, but, experimentally, this was rare, as the number of scenarios with which the solutions have been built, where the condition was enforced, seems to be sufficiently high to guarantee the respect of such constraint with high probability.

In the first plot (Fig. 7.3), we can appreciate a comparison by using a right skewed beta distribution. It can be clearly seen that there is a big difference

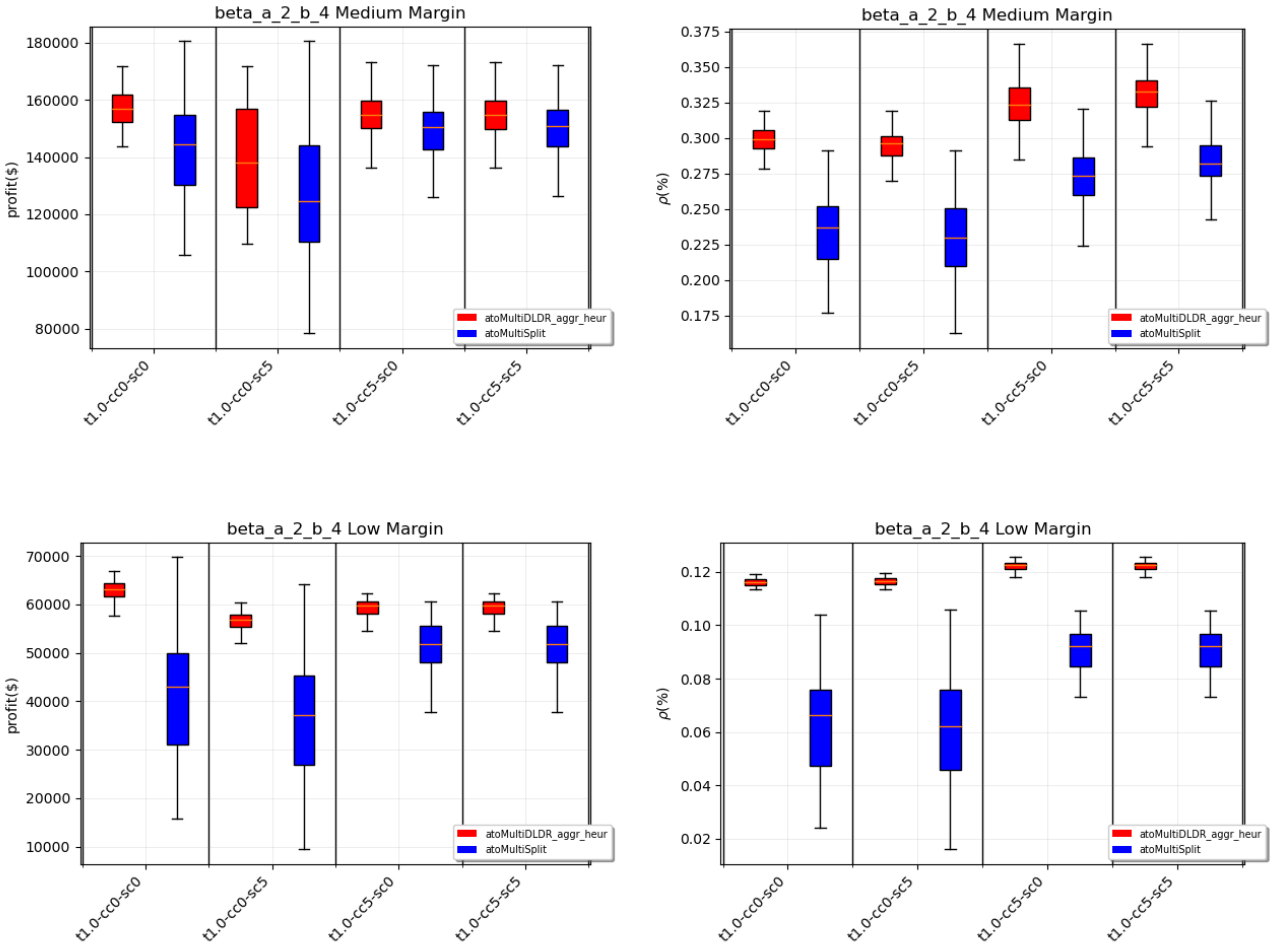

Figure 12: profits (left) and rho (right) for different level of profitability of a right skewed beta distribution with multi-period and heuristic models.

caused by the profitability: in case of low profitability the difference between the two approaches becomes larger, in terms of both absolute profit and roi. In general, with this distribution, the decision rules approach outperforms the multi-period model and the only case in which the difference is tight, at least in terms of profits, is when profitability is large enough and there are common components; this is the best possible setting for our problem, because there are the benefits from an high margin and the risk pooling effect on components demands. In general, we can conclude that the right skewness seems to generate a heavy over-

fitted multi-period model which does not perform well in out-of-sample trajectories. This can be the result of an heavy production plan because the bigger probability of positive events leads to a larger number of scenarios where production after time 0 is higher than usual, which leads to a too optimistic first stage solution.

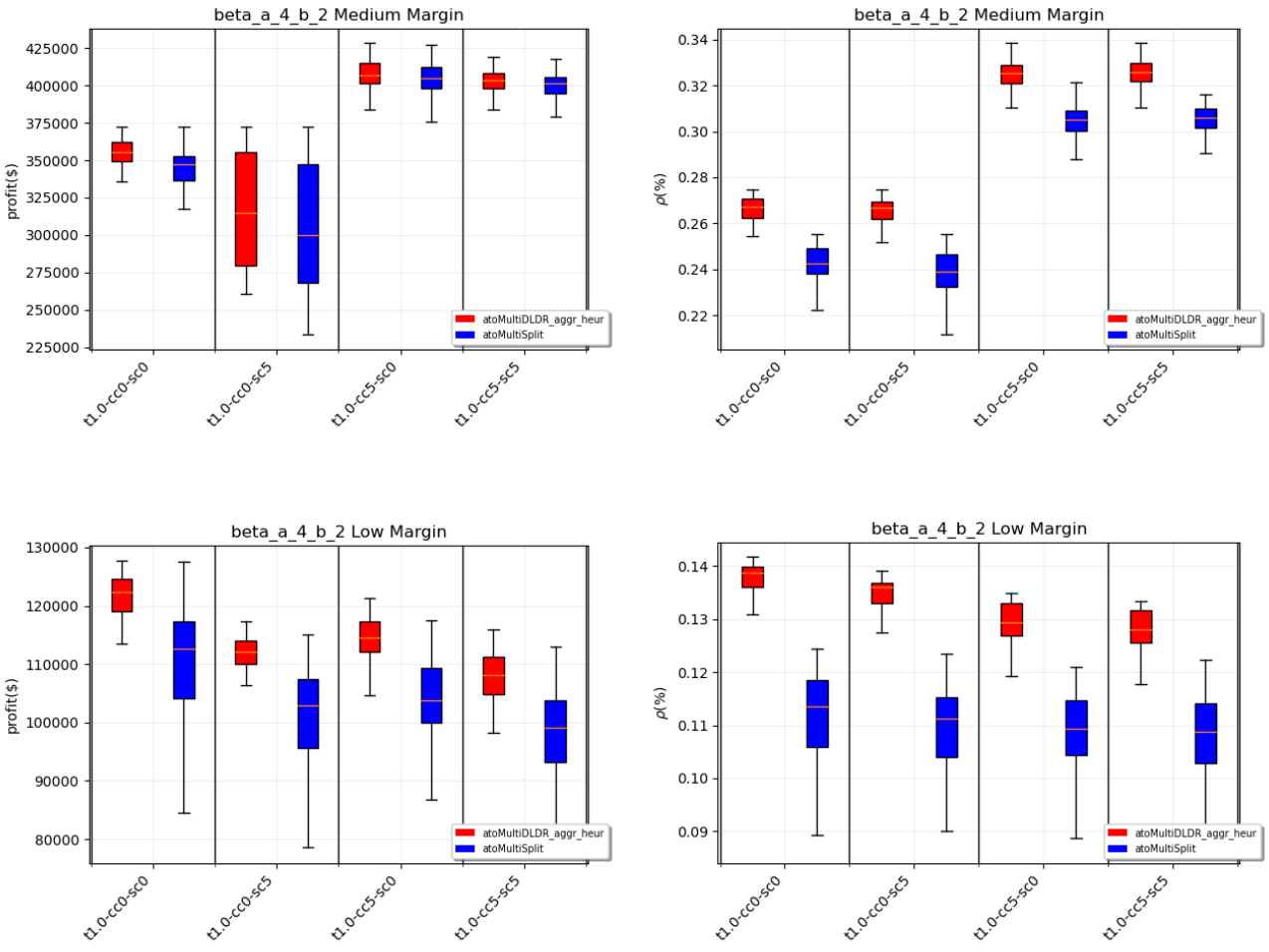

Figure 13: profits (left) and rho (right) for different level of profitability of a left skewed beta distribution with multi-period and heuristic models.

The situation is more fair for the left skewed beta distribution (Fig. 7.3), where the multi-period model behaves better, staying very close to its competitor, at least for absolute profit. This partially confirm the aforementioned effect of the right skewness. In particular, the benefit of the left skewness is evident in the low pro-

fitability case, where profits are close to the heuristic ones in almost every setting of components types. In the end, also for this distributional setting, decision rules approach seems to guarantee a better performance, but the gap is reduced for sure.

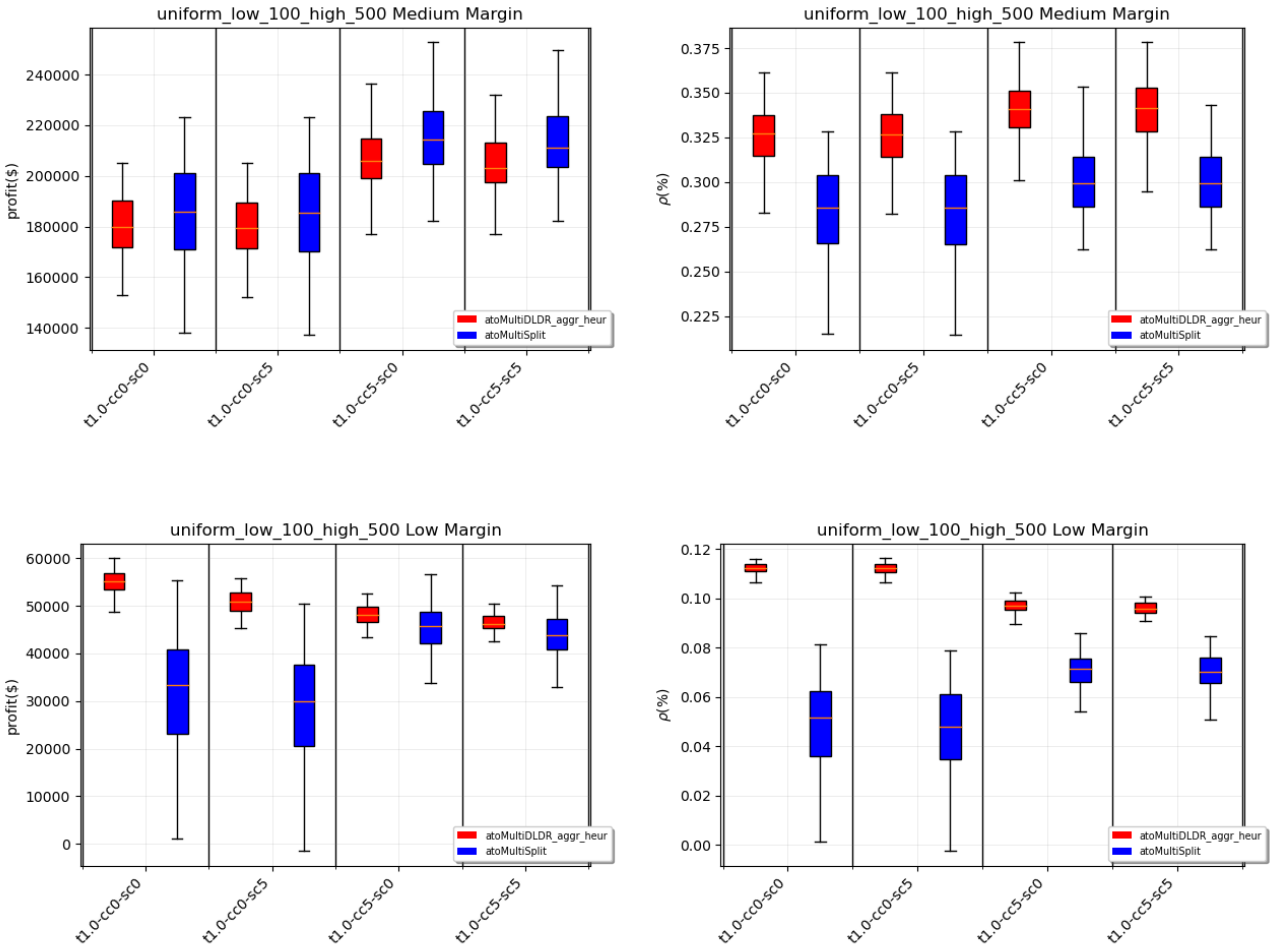

Figure 14: profits (left) and rho (right) for different level of profitability of a uniform distribution (finite and symmetric) with multi-period and heuristic models.

In Fig. 7.3 we can appreciate the symmetric case of a uniform distribution. The insights are extremely interesting; first of all, looking at the upper left picture, we can appreciate how in this case the multi-period model performs slightly better than its competitor, with the variance which is comparable in every setting, also

when there are no common components, where in the previous analysis the variability of multi-period was sufficiently higher. The absolute result is overturned in the ROI picture, where the heuristic has a clear advantage. It is also interesting to note that, in the low profitability case, the comparison between the two method is strongly dependent on the number of common components: if they are present, in absolute terms the two models are basically equivalent, in case of absent of common components the difference is huge as much as in the right skewed case.

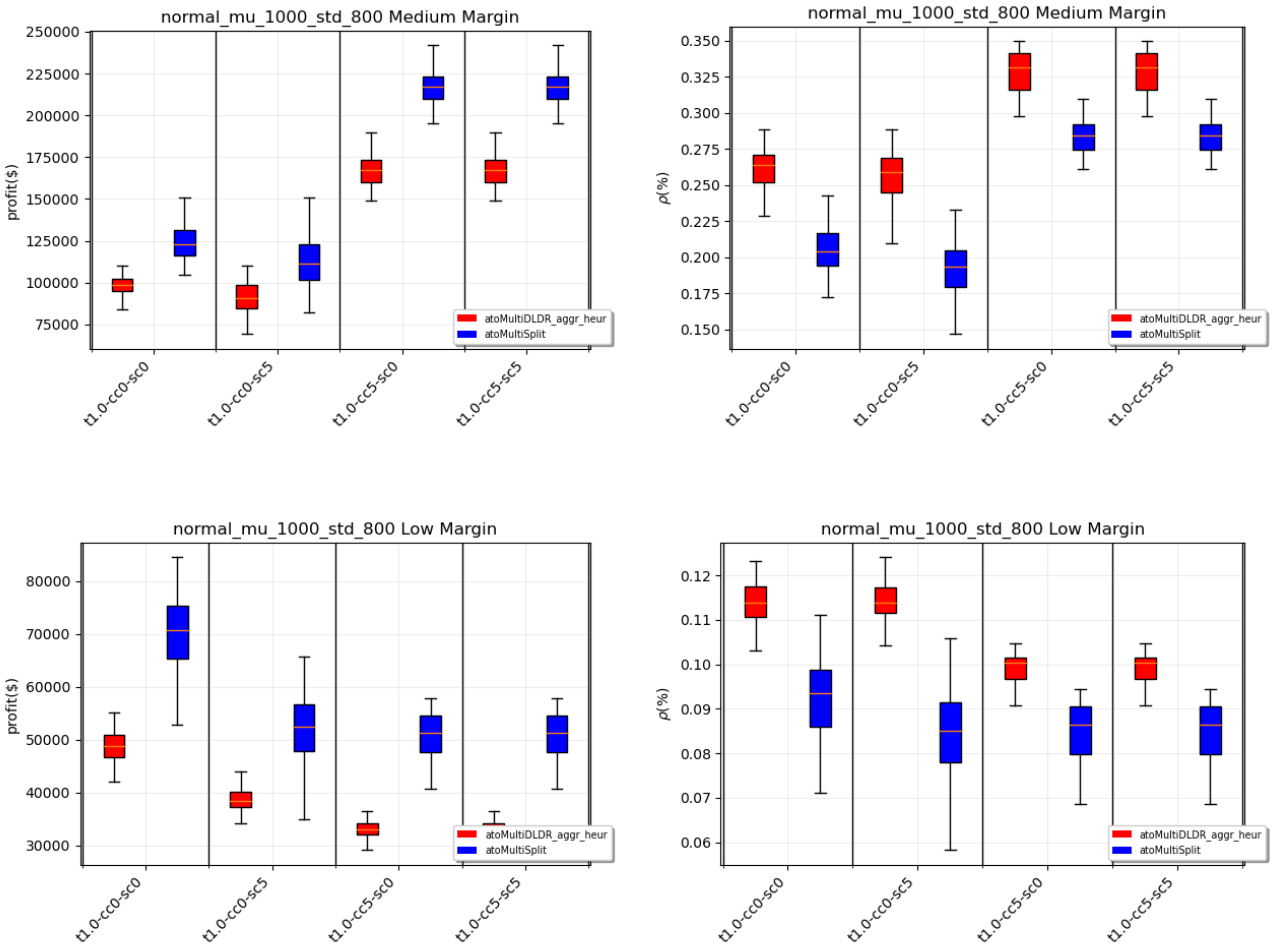

Figure 15: profits (left) and rho (right) for different level of profitability of a normal distribution (infinite and symmetric) with multi-period and heuristic models.

Finally, in the normal distribution case (Fig. 7.3), we see the weakness of

the heuristic approach. As we have already discussed, this is the effect of non-finiteness of the distribution, which lead to a very conservative mean plan and adjustment for the heuristic approach: this is due to the fact that in-sample scenarios must respect demand constraint and capacity constraints and, with a possibility of extremely high demand realization, the mean case shifts down and adjustment must be kept really small. This is a reflection on the really low and tight box-plots for profits and high and tight for the ROI, signaling that trade-off between stability and absolute profit is biased toward the former. This is the distributional case where, no matter the other characteristic, the multi-period model is clearly the best option.

Overall, in all figures, models show a good behaviour in terms of stability except when the number of specific components is high and the number of common ones is low: this is due to the fact that there is no risk pooling effect and profits depend largely on demands which have low correlation. The performances of the decision rules heuristic are better and in some sense they justify the fact that this model requires a little more time to find a solution, time which is then an advantage feature in evaluation mode, where it is necessary to solve only once the problem. By the way, both are able to provide a solid solution in less than 5 minutes with these settings, so computational requirements are very affordable with both models. We stress the fact that in these results, we have deployed the simplified version of the DLDR model, with time dependencies possible only for in time demand realizations for sale decisions and just-realized demands for production decision; this is because, in our simple setting of independent demand, the aggregated information coming from the inventory seems to be sufficient to obtain good performances.

8 Towards the dynamic programming approach

In the previous chapter we have seen how complicated can be the evaluation of a multistage solution, especially in rolling horizon framework, since it involves the repeated solution of a difficult problem with different initial conditions and demand samples. The basic idea behind this chapter will be to develop a meaningful way to adapt the two-stage models into the multi-stage framework, trying to understand if the multi-stage machinery is worth the effort or if a more simplistic approach may be enough to generate consistent returns. There is an obvious problem: in the multistage case, the inventory has a cost (which in our case is expressed as a percentage on the cost of production) but plays a key role because it allows to satisfy demand that would be lost if simply using the just-made products. Inventory is basically the state variable which avoid a myopic behaviour and links every information stage with each other. In the classical two stage models, we can for sure penalize inventory by adding to the inventory cost for the remaining components, but they are useful in subsequent periods and it is less obvious what value to give to these remaining items, a value which is known as *terminal value* or *continuation value* in the literature. The name and the idea behind this term is similar to the dynamic programming approach we have discussed in Chapter 2, where the focus was on finite horizon DP. There, there was a terminal value which was the starting point of the backward induction process and the value function was the key to assign a (economic) value to the state in which we arrive when a decision was taken. In the same way, in the multistage evaluation process we would be able to put together the immediate profits generated by the current sales plus the benefit of having something in the inventory that is a partial insurance against unexpectedly high demand realisation in the future.

Listing 18 Skeleton of continuation value class.

```
1 class ContinuationValue():
2     def __init__(self, inst, n_scen_out=500, n_scen_in=500):
3     ...
4     def solutions_report(self, value=20):
5     ...
```

In order to find this value, which is proper to every different component, we do a reasoning similar to what we have done to find suitable limits for the capacity constraints, i.e we solve the ato problem. However, this time not there will not be a relaxed form: we want to understand the importance of the initial inventory to be able to quantify the value of what passes between a time stage and the subsequent one. The class in Listing 18 is where such values are computed and in its first version the computation is quite simple. The initialisation requires an the instance considered to solve the two-stage model we are interested in and two

number of scenarios: one for solving the ato problem and one for evaluating the solutions. The main method, `solutions_report`, requires a simple value to return the coefficient of the continuation values for each component. The idea is basically to solve $J+1$ ato problem in different condition, where J is the number of products for sale. The differences among this problems is in the boundary conditions: we want to solve our real problem, with no initial inventory and compare the solution against the one found in case of an inventory where we have the exact amount to build k objects of type $j \in J$. This is what is accomplished in listing 19.

Listing 19 Calculation of continuation value coefficients.

```
def solutions_report(self, value=20):
    coeff = np.zeros(self.inst.n_components)
    dict_data = self.inst.get_data()
    prb_rp = Ato(stoch_type="st", var_type="con")
    of, sol_rp, time_rp = prb_rp.solve(dict_data)

    eval_demands = self.inst.generate_demand(self.n_scenarios)
    ev = EvaluateTwostage()
    actual_profits = ev.solve_second_stages(self.inst, sol_rp, self.n_scenarios,
                            eval_demands, False, False)
    mean_profit = np.mean(actual_profits)
    used_inst = copy(self.inst)
    for j in range(used_inst.n_items):
        used_inst.initial_inventory = np.zeros(used_inst.n_components)
        used_inst.initial_inventory += used_inst.gozinto[j, :] * value
        sunk_cost = 0
        for i in range(used_inst.n_components):
            sunk_cost += used_inst.initial_inventory[i]*used_inst.costs[i, 0]
        dict_data = used_inst.get_data()
        of, sol_rp, time_rp = prb_rp.solve(dict_data)
        profits = ev.solve_second_stages(used_inst, sol_rp, self.n_scenarios,
                                eval_demands, False, False)
        adv_prof = np.max([np.mean(profits) - mean_profit - sunk_cost, 0])
        for i in range(used_inst.n_components):
            if used_inst.gozinto[j, i] > 0:
                coeff[i] += adv_prof / (value * used_inst.gozinto[j, i] *
                            sum(used_inst.gozinto[j, :] > 0))
    return 1.1**self.time_instants * coeff
```

After having the solutions, we compare the average profit made without inventory with the one made in one between the J situation and we save the positive difference between the two as the advantage of having the equivalent of k items of type j in the inventory. Then, we divide this number by k, to obtain the gain for a single item and then we divide the merit of the extra profit equally among the

components and, for each component, we still divide for the number of the type of components used to build the product. We repeat this logic for each type of end products. Finally, the coefficient are multiplied by a coefficient build with the time steps necessary to arrive to the horizon T: this is an empirical way to let the two-stage model understand how much time is passed.

After these premises we have a set of coefficient which try to give a value for the components remaining in the inventory at the end of the second stage, so when we are going to evaluate the two-stage models in the multi-stage setting there will be a new part in the objective function that uses these coefficients (see Listing 20).

Listing 20 Changes in the objective function ato class.

```
class Ato():
    ...
    def solve(
        self, dict_data: Dict, cont_value=False, time_limit=None,
        gap=None, qnt_approx=0.8, verbose=False
    ):
    ...
    if cont_value:
        cv_coeff = dict_data["cv_coeff"]
        expr += quicksum(
            pi_s * cv_coeff[i] * (X[i] + I_0[i] - quicksum(
                dict_data['gozinto'][j, i] * Y[j, s]
                for j in items
                )
                )
            for s in scenarios
            for i in components
        )
```

We want to stress the fact that the model's solution in this case tries to be less myopic by giving a positive value to the inventory, which is a non sense in the two-stage framework.

8.1 Comparison between fast methods: continuation value vs DLDR heuristic

We have highlighted the difficulties in using multi-stage models in Chapter 7, so we now propose another strategy to obtain faster results, based on classical two-stage models. This is the continuation value we have just discussed and we compare it with the heuristic approach we have built in the second section of Chapter 7, which we have shown to be superior with respect to the other fast approach based on a multi-period-two-stage model. From a pragmatic point of view, the difference in the evaluation step of the solutions find with the continuation value are in the update of the instance, which is now of type `Instance` and not `InstanceMultistage`, but, apart from that, there are no other differences from the scheme already described in the precedent chapter when we have talked about rolling horizon evaluation (there is no sense in shrinking the two-stage horizon). In computational terms, the fastest method to evaluate is still the heuristic approach, since the comprehensive model needs to be solved only once, whereas the new approach is extremely fast in the single solving (it is a modified version of the classical recourse version of Chapter 3), but it has to be solved for each scenario T time steps. We know try to analyze the impact of common and specific components as well as the distribution type and the profitability, while we keep the tightness parameter fixed to 1.0 in this case: we wanted a situation in which inventory is important but we have limit to what we can produce, so that a meaningful production plan is used. Specifically, as in the previous chapter, results are obtained with 10 products, 30 components, over 5 machines and for 6 time instants (considered also time 0). We know use some plots to gain an insight about the solutions behaviour. In Fig. 16, it can be seen how the approach based on the decision rules maintains, when a non-finite distribution like the normal is used, as already noted in all previous analysis, a conservative behaviour, observable from the tight and low box-plots in terms of profit and tight and high for the ROI; the continuation value model seems instead to be less stable with 500 realization with respect to its competitor, but it is over-performing the heuristics. In terms of common and specific components with the normal distribution a shift in both profit and ROI happens, but the differences between the two approaches remains the same. Finally, profitability does not seems to differenciate the structure of both profits and ROI, but act simply with a vertical shift again.

A more tight comparison can be done when observing the figures related to finite distributions. For example, in Fig. 8.1 we can see that the two models are basically equivalent when a right skewed distribution is used, and the role of number of specific components is more important than common components, as there is high variability when 5 specific components are used. It is interesting to note that this high variability is present in case of a higher profitability, while it disappears when all items generate a low margin. It is also noteworthy to mention that, when profitability is higher there is a substantial equivalence in both profits and ROI, while, when margins are lower, continuation value model has a slightly

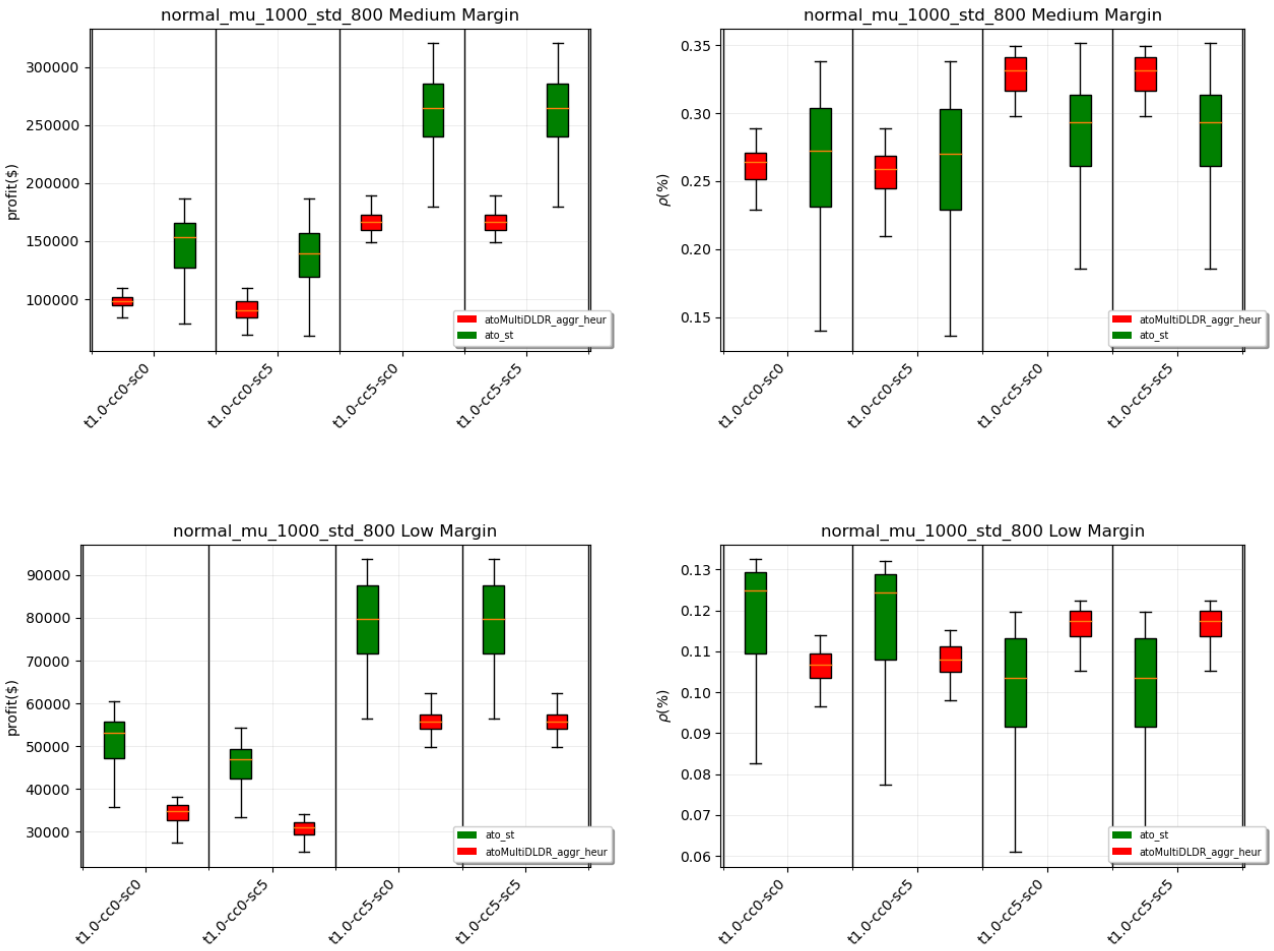

Figure 16: profits (left) and ROI (right) under different profitabilities for a normal distribution under standard (1.0) tightness condition.

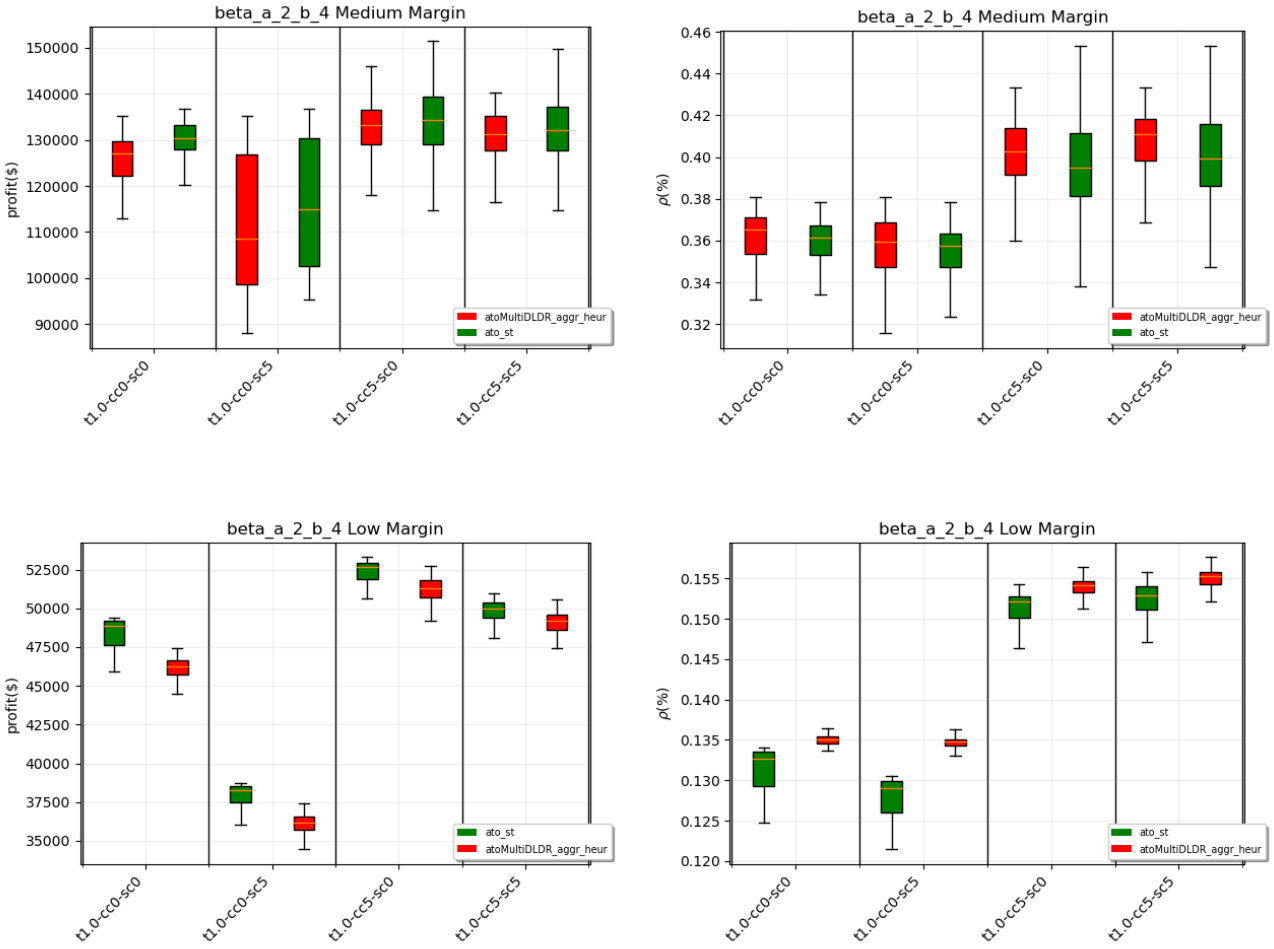

Figure 17: profits (left) and ROI (right) under different profitabilities for a right skewed beta distribution under standard (1.0) tightness condition.

higher absolute profit but a slightly lower ROI, so that it is still difficult to understand what is the best model. We note that, as already seen in the previous chapters, the risk pooling effect of the common components seems to shift both profits and ROI, while the specificity of components impacts on the variability of the solution, as already highlighted before.

In Fig. 8.1 the situation become even more subtle: there is an overall advantage for the heuristic model when margins are higher, since absolute profits are equal but the ROI is higher for the heuristic, but profits become in favor of the con-

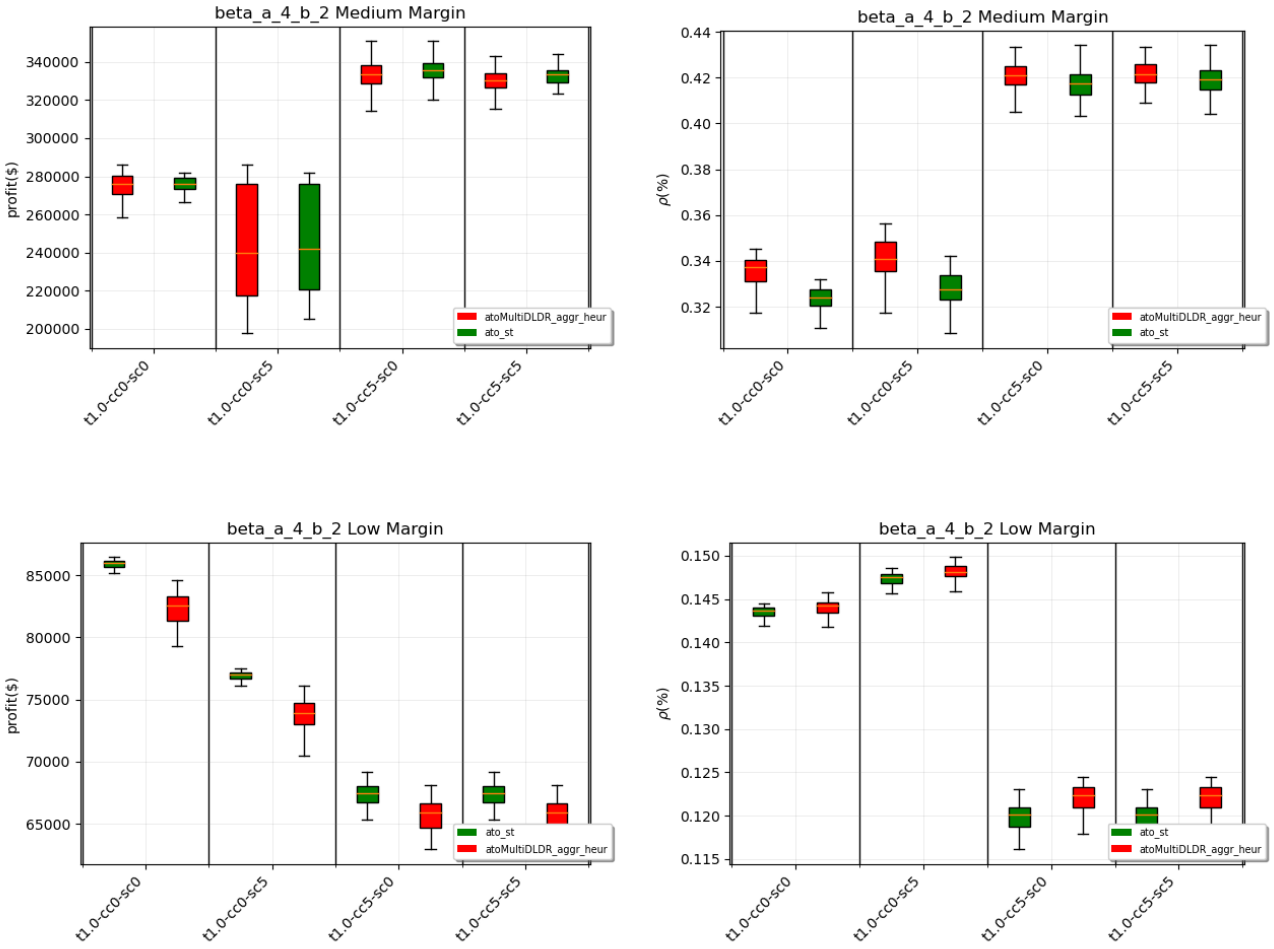

Figure 18: profits (left) and ROI (right) under different profitabilities for a left skewed beta distribution under standard (1.0) tightness condition.

tinuation value model when margins are low. There is not a enormous difference, but from the box-plots we can see that it is significant statistically speaking, so in the latter case the continuation value model seems to outperform its competitor.

Lastly, in Fig. 8.1, when the distribution is symmetric but finite, the two models are again difficult to compare: when medium margins are evaluated, profits are higher for continuation value model, while ROI is lower, but in both cases the difference is subtle. Moreover, in this medium case in terms of profitability, number of specific components is not playing any kind nor in shifting the figures nor

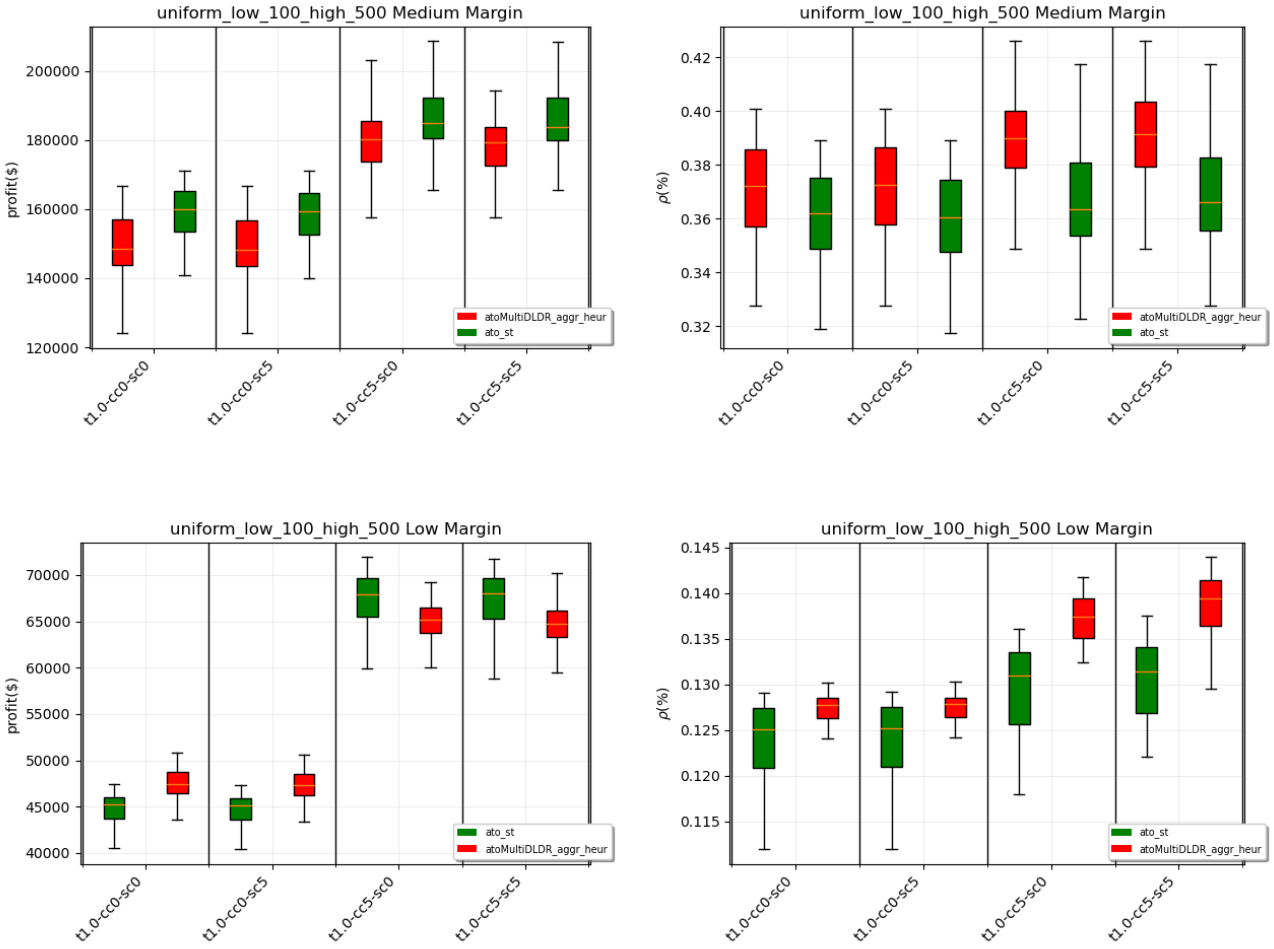

Figure 19: profits (left) and ROI (right) under different profitabilities for a uniform distribution under standard (1.0) tightness condition.

in augmenting the variability, while the number of common components seems to generate higher absolute profits. In the low margin case instead, when there are no common components, heuristic method becomes superior in both profits and ROI, while there is a partial reversion when common components are present, since the continuation value model becomes more profitable even if the ROI is still lower than the one of its competitor. Also in this case, the number of common components has a shifting effect, but in this case this effect can be appreciated both in terms of profits and ROI.

Finally, we want to stress that the continuation value model has been built following a sensitivity analysis on the inventory, an idea based on a value found for one subsequent period and not still seeing a global picture, which has been added by multiplying the coefficient found by a simple function of the time to go. This implies that there are no guarantees about the behaviour of such model when path dependencies of the demand stochastic processes are added, while the heuristic model does not rely directly on this assumption, but we are not sure on its performances as well when more complicated processes are evaluated.

9 Conclusions and further work

In the previous chapters we have tried to answer some questions related to an assembly-to-order system under demand uncertainty and here, we summarize the results. First of all, taking into account uncertainty seems to be crucial, as highlighted by the high VSS when conditions are unfavourable, i.e when there are a lot of specific components or profitability is low. In such conditions not considering uncertainty means basically going out of business.
Secondly, we have shown that perfect information in this kind of problems is highly valuable and again this is particularly true for situations where the risk-pooling effect of common components is scarce: this is because, when there are a lot of common components, the aggregated demand for components has less variability and in some sense, this is equivalent to an augmentation of information for the recourse problem.
In the same chapter, we have then compared the recourse problem against the decision rules approaches, concluding the supremacy of the former when the distribution is normal and in stressing condition for the firm, as for example in case of low profitability for every component. Nonetheless, deflected rules are very similar in the result in a lot of settings, suggesting that the elasticity price of an easily tractable form for the problem is not too high while, forcing a linearity condition seems to lead to extremely conservative plans when not needed. Then, we have briefly discussed about the trade-off between stability and performances for the robust models, showing a similar structure for the decision rules approaches in the non-robust and robust cases. After that, we have shifted our attention to multi-stage models. We have built multi-stage recourse and deflected rules models, which were the building blocks for further evaluation based on them. in particular we have seen how the limited support of the distribution seems to be a determinant factor for assessing the goodness of a heuristic based on the decision rules approach, which is a method sufficiently good to outperform the two-stage multi-period model, a popular way to avoid the computational burden of multistage programs. Both these two methods are fast to solve and can be implemented in minutes in real-sized problems, but in the evaluation phase, there is a preference for the heuristic method, as it requires the problem solution only once. We have finally proposed a two-stage variant with continuation value to build meaningful and not myopic inventory management using a two-stage model in the multi-stage settings. By comparing it to the heuristic approach based on decision rules, we have shown once again the pitfall of the normal distribution case for the decision rules model, but apart from that, the two methods proposed are nearly equivalent in all other settings, suggesting good inventory management in both cases. Overall, the decision rule approach seems to be a valid alternative for finite distributions and medium conditions in terms of profitability and specificity of components. We have explored the multistage setting in the easiest possible case of time-independent demand and the first research theme could be to check whatever our findings for the multistage framework remain valid for more complex stochastic processes. So,

in this work, we claim that, under suitable economical conditions, when classical recourse version of the problem begins to suffer from the curse of dimensionality, decision rules based approach may be a good option, since it does not explode, in terms of the number of variables, exponentially when the number of scenario increases, but only linearly. In addition, there is the possibility of investigating what happens when there is correlation for demand in the same time period, how changes the risk-pooling effect of the common components when another source of interaction is added. Another interesting continuation for the work is to run experiments with real data and possibly using other risk factors, generated by dimensionality reduction techniques from the demand data available: in our case of synthetic data it would be useless, but greater performances in terms of computational requirements are possible if risk factors are kept limited in size. Finally, it would be interesting, when the dimension of the problem grows in terms of the number of components and end products, to build a machine learning approach to understand what are the critical items, with bad characteristic, to be treated scenario by scenario, and what are the simpler products that can be treated by expected value or by decision rules, creating a mix and having a trade-off between tractability and performance.

References

Z. Atan, T. Ahmadi, C. Stegehuis, T. de Kok, and I. Adan. Assemble-to-order systems: A review. *European Journal of Operational Research*, 261:866 – 879, 2017. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2017.02.029. URL https://www.sciencedirect.com/science/article/abs/pii/S0377221717301510.

A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99:351–376, 2004. doi: https://doi.org/10.1007/s10107-003-0454-y.

A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, October 2009.

D. P. Bertsekas. *Dynamic Programming: Models and Applications*, volume I. 4th edition, 2017. ISBN 1-886529-43-4. URL http://athenasc.com/dpbook.html.

P. Brandimarte. *Variance Reduction Methods*, chapter 8, pages 341–377. John Wiley & Sons, Ltd, 2014. ISBN 9781118593264. doi: https://doi.org/10.1002/9781118593264.ch8. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118593264.ch8.

P. Brandimarte, E. Fadda, and A. Gennaro. The value of the stochastic solution in a two-stage assembly-to-order problem. *working paper*, 2020.

G. C. Calafiore. Multi-period portfolio optimization with linear control policies. *Automatica*, 44(10):2463–2473, 2008. URL https://doi.org/10.1016/j.automatica.2008.02.007.

X. Chen, M. Sim, and P. Sun. A robust optimization perspective on stochastic programming. *Operations Research*, 55(6):1058–1071, 2007. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/25147146.

E. V. Denardo. *Dynamic Programming: Models and Applications*. 2013. ISBN 978-0486428109. URL https://www.amazon.com/Dynamic-Programming-Applications-Computer-Science/dp/0486428109.

M. Elhafsi, L. Zhi, H. Camus, and E. Craye. An assemble-to-order system with product and components demand with lost sales. *International Journal of Production Research*, 53(3):718–735, 2015. doi: 10.1080/00207543.2014.920547.

A. Georghiou, D. Kuhn, and W. Wiesemann. Generalized decision rule approximations for stochastic programming via liftings. *Mathematical Programming*, 152:301–338, 2014. doi: http://dx.doi.org/10.1007/s10107-014-0789-6. URL https://dspace.mit.edu/handle/1721.1/103397.

A. Georghiou, D. Kuhn, and W. Wiesemann. The decision rule approach to optimization under uncertainty: methodology and applications. *Computational*

Management Science, 16:545 – 576, 2019. ISSN 1619-6988. doi: https://doi.org/10.1007/s10287-018-0338-5. URL https://link.springer.com/article/10.1007%2Fs10287-018-0338-5.

C. C. Moallemi and M. Sağlam. Dynamic portfolio choice with linear rebalancing rules. *Journal of Financial and Quantitative Analysis*, 52(3):1247–1278, 2017. doi: https://doi.org/10.1017/S0022109017000345.

E. Nadar, M. Akan, and A. Scheller-Wolf. Technical note—optimal structural results for assemble-to-order generalized m-systems. *Operations Research*, 62(3): 571–579, 2014. doi: 10.1287/opre.2014.1271.

M. I. Reiman and Q. Wang. Asymptotically optimal inventory control for assemble-to-order systems with identical lead times. *Operations Research*, 63(3):716–732, 2015. doi: 10.1287/opre.2015.1372.

Appendices

A Tables for two-stage results

Sol.	Perc. low.	cc	sc	Mean	Std	CI lower	CI upper
EV	0.4	0	0	0.06360	0.08293	0.04734	0.07985
RP	0.4	0	0	0.18394	0.06465	0.17127	0.19661
LDRV	0.4	0	0	0.32248	0.00991	0.32054	0.32442
DLDRV	0.4	0	0	0.18842	0.10478	0.16789	0.20896
EV	0.4	0	10	0.05065	0.08062	0.03485	0.06645
RP	0.4	0	10	0.18913	0.05935	0.17749	0.20076
LDRV	0.4	0	10	0.31975	0.00944	0.31790	0.32160
DLDRV	0.4	0	10	0.16660	0.10923	0.14519	0.18800
EV	0.4	10	0	0.27057	0.04882	0.26101	0.28014
RP	0.4	10	0	0.29015	0.03911	0.28249	0.29782
LDRV	0.4	10	0	0.38301	0.02641	0.37783	0.38818
DLDRV	0.4	10	0	0.34304	0.03346	0.33648	0.34960
EV	0.4	10	10	0.31136	0.04466	0.30261	0.32011
RP	0.4	10	10	0.32022	0.04220	0.31195	0.32849
LDRV	0.4	10	10	0.40154	0.03213	0.39524	0.40784
DLDRV	0.4	10	10	0.37700	0.03424	0.37029	0.38371
EV	1	0	0	-0.10938	0.06545	-0.12221	-0.09656
RP	1	0	0	0.06334	0.02813	0.05783	0.06886
LDRV	1	0	0	0.11467	0.01244	0.11223	0.11710
DLDRV	1	0	0	0.11286	0.00811	0.11127	0.11445
EV	1	0	10	-0.09533	0.06312	-0.10771	-0.08296
RP	1	0	10	0.06902	0.03044	0.06305	0.07498
LDRV	1	0	10	0.12813	0.01546	0.12510	0.13116
DLDRV	1	0	10	0.12905	0.01516	0.12608	0.13202
EV	1	10	0	0.11991	0.02862	0.11430	0.12552
RP	1	10	0	0.13401	0.01854	0.13038	0.13764
LDRV	1	10	0	0.17723	0.00450	0.17635	0.17811
DLDRV	1	10	0	0.16198	0.00648	0.16071	0.16325
EV	1	10	10	0.11641	0.02496	0.11151	0.12130
RP	1	10	10	0.12853	0.01912	0.12478	0.13228
LDRV	1	10	10	0.17111	0.00628	0.16988	0.17234
DLDRV	1	10	10	0.15597	0.00736	0.15453	0.15742

Table 1: ROI, normal distribution, tightness set to 0.8.

Sol.	Perc. low.	cc	sc	Mean	Std	CI lower	CI upper
EV	0.4	0	0	0.14007	0.05631	0.12903	0.15110
RP	0.4	0	0	0.21669	0.03444	0.20994	0.22344
LDRV	0.4	0	0	0.27104	0.00019	0.27100	0.27107
DLDRV	0.4	0	0	0.22578	0.04481	0.21699	0.23456
EV	0.4	0	10	0.14114	0.05104	0.13113	0.15114
RP	0.4	0	10	0.22296	0.03250	0.21658	0.22933
LDRV	0.4	0	10	0.28450	0.00028	0.28444	0.28455
DLDRV	0.4	0	10	0.24098	0.03966	0.23321	0.24875
EV	0.4	10	0	0.32593	0.03076	0.31990	0.33196
RP	0.4	10	0	0.33393	0.02586	0.32887	0.33900
LDRV	0.4	10	0	0.34269	0.02318	0.33814	0.34723
DLDRV	0.4	10	0	0.34147	0.02354	0.33685	0.34608
EV	0.4	10	10	0.31194	0.02983	0.30610	0.31779
RP	0.4	10	10	0.32275	0.02443	0.31797	0.32754
LDRV	0.4	10	10	0.32426	0.02457	0.31945	0.32908
DLDRV	0.4	10	10	0.32244	0.02445	0.31764	0.32723
EV	1	0	0	0.00357	0.03981	-0.00423	0.01137
RP	1	0	0	0.09978	0.01203	0.09742	0.10214
LDRV	1	0	0	0.12303	0.00007	0.12302	0.12305
DLDRV	1	0	0	0.12303	0.00007	0.12302	0.12305
EV	1	0	10	-0.00379	0.04113	-0.01185	0.00427
RP	1	0	10	0.10230	0.01621	0.09912	0.10547
LDRV	1	0	10	0.13360	0.00004	0.13359	0.13361
DLDRV	1	0	10	0.13340	0.00012	0.13338	0.13343
EV	1	10	0	0.13773	0.01149	0.13548	0.13999
RP	1	10	0	0.14142	0.00757	0.13994	0.14290
LDRV	1	10	0	0.15414	0.00418	0.15332	0.15496
DLDRV	1	10	0	0.15200	0.00441	0.15113	0.15286
EV	1	10	10	0.13730	0.01329	0.13469	0.13990
RP	1	10	10	0.14312	0.00811	0.14153	0.14472
LDRV	1	10	10	0.15722	0.00438	0.15636	0.15808
DLDRV	1	10	10	0.15380	0.00478	0.15286	0.15473

Table 2: ROI, uniform distribution, tightness set to 0.8.

Sol.	Perc. low.	cc	sc	Mean	Std	CI lower	CI upper
EV	0.4	0	0	0.21523	0.03505	0.20836	0.22210
RP	0.4	0	0	0.25989	0.02028	0.25591	0.26386
LDRV	0.4	0	0	0.29518	0.00056	0.29507	0.29529
DLDRV	0.4	0	0	0.26703	0.02870	0.26141	0.27266
EV	0.4	0	10	0.20534	0.03604	0.19828	0.21241
RP	0.4	0	10	0.25837	0.02009	0.25444	0.26231
LDRV	0.4	0	10	0.29903	0.00110	0.29882	0.29925
DLDRV	0.4	0	10	0.26375	0.02673	0.25851	0.26899
EV	0.4	10	0	0.31289	0.02162	0.30866	0.31713
RP	0.4	10	0	0.31598	0.01801	0.31245	0.31951
LDRV	0.4	10	0	0.31927	0.01687	0.31596	0.32257
DLDRV	0.4	10	0	0.31753	0.01663	0.31427	0.32079
EV	0.4	10	10	0.30424	0.01684	0.30094	0.30754
RP	0.4	10	10	0.31376	0.01553	0.31072	0.31681
LDRV	0.4	10	10	0.32512	0.01514	0.32216	0.32809
DLDRV	0.4	10	10	0.31762	0.01527	0.31463	0.32062
EV	1	0	0	0.02997	0.03128	0.02384	0.03610
RP	1	0	0	0.10947	0.00937	0.10764	0.11131
LDRV	1	0	0	0.12735	0.00015	0.12732	0.12738
DLDRV	1	0	0	0.12728	0.00020	0.12725	0.12732
EV	1	0	10	0.00065	0.03685	-0.00657	0.00787
RP	1	0	10	0.09235	0.00854	0.09068	0.09402
LDRV	1	0	10	0.10584	0.00018	0.10580	0.10587
DLDRV	1	0	10	0.10600	0.00026	0.10595	0.10605
EV	1	10	0	0.14530	0.00710	0.14391	0.14669
RP	1	10	0	0.14665	0.00483	0.14570	0.14759
LDRV	1	10	0	0.15422	0.00353	0.15353	0.15491
DLDRV	1	10	0	0.15114	0.00373	0.15041	0.15187
EV	1	10	10	0.14128	0.00801	0.13971	0.14285
RP	1	10	10	0.14311	0.00521	0.14209	0.14413
LDRV	1	10	10	0.14987	0.00389	0.14911	0.15063
DLDRV	1	10	10	0.14735	0.00400	0.14656	0.14813

Table 3: ROI, right skewed beta distribution, tightness set to 0.8.

Sol.	Perc. low.	cc	sc	Mean	Std	CI lower	CI upper
EV	0.4	0	0	0.24489	0.02650	0.23969	0.25008
RP	0.4	0	0	0.26856	0.01750	0.26513	0.27199
LDRV	0.4	0	0	0.28009	0.01570	0.27702	0.28317
DLDRV	0.4	0	0	0.26520	0.02427	0.26045	0.26996
EV	0.4	0	10	0.23253	0.02618	0.22740	0.23766
RP	0.4	0	10	0.26218	0.01405	0.25943	0.26494
LDRV	0.4	0	10	0.28309	0.01290	0.28056	0.28561
DLDRV	0.4	0	10	0.25366	0.02078	0.24959	0.25773
EV	0.4	10	0	0.34619	0.01037	0.34416	0.34822
RP	0.4	10	0	0.34913	0.01094	0.34698	0.35127
LDRV	0.4	10	0	0.34871	0.01067	0.34662	0.35080
DLDRV	0.4	10	0	0.34895	0.01068	0.34686	0.35104
EV	0.4	10	10	0.33798	0.01180	0.33567	0.34029
RP	0.4	10	10	0.34232	0.01129	0.34011	0.34454
LDRV	0.4	10	10	0.34186	0.01167	0.33957	0.34415
DLDRV	0.4	10	10	0.34217	0.01157	0.33991	0.34444
EV	1	0	0	0.06834	0.02103	0.06422	0.07247
RP	1	0	0	0.10927	0.00993	0.10733	0.11122
LDRV	1	0	0	0.12657	0.00075	0.12642	0.12672
DLDRV	1	0	0	0.11572	0.01223	0.11332	0.11811
EV	1	0	10	0.07268	0.02495	0.06779	0.07757
RP	1	0	10	0.11479	0.01078	0.11268	0.11690
LDRV	1	0	10	0.13005	0.00011	0.13003	0.13007
DLDRV	1	0	10	0.11973	0.01123	0.11753	0.12193
EV	1	10	0	0.13878	0.00471	0.13786	0.13970
RP	1	10	0	0.13908	0.00420	0.13825	0.13990
LDRV	1	10	0	0.14090	0.00373	0.14016	0.14163
DLDRV	1	10	0	0.13897	0.00407	0.13818	0.13977
EV	1	10	10	0.11966	0.00583	0.11851	0.12080
RP	1	10	10	0.12305	0.00267	0.12253	0.12358
LDRV	1	10	10	0.12350	0.00259	0.12299	0.12400
DLDRV	1	10	10	0.12263	0.00260	0.12212	0.12314

Table 4: ROI, left skewed beta distribution, tightness set to 0.8.

Sol.	Perc. low.	cc	sc	Mean	Std	CI lower	CI upper
PerfectInfo	0.4	0	0	577300.84	58510.25	565691.14	588910.54
EV	0.4	0	0	110629.54	144266.58	82003.92	139255.16
RP	0.4	0	0	240768.05	84628.12	223975.99	257560.11
LDRV	0.4	0	0	10108.04	310.56	10046.42	10169.66
DLDRV	0.4	0	0	165579.52	92073.46	147310.15	183848.90
PerfectInfo	0.4	0	10	464189.02	47119.90	454839.41	473538.63
EV	0.4	0	10	71705.31	114133.92	49058.66	94351.95
RP	0.4	0	10	189231.23	59384.46	177448.07	201014.40
LDRV	0.4	0	10	7253.88	214.24	7211.37	7296.39
DLDRV	0.4	0	10	114419.67	75018.69	99534.34	129305.01
PerfectInfo	0.4	10	0	959794.96	90543.70	941829.13	977760.80
EV	0.4	10	0	787997.05	142166.42	759788.15	816205.95
RP	0.4	10	0	839494.31	113145.95	817043.70	861944.92
LDRV	0.4	10	0	607594.91	41890.20	599282.99	615906.84
DLDRV	0.4	10	0	750947.37	73255.94	736411.80	765482.93
PerfectInfo	0.4	10	10	970548.56	89718.07	952746.55	988350.57
EV	0.4	10	10	847250.46	121530.21	823136.23	871364.69
RP	0.4	10	10	865406.18	114058.29	842774.54	888037.82
LDRV	0.4	10	10	717634.46	57424.04	706240.29	729028.64
DLDRV	0.4	10	10	790803.21	71817.88	776552.98	805053.43
PerfectInfo	1	0	0	169492.01	12141.99	167082.77	171901.24
EV	1	0	0	-158798.74	95016.51	-177652.08	-139945.41
RP	1	0	0	34887.07	15495.28	31812.47	37961.67
LDRV	1	0	0	509.64	55.29	498.67	520.61
DLDRV	1	0	0	972.96	69.94	959.08	986.84
PerfectInfo	1	0	10	184231.50	12361.18	181778.77	186684.23
EV	1	0	10	-135944.53	90011.14	-153804.70	-118084.37
RP	1	0	10	37116.44	16367.64	33868.75	40364.14
LDRV	1	0	10	1093.46	131.92	1067.28	1119.63
DLDRV	1	0	10	1123.16	131.92	1096.98	1149.33
PerfectInfo	1	10	0	364282.29	22229.53	359871.47	368693.11
EV	1	10	0	296361.67	70733.71	282326.57	310396.78
RP	1	10	0	310989.76	43035.55	302450.57	319528.95
LDRV	1	10	0	79788.13	2026.24	79386.08	80190.18
DLDRV	1	10	0	215514.92	8621.82	213804.16	217225.68
PerfectInfo	1	10	10	294362.33	14860.50	291413.68	297310.97
EV	1	10	10	237596.41	50954.77	227485.88	247706.94
RP	1	10	10	248908.93	37020.21	241563.32	256254.55
LDRV	1	10	10	78215.30	2869.60	77645.91	78784.69
DLDRV	1	10	10	181463.00	8558.77	179764.75	183161.24

Table 5: Profits, normal distribution, tightness set to 0.8.

Sol.	Perc. low.	cc	sc	Mean	Std	CI lower	CI upper
PerfectInfo	0.4	0	0	143574.21	8814.77	141825.17	145323.25
EV	0.4	0	0	65036.31	26144.48	59848.68	70223.94
RP	0.4	0	0	89831.18	14277.67	86998.18	92664.18
LDRV	0.4	0	0	53776.21	37.05	53768.86	53783.56
DLDRV	0.4	0	0	77959.94	15471.28	74890.11	81029.78
PerfectInfo	0.4	0	10	116822.90	6076.20	115617.25	118028.55
EV	0.4	0	10	52010.25	18807.98	48278.34	55742.16
RP	0.4	0	10	73998.48	10788.05	71857.89	76139.06
LDRV	0.4	0	10	44678.80	43.84	44670.10	44687.50
DLDRV	0.4	0	10	64852.69	10673.13	62734.91	66970.47
PerfectInfo	0.4	10	0	256513.46	15864.18	253365.67	259661.26
EV	0.4	10	0	241863.51	22827.16	237334.11	246392.91
RP	0.4	10	0	246307.39	19074.37	242522.62	250092.15
LDRV	0.4	10	0	236565.28	16004.99	233389.54	239741.01
DLDRV	0.4	10	0	240871.77	16606.06	237576.77	244166.77
PerfectInfo	0.4	10	10	214682.27	13181.89	212066.69	217297.84
EV	0.4	10	10	199507.15	19080.54	195721.15	203293.14
RP	0.4	10	10	205576.22	15558.93	202488.99	208663.45
LDRV	0.4	10	10	200455.95	15186.10	197442.70	203469.20
DLDRV	0.4	10	10	203806.67	15456.92	200739.68	206873.66
PerfectInfo	1	0	0	61258.19	2670.82	60728.24	61788.14
EV	1	0	0	1619.36	18043.22	-1960.81	5199.52
RP	1	0	0	30994.92	3736.08	30253.60	31736.24
LDRV	1	0	0	22654.11	13.62	22651.41	22656.82
DLDRV	1	0	0	22654.11	13.62	22651.41	22656.82
PerfectInfo	1	0	10	50709.13	2037.14	50304.92	51113.34
EV	1	0	10	-1335.78	14495.35	-4211.97	1540.42
RP	1	0	10	26914.40	4265.12	26068.10	27760.69
LDRV	1	0	10	19401.93	5.57	19400.83	19403.04
DLDRV	1	0	10	19648.69	17.80	19645.15	19652.22
PerfectInfo	1	10	0	101231.58	3167.30	100603.12	101860.04
EV	1	10	0	93783.26	7826.59	92230.29	95336.22
RP	1	10	0	96549.55	5166.47	95524.41	97574.69
LDRV	1	10	0	78440.53	2126.91	78018.50	78862.55
DLDRV	1	10	0	83821.10	2434.41	83338.06	84304.14
PerfectInfo	1	10	10	92029.18	2659.53	91501.47	92556.89
EV	1	10	10	83739.07	8103.30	82131.20	85346.94
RP	1	10	10	87132.20	4938.93	86152.21	88112.20
LDRV	1	10	10	73196.72	2040.49	72791.85	73601.60
DLDRV	1	10	10	77682.64	2416.74	77203.10	78162.17

Table 6: Profits, uniform distribution, tightness set to 0.8.

Sol.	Perc. low.	cc	sc	Mean	Std	CI lower	CI upper
PerfectInfo	0.4	0	0	114456.37	5527.16	113359.66	115553.08
EV	0.4	0	0	72586.90	11819.36	70241.68	74932.12
RP	0.4	0	0	85530.38	6674.97	84205.92	86854.84
LDRV	0.4	0	0	59403.76	113.22	59381.30	59426.23
DLDRV	0.4	0	0	80924.36	8698.28	79198.43	82650.28
PerfectInfo	0.4	0	10	90344.18	3752.27	89599.65	91088.71
EV	0.4	0	10	54873.01	9631.02	52962.01	56784.01
RP	0.4	0	10	68105.43	5294.47	67054.90	69155.97
LDRV	0.4	0	10	48685.97	178.87	48650.48	48721.46
DLDRV	0.4	0	10	64545.47	6541.66	63247.46	65843.48
PerfectInfo	0.4	10	0	154487.45	7650.03	152969.52	156005.39
EV	0.4	10	0	147448.36	10188.38	145426.77	149469.96
RP	0.4	10	0	148931.11	8486.51	147247.20	150615.02
LDRV	0.4	10	0	147050.84	7770.17	145509.07	148592.61
DLDRV	0.4	10	0	147954.41	7750.64	146416.52	149492.30
PerfectInfo	0.4	10	10	135869.54	6164.79	134646.32	137092.77
EV	0.4	10	10	128136.82	7093.58	126729.30	129544.34
RP	0.4	10	10	132124.99	6538.94	130827.52	133422.46
LDRV	0.4	10	10	127900.77	5957.74	126718.62	129082.91
DLDRV	0.4	10	10	130362.43	6269.31	129118.46	131606.40
PerfectInfo	1	0	0	45489.86	1380.53	45215.94	45763.79
EV	1	0	0	9859.95	10293.36	7817.53	11902.38
RP	1	0	0	31193.83	2669.04	30664.23	31723.42
LDRV	1	0	0	23071.78	27.91	23066.24	23077.31
DLDRV	1	0	0	23685.69	36.29	23678.49	23692.89
PerfectInfo	1	0	10	37247.57	1300.94	36989.43	37505.70
EV	1	0	10	216.63	12262.96	-2216.61	2649.87
RP	1	0	10	24752.19	2287.80	24298.24	25206.14
LDRV	1	0	10	18835.83	32.55	18829.37	18842.29
DLDRV	1	0	10	19264.43	47.35	19255.04	19273.83
PerfectInfo	1	10	0	66668.99	1836.96	66304.50	67033.48
EV	1	10	0	64353.36	3145.91	63729.14	64977.57
RP	1	10	0	65106.83	2145.35	64681.15	65532.52
LDRV	1	10	0	59608.20	1362.63	59337.83	59878.58
DLDRV	1	10	0	61987.90	1530.24	61684.27	62291.53
PerfectInfo	1	10	10	65648.68	1657.05	65319.89	65977.47
EV	1	10	10	62338.23	3533.46	61637.12	63039.35
RP	1	10	10	63457.73	2312.19	62998.94	63916.52
LDRV	1	10	10	58096.87	1507.79	57797.69	58396.05
DLDRV	1	10	10	60289.18	1635.22	59964.71	60613.64

Table 7: Profits, right skewed beta distribution, tightness set to 0.8.

Sol.	Perc. low.	cc	sc	Mean	Std	CI lower	CI upper
PerfectInfo	0.4	0	0	149438.09	3702.59	148703.41	150172.76
EV	0.4	0	0	114084.66	12347.73	111634.61	116534.72
RP	0.4	0	0	122758.29	7999.36	121171.04	124345.54
LDRV	0.4	0	0	108732.17	6095.61	107522.67	109941.68
DLDRV	0.4	0	0	117452.30	10746.66	115319.92	119584.67
PerfectInfo	0.4	0	10	152975.86	4127.66	152156.84	153794.88
EV	0.4	0	10	113159.36	12740.04	110631.46	115687.26
RP	0.4	0	10	127788.16	6849.34	126429.11	129147.22
LDRV	0.4	0	10	112315.71	5116.13	111300.56	113330.86
DLDRV	0.4	0	10	122693.08	10051.00	120698.75	124687.42
PerfectInfo	0.4	10	0	244897.23	6831.21	243541.77	246252.69
EV	0.4	10	0	239308.76	7169.85	237886.10	240731.41
RP	0.4	10	0	240738.52	7545.80	239241.27	242235.77
LDRV	0.4	10	0	239922.02	7341.55	238465.30	241378.75
DLDRV	0.4	10	0	240468.93	7360.77	239008.39	241929.46
PerfectInfo	0.4	10	10	255061.10	8054.91	253462.83	256659.37
EV	0.4	10	10	247672.94	8645.19	245957.55	249388.34
RP	0.4	10	10	250349.81	8258.17	248711.21	251988.41
LDRV	0.4	10	10	248530.24	8486.81	246846.27	250214.20
DLDRV	0.4	10	10	249087.79	8421.31	247416.82	250758.76
PerfectInfo	1	0	0	75427.65	1423.53	75145.19	75710.10
EV	1	0	0	37493.11	11535.35	35204.25	39781.98
RP	1	0	0	57743.17	5246.06	56702.24	58784.10
LDRV	1	0	0	34088.06	201.00	34048.18	34127.94
DLDRV	1	0	0	51297.41	5421.56	50221.66	52373.16
PerfectInfo	1	0	10	74076.21	1553.85	73767.89	74384.52
EV	1	0	10	38713.77	13287.85	36077.17	41350.37
RP	1	0	10	58325.33	5479.39	57238.10	59412.56
LDRV	1	0	10	34035.58	28.88	34029.85	34041.31
DLDRV	1	0	10	51958.51	4871.88	50991.82	52925.20
PerfectInfo	1	10	0	108480.08	2880.56	107908.52	109051.65
EV	1	10	0	105755.80	3586.08	105044.24	106467.36
RP	1	10	0	106836.42	3222.92	106196.93	107475.92
LDRV	1	10	0	104704.46	2770.14	104154.80	105254.11
DLDRV	1	10	0	106577.24	3117.41	105958.68	107195.80
PerfectInfo	1	10	10	86787.27	1733.83	86443.24	87131.30
EV	1	10	10	83325.86	4057.49	82520.77	84130.96
RP	1	10	10	85770.73	1862.47	85401.17	86140.28
LDRV	1	10	10	84304.84	1765.01	83954.62	84655.05
DLDRV	1	10	10	85287.73	1807.97	84928.99	85646.47

Table 8: Profits, left skewed beta distribution, tightness set to 0.8.

Sol.	Perc. low.	tight	sc	Mean	Std	CI lower	CI upper
PerfectInfo	1	0.8	0	75427.65	1423.53	75145.19	75710.10
EV	1	0.8	0	37493.11	11535.35	35204.25	39781.98
RP	1	0.8	0	57743.17	5246.06	56702.24	58784.10
PerfectInfo	1	0.8	10	74076.21	1553.85	73767.89	74384.52
EV	1	0.8	10	38713.77	13287.85	36077.17	41350.37
RP	1	0.8	10	58325.33	5479.39	57238.10	59412.56
PerfectInfo	1	1.3	0	92753.50	3474.82	92064.02	93442.98
EV	1	1.3	0	38755.98	17323.96	35318.53	42193.43
RP	1	1.3	0	63195.24	6211.79	61962.68	64427.79
PerfectInfo	1	1.3	10	74810.66	2916.30	74232.00	75389.32
EV	1	1.3	10	25416.16	15430.00	22354.51	28477.81
RP	1	1.3	10	50406.15	5311.27	49352.28	51460.03

Table 9: EVPI and VSS, left skewed beta distribution, no common components, low profitability.

Sol.	Perc. low.	tight	sc	Mean	Std	CI lower	CI upper
PerfectInfo	0.4	0.8	0	149438.09	3702.59	148703.41	150172.76
EV	0.4	0.8	0	114084.66	12347.73	111634.61	116534.72
RP	0.4	0.8	0	122758.29	7999.36	121171.04	124345.54
PerfectInfo	0.4	0.8	10	152975.86	4127.66	152156.84	153794.88
EV	0.4	0.8	10	113159.36	12740.04	110631.46	115687.26
RP	0.4	0.8	10	127788.16	6849.34	126429.11	129147.22
PerfectInfo	0.4	1.3	0	196816.70	7448.00	195338.86	198294.55
EV	0.4	1.3	0	142192.26	18248.39	138571.39	145813.14
RP	0.4	1.3	0	154805.04	11499.13	152523.36	157086.72
PerfectInfo	0.4	1.3	10	172768.17	7874.82	171205.64	174330.71
EV	0.4	1.3	10	123678.97	16921.44	120321.39	127036.55
RP	0.4	1.3	10	136705.23	10705.82	134580.97	138829.50

Table 10: EVPI and VSS, left skewed beta distribution, no common components, medium profitability.

Sol.	Perc. low.	tight	sc	Mean	Std	CI lower	CI upper
PerfectInfo	1	0.8	0	45489.86	1380.53	45215.94	45763.79
EV	1	0.8	0	9859.95	10293.36	7817.53	11902.38
RP	1	0.8	0	31193.83	2669.04	30664.23	31723.42
PerfectInfo	1	0.8	10	37247.57	1300.94	36989.43	37505.70
EV	1	0.8	10	216.63	12262.96	-2216.61	2649.87
RP	1	0.8	10	24752.19	2287.80	24298.24	25206.14
PerfectInfo	1	1.3	0	56583.60	3600.41	55869.21	57298.00
EV	1	1.3	0	4075.16	13476.80	1401.07	6749.25
RP	1	1.3	0	34216.63	3049.04	33611.63	34821.62
PerfectInfo	1	1.3	10	48934.36	2651.72	48408.21	49460.52
EV	1	1.3	10	6763.54	10518.30	4676.48	8850.59
RP	1	1.3	10	30055.35	2431.70	29572.85	30537.85

Table 11: EVPI and VSS, right skewed beta distribution, no common components, low profitability.

Sol.	Perc. low.	tight	sc	Mean	Std	CI lower	CI upper
PerfectInfo	0.4	0.8	0	114456.37	5527.16	113359.66	115553.08
EV	0.4	0.8	0	72586.90	11819.36	70241.68	74932.12
RP	0.4	0.8	0	85530.38	6674.97	84205.92	86854.84
PerfectInfo	0.4	0.8	10	90344.18	3752.27	89599.65	91088.71
EV	0.4	0.8	10	54873.01	9631.02	52962.01	56784.01
RP	0.4	0.8	10	68105.43	5294.47	67054.90	69155.97
PerfectInfo	0.4	1.3	0	129813.94	8633.82	128100.80	131527.08
EV	0.4	1.3	0	71860.84	16882.32	68511.02	75210.66
RP	0.4	1.3	0	90762.72	7885.56	89198.06	92327.39
PerfectInfo	0.4	1.3	10	105724.21	6826.58	104369.67	107078.75
EV	0.4	1.3	10	57595.38	12538.75	55107.42	60083.34
RP	0.4	1.3	10	73981.93	6932.50	72606.38	75357.49

Table 12: EVPI and VSS, right skewed beta distribution, no common components, medium profitability.

Sol.	Perc.	low.	tight	sc	Mean	Std	CI lower	CI upper
PerfectInfo	1	0.8	0	169492.01	12141.99	167082.77	171901.24
EV	1	0.8	0	-158798.74	95016.51	-177652.08	-139945.41
RP	1	0.8	0	34887.07	15495.28	31812.47	37961.67
PerfectInfo	1	0.8	10	184231.50	12361.18	181778.77	186684.23
EV	1	0.8	10	-135944.53	90011.14	-153804.70	-118084.37
RP	1	0.8	10	37116.44	16367.64	33868.75	40364.14
PerfectInfo	1	1.3	0	259362.00	32376.32	252937.84	265786.17
EV	1	1.3	0	-229223.76	129956.19	-255009.89	-203437.63
RP	1	1.3	0	49910.74	21170.97	45709.96	54111.52
PerfectInfo	1	1.3	10	203087.89	23736.88	198377.98	207797.80
EV	1	1.3	10	-161120.18	94163.28	-179804.22	-142436.14
RP	1	1.3	10	42556.54	18380.21	38909.51	46203.57

Table 13: EVPI and VSS, normal distribution, no common components, low profitability.

Sol.	Perc.	low.	tight	sc	Mean	Std	CI lower	CI upper
PerfectInfo	0.4	0.8	0	577300.84	58510.25	565691.14	588910.54
EV	0.4	0.8	0	110629.54	144266.58	82003.92	139255.16
RP	0.4	0.8	0	240768.05	84628.12	223975.99	257560.11
PerfectInfo	0.4	0.8	10	464189.02	47119.90	454839.41	473538.63
EV	0.4	0.8	10	71705.31	114133.92	49058.66	94351.95
RP	0.4	0.8	10	189231.23	59384.46	177448.07	201014.40
PerfectInfo	0.4	1.3	0	644874.09	84724.24	628062.96	661685.21
EV	0.4	1.3	0	72621.80	146740.36	43505.33	101738.27
RP	0.4	1.3	0	242824.57	76309.69	227683.07	257966.06
PerfectInfo	0.4	1.3	10	464473.84	62072.11	452157.39	476790.29
EV	0.4	1.3	10	22057.40	112263.51	-218.11	44332.92
RP	0.4	1.3	10	168290.72	48736.58	158620.33	177961.12

Table 14: EVPI and VSS, normal distribution, no common components, medium profitability.

Sol.	Perc.	low.	tight	sc	Mean	Std	CI lower	CI upper
PerfectInfo	1	0.8	0	61258.19	2670.82	60728.24	61788.14
EV	1	0.8	0	1619.36	18043.22	-1960.81	5199.52
RP	1	0.8	0	30994.92	3736.08	30253.60	31736.24
PerfectInfo	1	0.8	10	50709.13	2037.14	50304.92	51113.34
EV	1	0.8	10	-1335.78	14495.35	-4211.97	1540.42
RP	1	0.8	10	26914.40	4265.12	26068.10	27760.69
PerfectInfo	1	1.3	0	56839.95	3903.19	56065.48	57614.43
EV	1	1.3	0	-7742.65	16725.99	-11061.45	-4423.85
RP	1	1.3	0	25844.40	2949.71	25259.11	26429.68
PerfectInfo	1	1.3	10	57526.97	4370.78	56659.71	58394.23
EV	1	1.3	10	-10235.33	17766.39	-13760.57	-6710.09
RP	1	1.3	10	26812.68	3289.78	26159.91	27465.44

Table 15: EVPI and VSS, uniform distribution, no common components, low profitability.

Sol.	Perc.	low.	tight	sc	Mean	Std	CI lower	CI upper
PerfectInfo	0.4	0.8	0	143574.21	8814.77	141825.17	145323.25
EV	0.4	0.8	0	65036.31	26144.48	59848.68	70223.94
RP	0.4	0.8	0	89831.18	14277.67	86998.18	92664.18
PerfectInfo	0.4	0.8	10	116822.90	6076.20	115617.25	118028.55
EV	0.4	0.8	10	52010.25	18807.98	48278.34	55742.16
RP	0.4	0.8	10	73998.48	10788.05	71857.89	76139.06
PerfectInfo	0.4	1.3	0	176396.72	13375.48	173742.73	179050.70
EV	0.4	1.3	0	84988.93	25849.08	79859.91	90117.95
RP	0.4	1.3	0	109964.17	13892.10	107207.67	112720.66
PerfectInfo	0.4	1.3	10	121177.32	10158.72	119161.61	123193.03
EV	0.4	1.3	10	42506.44	22120.28	38117.29	46895.58
RP	0.4	1.3	10	70173.91	10331.60	68123.90	72223.92

Table 16: EVPI and VSS, uniform distribution, no common components, medium profitability.