AMAI LION12 Special Issue manuscript No.
(will be inserted by the editor)

Combining Intelligent Heuristics with Simulators in Hotel Revenue Management

Mauro Brunato · **Roberto Battiti**

Abstract Revenue Management uses data-driven modelling and optimization methods to decide what to sell, when to sell, to whom to sell, and for which price, in order to increase revenue and profit. Hotel Revenue Management is a very complex context characterized by nonlinearities, many parameters and constraints, and stochasticity, in particular in the demand by customers. It suffers from the *curse of dimensionality* [5]: when the number of variables increases (number of rooms, number possible prices and capacities, number of reservation rules and constraints) exact solutions by dynamic programming or by alternative global optimization techniques cannot be used and one has to resort to intelligent heuristics, i.e., methods which can improve current solutions but without formal guarantees of optimality. Effective heuristics can incorporate "learning" ("reactive" schemes) that update strategies based on the past history of the process, the past reservations received up to a certain time and the previous steps in the iterative optimization process.

Different approaches can be classified according to the specific *model* considered (stochastic demand and hotel rules), the control mechanism (the *pricing policy*) and the *optimization technique* used to determine improving or optimal solutions. In some cases, model definitions, control mechanism and solution techniques are strongly interrelated: this is the case of dynamic programming, which demands suitably simplified problem formulations.

We design a flexible discrete-event simulator for the hotel reservation process and experiment different approaches though measurements of the expected effect on profit (obtained by carefully separating a "training" phase from the final "validation" phase obtained from different simulations).

The experimental results show the effectiveness of intelligent heuristics with respect to exact optimization methods like dynamic programming, in particular for more constrained situations (cases when demand tends to saturate hotel room availability), when the simplifying assumptions needed to make the problem analytically treatable do not hold.

Keywords Machine learning and Intelligent optimization · Hotel revenue management · Simulation-Based Optimization · Optimization Heuristics

M. Brunato, ORCID: 0000-0002-7885-4255, corresponding author
DISI — Università di Trento, Via Sommarive 5, I-38123 Trento, Italy
E-mail: mauro.brunato@unitn.it

R. Battiti, ORCID: 0000-0002-0259-8603
DISI — Università di Trento, Via Sommarive 5, I-38123 Trento, Italy
E-mail: roberto.battiti@unitn.it

1 Revenue management and intelligent heuristics

Optimization techniques are heavily used in Revenue Management, which flourished after the deregulation of the airline industry in the 1970s. In airline flights the products available for sale are sequences of individual flights ("legs") to go from an origin to a final destination, in possible multi-hop flights. In hotels, the sold products are a sequence of resources consisting of single overnight stays (from check-in to check-out). The "network" effect is caused by the interaction between different products: if a room is not available for one day, a more extended stay at the hotel is impossible.

An approximated network flow formulation for network revenue management is proposed in [15]. The work [6] developed the EMSR (Expected Marginal Seat Revenue) heuristic to determine booking limits for the different fare classes, while [25] used simulation to address the network aspect of RM. A seminal work on optimal dynamic pricing of inventories with stochastic demand over a finite horizons is [14].

The idea of Simulation-Based-Optimization and Monte Carlo integration in the RM context is advocated in [21] for the single-leg problem. Simulation-Based booking limits for RM in the airline business are studied in [7], and later extended by [24], which considers a continuous model, less realistic in terms of the fine-grained details but leading to an easier continuous optimization problem. Simulation-based optimization and a stochastic steepest ascent algorithm is used in [22] to maximize revenue under customer choice behaviour. According to the authors modelling *flexibility* is a plus: "one can make essentially arbitrary changes in the model of demand and customer behaviour without impacting the way the optimization algorithm functions." The work [13] present a stochastic gradient algorithm for improvement of bid prices with customer choice. Auto-adaptive bid prices by means of the metaheuristic *scatter search* are introduced in [17], assuming independent demand. Simultaneously learning and optimizing are considered in [9] and [8].

Revenue Management is based upon demand modelling and forecasting, which provide the essential input for the optimization model. Early RM setups assumed the "independent demand model", i.e., assumed that demand is independent for different products or price classes [23]. This is a rough approximation, which excludes any substitution effects by consumers.

In Theoretical Computer Science, the strong negative results of the last decades in the area of Computational Complexity [18] imply that many real-world optimization tasks with large number of variables cannot be solved to optimality in reasonable computing times. Using heuristics instead of exact techniques is not done because of laziness but because there is no other way to deliver workable solutions. It has to be noted that, given the high level of "measurement noise" which is characteristic of hotels, in many cases the difference between theoretically optimal solutions and heuristic ones can be almost irrelevant in practice, being hidden by the measurement noise.

But resorting to heuristics does not imply abandoning the procedures of careful experimental science. On the contrary, heuristics need to be designed by following statistically sound experiments. Real experiments with hotels are almost ruled out, apart from exceptional and carefully controlled tests, because of the very long times involved (many months or years) and, in particular, because of the unacceptable disruption that they would cause on the hotel daily operations. Fortunately, massive experiments are now made possible by fast *hotel simulators* which cab be designed and trained on the hotel data to simulate with high fidelity the hotel reservation process and the final distribution of economic results.

Two theoretical advancements in the past years which permit disruptive innovation in areas characterized by noisy data, lack of exhaustive mathematical models and extremely complex optimization tasks are *machine learning* and *intelligent optimization* [2]. *Machine learning* (*ML*), or *learning from data*, is a theory for deriving flexible models by starting only from the data produced by the business. The objective of these models is to *generalize* in a sound manner for cases not already encountered in the past. They can be used by hotel to model demand by customers, price elasticity, dependence of demand on season, day of week, events, weather predictions, competition, etc. After a model is available, computers can simulate the effects of millions of possible decisions, by predicting the output (for example the total profit of the hotel), and by creating and selecting one among the best decisions. *Intelligent optimization* (*IO*) is this automated process of creating in an intelligent manner a large series of possible decisions, aiming at improving the current way of doing business. In *simulation-based optimization*, a simulator of the system (e.g., a simulator to derive the annual profit of an hotel) is run starting from different settings of the possible choices generated in an intelligent manner by the optimization algorithm. Sample-based estimates of the mean profit can be used as noisy signals to guide the optimization process. The final configuration will then be evaluated by considering a different an independent number of runs with different random seeds (to avoid over-optimistic results caused by the difference between the real mean value and the noisy estimates).

The specific proposed simulation model increases the flexibility by avoiding the bid-prices based control considered in [17] and the assumptions of virtual nesting controls in [24]. The focus on learning models from data by machine learning (without requiring a high level of expertise in designing explicit models) and on using black-box heuristics for identifying improving solutions is intended to simplify adoption in contexts which do not have a software support based on dynamic programming and approximations thereof.

The following part of this paper is organized as follows. Section 2 defines the mathematical model of the hotel reservation process, as seen by the hotel owner and by his potential customers, which is the basis for developing our discrete-event simulator.

Section 3 introduces the specific experimental setting used in this paper by describing the stochastic demand generator, the price acceptance model (customers are assumed to be price takers, they observe the price offered by the seller and react by buying or not buying) and the simulation details.

Section 4 describe the pricing policies that we consider in this work: price is the basic control variable influencing the final revenue, which is described by the value function. In addition we mention the three heuristic optimization strategies considered in this work (for brevity the full details of the heuristics are contained in the given citations).

Finally, Section 5 analyses the results of experiments and derives some conclusions and insight.

2 Problem definition

Effective models need to consider current ways of reserving rooms, which happen more and more though electronic means, in particular through hotel *booking engines* inserted into hotel websites or through collaborating Online Travel Agencies (OTAs), where the potential customers can issue requests, obtain availability and prices and reserve in a self-service manner.

Our model is designed to capture this common reservation protocol as represented in Fig. 1, where a customer issues a *reservation request* with some parameters (e.g., check-in and check-out date, number of rooms…) and the hotel responds with a price offer; the customer can implicitly refuse the offer by not issuing any further communication (case 1

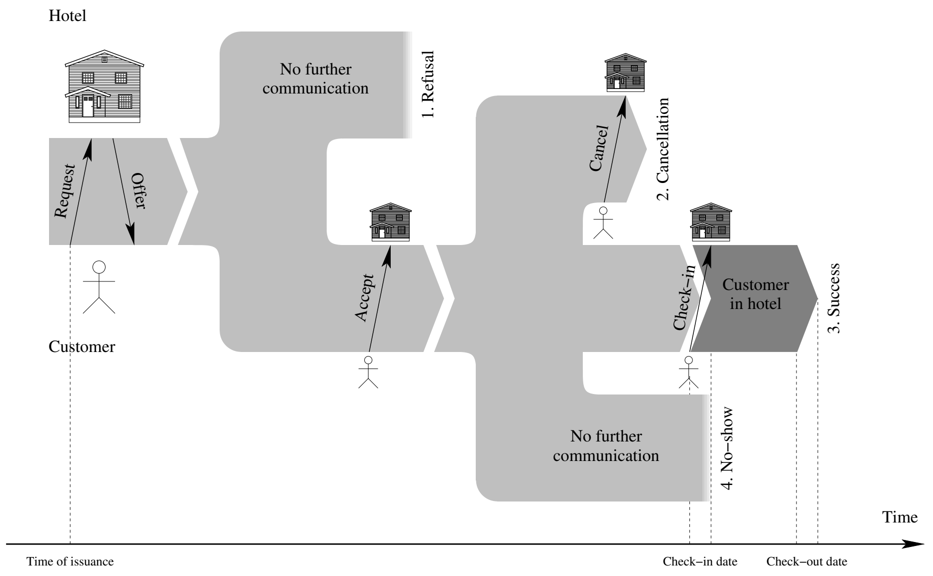


Fig. 1 Outcomes of the request/reservation process

in the Figure), or accept it with the option of cancelling it at a later date (case 2), check in (case 3) or simply avoid showing up (case 4).

A fulfilled reservation is modelled as a reservation whose cancellation time is $+\infty$; a refused proposal is modelled by cancelling the reservation immediately upon issuance (we assume that the request/proposal/acceptance handshake is immediate). A no-show is modelled as a reservation that is cancelled immediately before the latest possible check-in time.

2.1 Timeframe

We simulate a system that evolves in time, with a one-day granularity, over a limited period of D days; dates are identified by an integer $d = 0, \ldots, D-1$.

Each event is characterized by real-valued timestamp t (e.g., an event happening at noon of the second simulation day has a timestamp of 1.5). Dates and timestamps are comparable: we say that timestamp $t \in \mathbb{R}$ belongs to date $d \in \mathbb{N}$ if and only if $d \leq t < d+1$.

In the following, whenever timestamps are used as indexes they will be written as subscripts; dates as indexes will be written as superscripts.

2.2 Room pools and Hotels

An accommodation type (e.g. “single room”, “double room”, “superior double room”) is the product which is marketed and sold by modern hotels. The fact of selling a type and not a specific room (like “room number 163”) gives more flexibility for allocating guests at check-in, for creating spaces for longer possible reservations by permuting rooms, and for setting different price categories for the different types, to target customers with different available budgets. Rooms in a given accommodation type are indistinguishable from a marketing and sales prospective.

In this paper we will make the assumption that the hotel offers a single accommodation type. This is equivalent to the hypothesis that the customer's acceptance only depends on the proposed price and not on comparisons, therefore allowing us to model different room pools as if they were separate hotels. Note that this is not a limitation of the simulator, but a simplifying assumption for the investigation described in this paper.

A *hotel* is therefore modelled as a system that maintains:

– a set of N indistinguishable rooms;
– a set R of *reservations* related to the acquisition and release of these resources at specific dates (see below).

2.3 Customers

We assume a potentially infinite pool of concurrent and independent clients (the hotel customers) who plan in advance the acquisition of a resource for a given date interval. They perform reservation requests and may convert it into an actual reservation, as described below.

2.4 Reservations

A reservation is described by the following tuple of parameters:

$$r = (d_{\text{in}}, d_{\text{out}}, n_{\text{grp}}, t_{\text{iss}}, t_{\text{canc}}) \tag{1}$$

where d_{in} is the date in which rooms will be acquired (check-in), to be released at the beginning of the check-out date d_{out} (the resources are therefore available to be acquired again at d_{out}); n_{grp} is the number of requested resources, in order to simulate group reservations; $t_{\text{iss}} \in \mathbb{R}$ is the time at which the reservation was issued. After issuing a reservation, but within the acquisition date (inclusive), a customer may cancel it. This cancellation is encoded in timestamp $t_{\text{canc}} \in \mathbb{R} \cup \{+\infty\}$, where $t_{\text{canc}} = +\infty$ means that the reservation is not cancelled at any time.

Consumer may reserve for future days but for the current (check-in) date, and they must reserve for at least one day. Therefore,

$$\lfloor t_{\text{iss}} \rfloor \le d_{\text{in}} < d_{\text{out}}. \tag{2}$$

Given the restrictions on cancellation timestamps, the following constraint must also hold:

$$t_{\text{canc}} < +\infty \quad \Rightarrow \quad t_{\text{iss}} \le t_{\text{canc}} < d_{\text{in}} + 1. \tag{3}$$

The number of resources requested in a reservation must be positive:

$$n_{\text{grp}} \ge 1. \tag{4}$$

Given a reservation r defined as in (1), the following shorthand notation will be used throughout this work. The *length of stay* of a reservation is the number of dates it spans, including the check-in date but not the check-out:

$$\text{LoS}(r) = d_{\text{out}} - d_{\text{in}}; \tag{5}$$

the *time to arrival* is the time between the issue time-stamp and the last possible arrival time:

$$\mathrm{TtA}(r) = d_{\mathrm{in}} + 1 - t_{\mathrm{iss}}. \tag{6}$$

When confusion is possible or more than one reservations are being used in the same formula, the superscript notation d^r_{in} shall be used to denote field d_{in} of reservation r.

2.5 Hotel state

The state of the hotel is defined by the number N of rooms and by the set R of reservations. Without loss of generality, we can assume that no two reservation requests are issued at precisely the same (real-valued) timestamp, so that two identical requests cannot exist; they are totally ordered by their issuance time, and for every timestamp the sets of past and future requests are well defined.

In the simulator, an important concept is the *observed history of the selling process at time t*, the set of all relevant information available to the hotel up to t. Let R be the complete sequence of reservation requests for the whole simulation timespan. The hotel's *view* of a reservation r at time t is the following partial function:

$$\mathrm{view}_t(r) = \begin{cases} (d^r_{\mathrm{in}}, d^r_{\mathrm{out}}, n^r_{\mathrm{grp}}, t^r_{\mathrm{iss}}, t^r_{\mathrm{canc}}) & \text{if } t \geq t^r_{\mathrm{canc}} \\ (d^r_{\mathrm{in}}, d^r_{\mathrm{out}}, n^r_{\mathrm{grp}}, t^r_{\mathrm{iss}}, +\infty) & \text{if } t^r_{\mathrm{iss}} \leq t < t^r_{\mathrm{canc}} \\ \text{undefined} & \text{if } t < t_{\mathrm{iss}}, \end{cases} \tag{7}$$

where the middle line defines the case when the hotel does not know of the eventual cancellation yet. Therefore, we define the set of reservations known at time t as:

$$R_t = \{\mathrm{view}_t(r) \mid r \in R \wedge t^r_{\mathrm{iss}} \leq t\}. \tag{8}$$

In short, $\mathrm{view}_t(r)$ "filters out" future information from a reservation r provided that it is known at time t, while R_t is a version of R where all information newer than t is removed.

The full set of reservations for day d is the set of reservations that imply the lock of a resource for day d:

$$R^d = \{r \in R \mid t^r_{\mathrm{canc}} = +\infty \wedge d^r_{\mathrm{in}} \leq d \wedge d^r_{\mathrm{out}} > d\}. \tag{9}$$

We can combine (8) and (9) together to obtain the set of reservations, known by the hotel at time t, that imply the lock of a resource at day d:

$$R^d_t = R_t \cap R^d. \tag{10}$$

Note that sets R^d_t is *not* monotonic with respect to t due to the possibility of cancellations.

An important item of information that can be obtained from these sets is the *availability* at date d, i.e., the number of rooms that are not locked at date d:

$$\mathrm{avail}^d = N - \sum_{r \in R^d} n^r_{\mathrm{grp}}. \tag{11}$$

The hotel's knowledge at time t about the room availability for date d is therefore:

$$\mathrm{avail}^d_t = N - \sum_{r \in R^d_t} n^r_{\mathrm{grp}}. \tag{12}$$

2.6 Reservation pricing

A reservation request r is associated to a price which the hotel will compute by means of a deterministic function $\operatorname{price}(r,R)$ based on the request itself and on the hotel's state. Given the full set R of reservations in the whole timeframe, the price of reservation r will only depend on the past, valid reservations:

$$\operatorname{price}(r,R) = \operatorname{price}(d^r_{\text{in}}, d^r_{\text{out}}, n^r_{\text{grp}}, t^r_{\text{iss}}, R_{t^r_{\text{iss}}}). \tag{13}$$

Given this formulation, the hotel can determine the price on the basis of all knowledge available at the issuance time: the reservation's dates, group size, length of stay and time to arrival, room availability and past reservation history; for instance, it can perform statistics on past reservations. The only two preclusions that (13) takes into account are: (i) reference to future reservations or cancellations, which are explicitly excluded by (8), and (ii) reference to the reservation's cancellation date t^r_{canc}, which will only be decided later.

The function $\operatorname{price}(\cdot,\cdot)$ returns values in $\mathbb{R} \cup \{+\infty\}$. Since the pricing function will be used by the customer to decide whether to keep or cancel the reservation (see below), returning $+\infty$ is a way to force the customer's refusal: returning $+\infty$ will cause the reservation to be immediately cancelled. In particular, the price of a room will only be finite if enough rooms are available (to the hotel's knowledge at the reservation's issuing time) on all dates:

$$\operatorname{price}(r,R) < +\infty \quad \Rightarrow \quad \forall d = d^r_{\text{in}}, \dots, d^r_{\text{out}} - 1 \quad \left(\text{avail}^d_{t^r_{\text{iss}}} \geq n^r_{\text{grp}}\right). \tag{14}$$

2.7 Reservation requests

Note that a reservation such that $t_{\text{canc}} = t_{\text{iss}}$ (i.e., a request rejected at issuance time) is always excluded from the partial R^d and R^d_t sets for any timestamp t and date d, but it is included in all partial R_t sets where $t \geq t_{\text{iss}}$. By virtue of (13), the hotel can still use its information when determining the pricing of subsequent reservations, but it will not consider it valid (and hence occupy a room) at any time.

3 Experimental settings

In order to test the effectiveness of the proposed simulation model, we discuss its application to different pricing policies listed in Sec.4.

Besides the assessment of a policy's performance, we focus on using the proposed simulator to gain insight about the behaviour of the tested policies in different conditions. To this aim, a time-varying generative model for reservation requests is described in Sec. 3.1.1. Results are discussed in Sec. 5, with particular emphasis on day-by-day observations in Sec. 5.2.

3.1 Hotel and customer model: request arrivals and price acceptance

3.1.1 Generative model for requests

We consider hotels with $N = 10$ and $N = 50$ rooms, where rooms are indistinguishable In order to generate simulated data and test a pricing method, we simulate a set of reservation

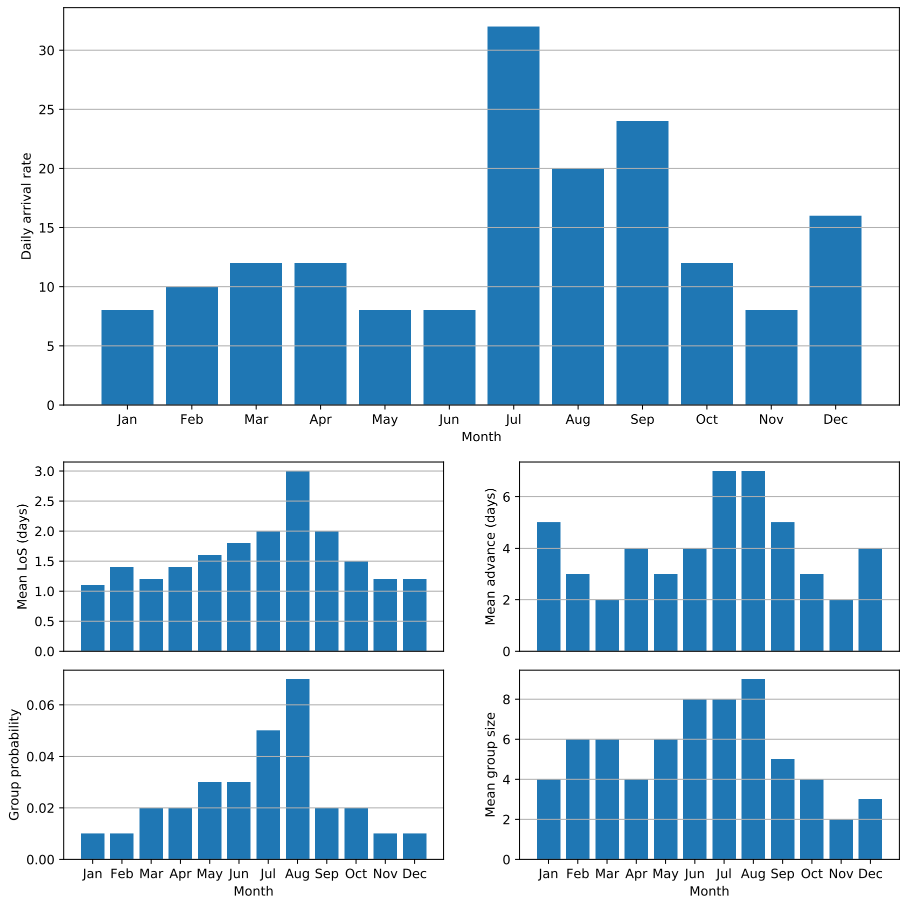


Fig. 2 Reservation generator time-varying parameters. Top: expected requests per day. Below, clockwise: expected length of stay, mean advance with respect to check-in date, expected size for group reservations (more than one room), probability of group reservations. Parameters adapted from historical data from some Northern Italian hotels.

requests whose check-in and check-out date fall within a $D = 360$ day period ("year"). The number of days has been chosen to be divided into twelve 30-day periods ("months"), each having a different distribution of reservation requests.

Fig. 2 shows the month-dependent parameter settings used for reservation generation in this work. These parameters are the average of historical data from a small set of Northern Italian hotels.

Reservation requests are independently generated so that their check-in dates d_{in} are distributed as in the "Daily arrival rate" chart of Fig. 2. Observe that the rates have been chosen so that the 50-room hotel can almost always accommodate them, while the 10-room hotel is often saturated. Based on the check-in date, the other reservation parameters are set as follows:

Advance time — the reservation's issuance time is set to $t_{\text{iss}} = d_{\text{in}} + 1 - \text{TtA}$, where TtA is the time-to-arrival, drawn from an exponential distribution whose mean depends on the month ("Mean advance" in Fig. 2);

Length of stay — the reservation's check-out date is set to $d_{\text{out}} = d_{\text{in}} + \lceil\lambda\rceil$, where λ is drawn from an exponential distribution whose mean depends on the month ("Mean LoS" in Fig. 2);

Group size — the reservation's group size is set to

$$n_{\text{grp}} = \begin{cases} \lceil\eta\rceil & \text{with probability } p \\ 1 & \text{with probability } 1-p, \end{cases}$$

where p is a (small) probability that the group's size is greater than 1 (we assume that most reservations are for one room), while η is drawn from an exponential distribution. Both p and the distribution's mean depend on the month ("Group probability" and "Mean group size" in Fig. 2).

3.1.2 Price acceptance model

In our experiments, we assume that the probability of a requesting customer accepting a given price u has a sigmoidal shape with parameters $\mu, \eta > 0$:

$$p_a(u) = 1 - \sigma\left(\frac{u-\mu}{\eta}\right), \tag{15}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the classical sigmoid function; p_a is therefore a decreasing function of price u; parameter μ sets the price with a 50% probability of acceptance, while parameter η defines the steepness of the decrease at the inflection point (the smaller, the steeper), and will be called *elasticity*.

In this paper, we assume that the price acceptance probability is given by (15) where $\mu = 1$ is the price with a 0.5 acceptance probability and $\eta = 0.1$ defines the customer's elasticity.

3.2 The simulation

Let the reservations in R be indexed by issuing order.

$$R = \{r_1, \dots, r_m\}, \quad \forall i, j = 1, \dots, m \quad (i < j \Leftrightarrow t_{\text{iss}}^{r_i} < t_{\text{iss}}^{r_j}),$$

where $m = |R|$ is the number of requests. Initially, all reservations are requests (i.e., $t_{\text{canc}}^r = t_{\text{iss}}^r$ for all $r \in R$).

The simulation consists of scanning all reservations in t_{iss} order, having the hotel determine the price, and having the customer decide whether to proceed with the reservation (by setting its cancellation date to $+\infty$) or not:

1. For each $i \leftarrow 1, \dots, m$:
   (a) Let $u \leftarrow \text{price}(d_{\text{in}}^{r_i}, d_{\text{out}}^{r_i}, n_{\text{grp}}^{r_i}, t_{\text{iss}}^{r_i}, R_{t_{\text{iss}}^{r_i}})$;
   (b) Let $t_{\text{canc}}^{r_i} \leftarrow \text{decision}(r_i, u)$.

The decision function is based on the price acceptance model (15) and returns a cancellation time t_{canc} with the following meaning:

- $t_{\text{canc}} = +\infty$: the reservation is confirmed, the room will be allocated for the requested time, the customer pays the requested price;
- $t^{r_i}_{\text{iss}} < t_{\text{canc}} < d^{r_i}_{\text{in}} + 1$: the reservation is initially confirmed, the room will be considered to be allocated until time t_{canc}, after which it will be free again, the customer shall not pay the requested price;
- $t_{\text{canc}} = t^{r_i}_{\text{iss}}$: the reservation is not confirmed, the room is not allocated, the customer shall not pay the requested price;

Since we use an infinite price as a way to keep customers from reserving (e.g., when a reservation cannot be fulfilled), we require the following constraint to be satisfied:

$$\forall r \quad \text{decision}(r, +\infty) = t^{r}_{\text{iss}}.$$

which reads as "all reservations with infinite price are immediately canceled."

4 Pricing policies

The main purpose of a revenue management simulation is to test and compare the effectiveness of various pricing policies under controlled and repeatable conditions, often unachievable in practice. In this work we consider two basic classes of pricing policies: exact policies that analytically determine prices based on known information and customer models, and heuristic policies that have free parameters to be trained by experimentation.

4.1 Exact policies

The following policies are based on knowledge of the price acceptance model and hotel capacity, and determine a reservation price purely based on analytical considerations. They do not have free parameters, and can therefore be considered exact techniques.

4.1.1 Median acceptance price — "Const median"

The room's daily price is always set to $u = \mu = 1$, so that acceptance probability is always 0.5. This rule only assumes knowledge of the price acceptance probability. This "baseline" pricing rule intends to model a situation in which the hotel owner lowers price when most customers reject offers, and increases price when most customers accept his offers, which is a reasonable first-order approximation of the behaviour of unsophisticated hotels.

4.1.2 Unsaturated equilibrium price — "Const equilibrium"

The room's constant daily price u maximizes the expected revenue on the hypothesis that there is an infinite supply of rooms, i.e., the hotel will never reach full occupancy (see Appendix A.1 for details):

$$u^* = \arg\max_{u\in\mathbb{R}} u \cdot p_a(u) = \eta \left(1 + W_0\left(e^{\frac{\mu}{\eta}-1}\right)\right), \tag{16}$$

where $W_0(\cdot)$ is the main branch of Lambert's W function.

4.1.3 Pickup-based dynamic price — "Dynamic"

To come up with an analytically treatable dynamic programming strategy, let us assume that all reservations are one-day long and that reservations for different days are independent and do not interfere with each other. Therefore, we can focus on a single check-in date: in this section, we assume that all reservations r compete for the same check-in date d^r_{in}.

Given a reservation request r, let $d = d^r_{\text{in}}$ be its check-in date. Based on all previously received requests for the same check-in date, assuming that advance times are exponentially distributes, we estimate the maximum-likelihood expected number N_r of requests for that date, and the mean advance time λ.

Time discretization — In order to apply a dynamic programming strategy, time before check-in is divided into intervals of size $\Delta t > 0$ and indexed backwards by $\tau = 0, 1, \ldots$:

$$\tau = \left\lfloor \frac{d^r_{\text{in}} + 1 - t^r_{\text{iss}}}{\Delta t} \right\rfloor,$$

so that time interval $\tau = 0$ is the latest possible time interval at which a request can be issued, and time interval τ ranges from $(\tau+1)\Delta t$ to $\tau\Delta t$ before check-in.

Reservation request probability — Let $f(t)$ be the probability distribution for a reservation request to be issued t days in advance (where t is a continuous variable). Then the expected number of reservation requests arriving at time interval $\tau = 0, 1, \ldots$ is

$$n_q(\tau) = N \int_{\tau\Delta t}^{(\tau+1)\Delta t} f(t)\,dt = N\Big(F\big((\tau+1)\Delta t\big) - F\big(\tau\Delta t\big)\Big). \tag{17}$$

If $n_q(\tau)$ is small enough (and we can make it arbitrarily small by reducing Δt), then the probability that more than one reservation arrives at time interval τ is negligible; therefore, the probability of a request being issued at time interval τ, out of N_r expected in the whole period, is

$$p_q(\tau) = n_q(\tau) + O\big(n_q(\tau)^2\big) \approx n_q(\tau). \tag{18}$$

Price acceptance probability — We assume that the hotel owner has no control over the arrival distribution of reservation requests, he can drive their conversion into actual reservations by manipulating their acceptance probability by means of the only control variable, the price u. The price influences the demand: economists talk about "price elasticity of demand" to measures the degree of responsiveness of quantity demanded to a change price. The elasticity of demand in in our case modeled by a probability of acceptance of an offer by a requesting customer $p_a(u,\tau)$, depending on the proposed price u and, possibly, on the time interval τ when the request is issued. Having the time interval as a parameter in intended to model customer behaviours depending on the "urgency" of the reservation. For example, a business customer booking with little advance might be inclined to accept a higher price.

Sale probability — The probability of selling a room at time interval τ with price u is the joint probability $p(u,\tau)$ of the two aforementioned events: arrival of a request at time interval τ and acceptance of price u:

$$p(u,\tau) = p_q(\tau)p_a(u,\tau), \tag{19}$$

where the acceptance event is implicitly conditioned to the occurrence of a request.

Value function — Suppose that, at time interval τ before check-in, the hotel has residual capacity $c \in \{0,\dots,N\}$ (i.e., c rooms are still to be sold). Then the expected value for a given price policy $u = u(c,\tau)$ can be defined by the following recurrent relation:

$$V(c,\tau) = p(u,\tau)\big(u + V(c-1,\tau-1)\big) + \big(1 - p(u,\tau)\big)V(c,\tau-1), \tag{20}$$

with the obvious boundary conditions:

$$\begin{aligned} c < 0 &\Rightarrow V(c,\tau) = -\infty \\ c \geq 0 \wedge \tau < 0 &\Rightarrow V(c,\tau) = 0. \end{aligned}$$

Optimal price policy — According to Bellman's principle the optimal price policy $u(c,\tau)$ is the one that maximizes $V(c,\tau)$ assuming that all $V(\cdot,\tau-1)$ are already optimal:

$$u(c,\tau) = \arg\max_{u>0} \Big[p(u,\tau)\big(u + V(c-1,\tau-1)\big) + \big(1 - p(u,\tau)\big)V(c,\tau-1)\Big]. \tag{21}$$

Let us fix c and τ; let

$$\begin{aligned} V_1 &= V(c-1,\tau-1), \\ V_2 &= V(c,\tau-1). \end{aligned}$$

Then the quantity between square brackets in (21), expressed as a function of price u, can be rewritten as

$$V(u) = p_q(\tau)p_a(u,\tau)(u + V_1 - V_2) + V_2.$$

Therefore, its derivative with respect to u is

$$V'(u) = p_q(\tau)\big(p_a(u,\tau) + (u + V_1 - V_2)p'_a(u,\tau)\big),$$

where for brevity $p'_a(u,\tau) = \partial p_a(u,\tau)/\partial u$. Thus, the optimal price can be obtained by solving for u the equation

$$p_a(u,\tau) + (u + V_1 - V_2)p'_a(u,\tau) = 0. \tag{22}$$

Observe that the equation does not explicitly depend on the probability $p_q(\tau)$ of receiving a reservation request: we determine a price on the assumption that a request has been received. On the other hand, $p_q(\cdot)$ enters into V_1 and V_2 by means of the recurrence equation (20), therefore the price for a reservation at time interval τ implicitly depends on the probability of receiving requests at subsequent time intervals $\tau-1, \tau-2, \dots, 1, 0$.

Explicit analytic form — By replacing (15) inside (22), we get the following closed-form optimal price policy for the sigmoidal acceptance case (see Appendix A.2 for details):

$$u(c,\tau) = \mu + \eta\Big(W_0(e^{\beta-1}) + 1 - \beta\Big), \quad \beta = \frac{V_1 - V_2 + \mu}{\eta}, \tag{23}$$

where $W_0(\cdot)$ is the main branch of Lambert's W function, and the dependency on c and τ is carried via β from V_1 and V_2. As an example, Fig. 3 shows the optimal price policy $u(c,\tau)$ for a $N = 10$-room hotel and a reservation horizon of 30 days with $N_r = 10$ expected reservations. Reservation advance time is given by a Weibull distribution with shape parameter $k = 3$ and a 10-day mean advance time; price $\mu = 1$ has 50% acceptance probability and elasticity is $\eta = 0.1$.

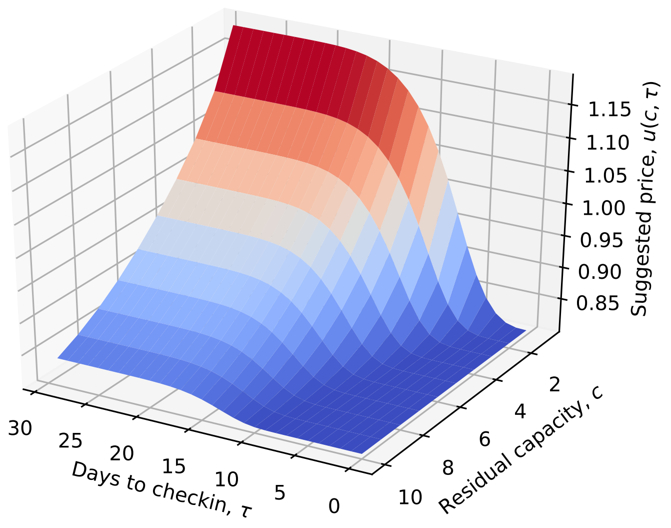


Fig. 3 Optimal price determined by dynamic programming for a 10-room hotel with 10 expected reservations. The price (u, vertical axis) is a function of the remaining number of free rooms c and the reservation advance time τ.

4.2 Heuristic policies

The following policies depend on free parameters, and need therefore the preliminary application of some procedure to find the optimal value set.

4.2.1 Best constant price — "Const Grid search"

Not unlike "Const median" and "Const equilibrium", this policy proposes a constant price. The price value is determined by a grid search on a training set of reservations for the price that maximizes the hotel's revenue. The only parameter, in this case, is the constant sale price.

4.2.2 Factored pricing — "Factored"

Pricing is determined as a product of four factors [4], each depending on a feature of the reservation r:

$$\text{price}(r) = f_1\bigl(\text{TtA}(r)\bigr) \cdot f_2\bigl(n^r_{\text{grp}}\bigr) \cdot f_3\bigl(\text{LoS}(r)\bigr) \cdot f_4(c) \cdot p_1$$

The first factor f_1 is a 2-piecewise linear function of the time to arrival which can accommodate for independent early and last-minute discounts or penalties, while the next three linearly depend on group size, length of stay and residual capacity at the time of reservation.

Finally, p_1 is an independent scaling constant. The time-to-arrival factor depends on three parameters, while the others depend on one; the overall policy depends on 7 parameters, which need to be optimized.

Observe that this policy reduces to the previous one if all factors are set to 1, with the only constant parameter p_1 allowed to vary.

While a comparison between heuristic optimization methods is out of scope for this paper, for this policy we consider three optimization strategies.

CMA-ES — An evolutionary optimization algorithm based on covariance matrix estimation [16].

Affine Shaker — Local search-based optimization method founded on the Collaborative Reactive Search Optimization (CoRSO) framework [10,1], particularly fit to low-dimensional search spaces.

Inertial Shaker — An alternative to the latter; less computation-intensive and therefore fit to higher-dimensional search spaces [11,12,3].

5 Experimental Results

The objective function used to evaluate all pricing policies is the revenue, obtained by adding the prices of all accepted reservations; however, we remove the initial and final 10 days from the computation in order to mitigate transient effects due to the fact that the hotel is initially empty and all guests leave within the final day of the simulation.

A single experiment consists of generating a set of reservation requests R and use it to train the heuristic pricing policies "Const grid search" and "Factored". Grid search is performed by discretizing the price range $[0.7, 1.3]$ into 100 intervals and finding the value that maximizes the hotel's revenue for R. Optimization of the factored model is performed by running the three optimization engines for a total of 10000 objective function evaluations.

The resulting policies are then tested on 100 newly generated sets of requests and the revenues are recorded.

The whole experimental suite is composed of 100 such experiments on each of four different kinds of hotels: a small hotel with $N = 10$ rooms, easily saturated, a larger hotel with $N = 30$ rooms, enough to accommodate the demand in non-peak months, a large hotel with $N = 50$ rooms, where most requests can be accommodated, and an $N = 100$-room hotel where refusals are basically due to customer price acceptance.

5.1 Revenue comparison

Fig. 4 shows the outcome of our tests: the revenues generated by all pricing schemes on 28 experiments, each comprising 100 test reservation sets are collected in boxplots. The median, first and third quartile are shown as a box, whiskers extend for $\frac{3}{2}$IQR and outliers are represented by circles, while the mean is represented by a triangle.

Since every pricing policy is evaluated on every reservation set, the Wilcoxon ranked-sum test can be applied to all pairs of pricing policies; in almost all cases, the null hypothesis (paired revenue differences being symmetrically distributed around zero) is rejected with very high confidence ($p < 10^{-70}$ on all pairs). The only exception ($p \approx .2$) is the comparison between dynamic programming and best constant price in the case of the 100-room hotel, where saturation issues do not come into play and the two policies become equivalent.

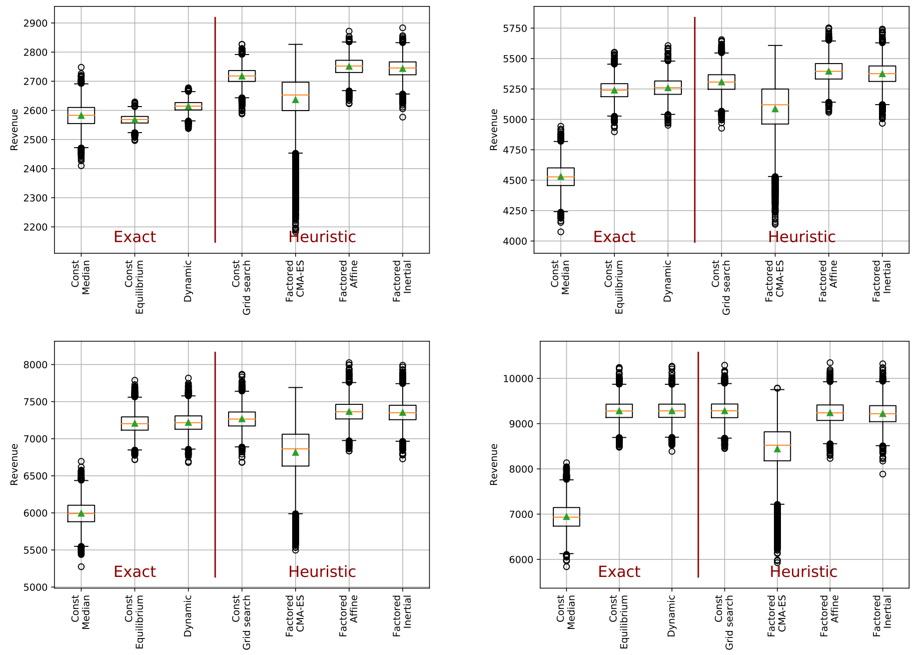


Fig. 4 Performance of the five pricing policies described in the text, combined when suitable with the three optimization techniques, for hotels with a varying number of rooms. From top left to bottom right: a 10-room hotel, often saturated, 30 rooms, 50 rooms, and 100 rooms. Boxes represent quartiles, the mean is represented by triangles, circles represent outliers (outside the $\frac{3}{2}$IQR-sized whiskers).

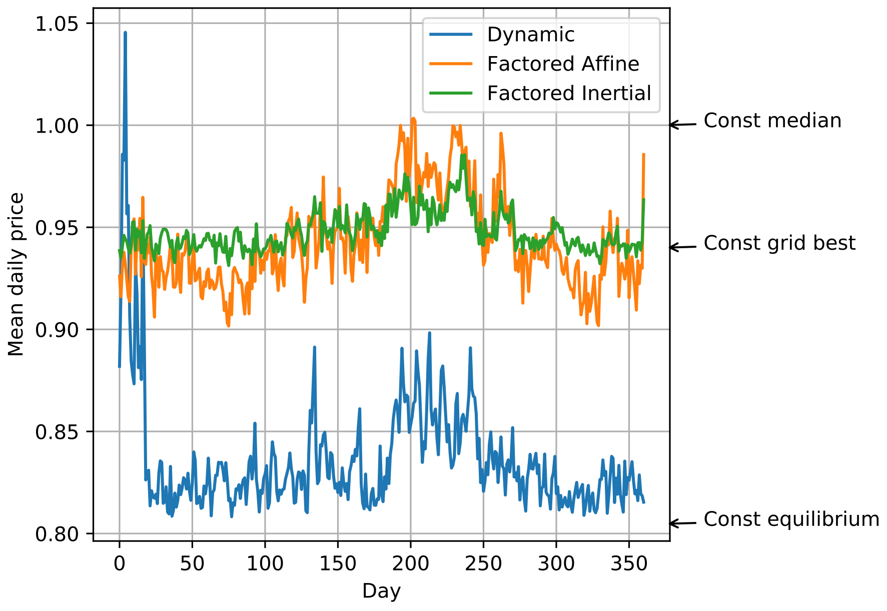
As expected, the baseline case "Const median" is quite inefficient; the theoretical unsaturated equilibrium price "Const equilibrium" works reasonably well in a large hotel, where the non-saturation assumption holds most of the time, while it fails to improve above the median case in a small one.

Among the analytically determined policies, Dynamic Programming is the best option. However, in the small-hotel saturated case, the simplifying assumptions that make the Dynamic Programming policy solvable (e.g., reservations in different days do not interfere) are too restrictive, and the policy does not achieve good results.

Parametric pricing policies meaningfully improve the revenue, particularly in the saturated case. Unlike the reactive optimizers, which show a consistently good performance, the CMA-ES optimizer appears quite fragile and provides very unstable policies, hardly usable in a realistic context.

5.2 Day-by-day comparison of pricing policies

In order to investigate the reason for the performance of the various pricing schemes, an analysis has been performed at the reservation level. Fig. 5 shows the mean proposed price for the reservations aggregated by check-in date. Consider the case of the 10-room, easily saturated hotel (top); the arrows report the constant price policies: observe that the dynamic programming based policy (solid line) follows the theoretical constant equilibrium price as a baseline, operating small non-negative corrections to it on the basis of request arrival history.



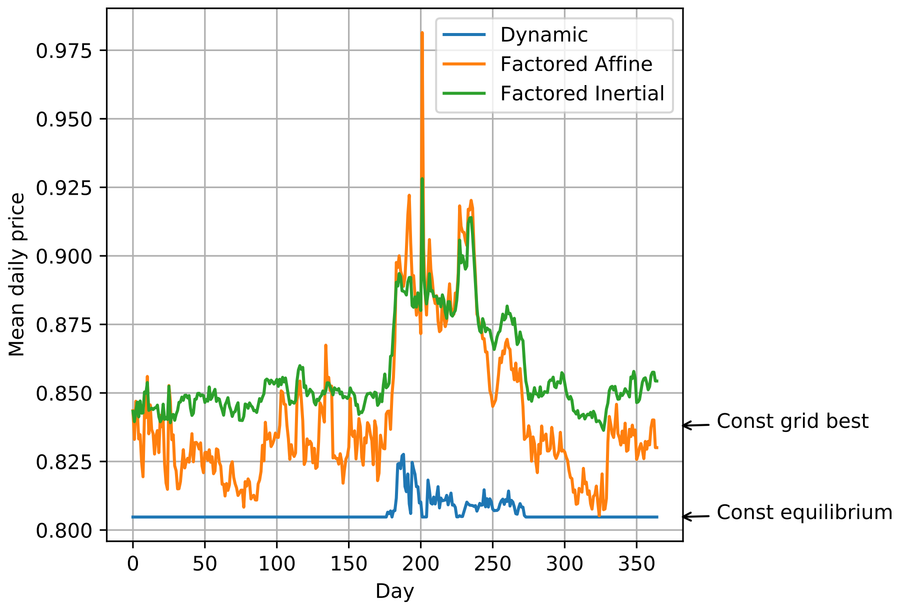


Fig. 5 Day-by-day comparison between pricing schemes on the same reservation sequences for a small hotel (top) and a large one (bottom).

On the other hand, the two shaker-optimized factorized pricing policies vary around the grid search-optimized constant price, with the most successful one (Affine shaker, dotted line) more akin to bolder, both positive and negative corrections. The CMA-ES-optimized policy hasn't been able to converge to usable values, and it is omitted from the plot for the sake of clarity.

The bottom chart of Fig. 5, referring to the 50-room hotel and the same request arrival sequence, clarifies how the Dynamic Programming strategy actually operates as a correction to the theoretical unsaturated equilibrium point. As long as the hotel is far from saturation (i.e., outside months 7–9, corresponding to days 180–269), the two policies are indistinguishable; only during the "hot" months, the Dynamic programming strategy introduces a small positive correction. Observe how, in this case, policies that are sensitive to occupancy (Dynamic programming, factorized) tend to be less effective with respect to the unsaturated case, and their prices are closer to the theoretical equilibrium value.

Fig. 6 allows us to compare the day-by-day effect of different pricing policies. In order to make plots readable in spite of the significant random noise due to request distribution, all data have been exponentially smoothed with a .95 persistence factor. The top row shows that the local search factored techniques consistently outperform the dynamic programming approach, in particular when the request rate is high (days 180–269). The middle row shows how the improvement in performance doesn't stem from more sales: in fact, average room occupancy is lower for the factored techniques with respect to dynamic programming. What is lost in terms of room occupation is gained in efficiency by an improved control of the selling price. As a consequence, the potential for higher income, measured as the number of refused proposals (capped at the actual number of available rooms) multiplied by the average price, as shown in the bottom row, is higher for the factored policies.

5.3 Simulation times

The simulator has been implemented as a single-threaded C++ application and tested on a 3.2GHz Intel Core i5 CPU with 16GB RAM.

Given the parameters chosen for the simulation, and shown in Fig. 2, a reservation set is composed by about 5100 reservation requests. The 10000 evaluations used to train the "Factored" pricing model require about 15 minutes, providing a throughput of more than 50000 reservation requests per second.

Throughputs with all tested pricing policies are very similar. In fact, while the Dynamic Programming policy initially requires a relatively large number of computations to compute all entries (23) of the optimal price tables for different values of expected arrivals N_r and mean advance λ, tables are stored and their computation times are amortized throughout the whole simulation.

6 Conclusions

In this paper we introduced a general framework for the assessment of pricing policies in a hotel revenue management setting with event-based simulations driven by a reservation generation model and a user acceptance model. This framework is intended to facilitate the experimental evaluation of pricing policies under controlled and repeatable conditions, the implementation of "what-if" scenarios for hotel managers, and the training of paramet-

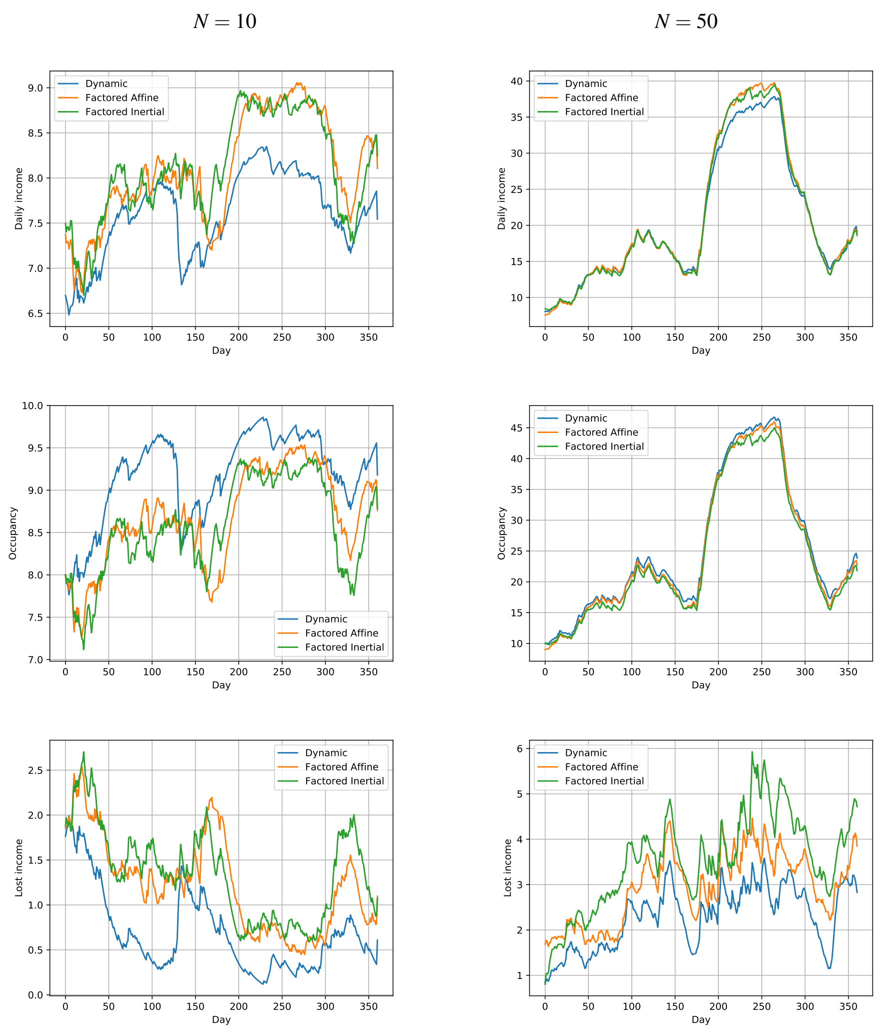


Fig. 6 Smoothed-out day-by-day effect of pricing policies on income (top), room occupancy (middle) and potentially lost income (bottom) for a small hotel (left) and a large one (right)

ric policies when a massive number of evaluations is required by a heuristic optimization scheme.

The results show that the proposed simulation environment is effective in training and comparing pricing policies and requires relatively little CPU time for reasonable simulated timespans. Our results suggest that heuristic policies, whose parameters are trained on separate request sets, can be more effective on average than analytically determined exact policies, due to the simplifications required to make them analytically tractable, but which risk making them not very realistic.

In the future evolution of the research we plan to explore the machine learning part by considering different and more refined demand forecast and flexible pricing schemes, and by adding support for online optimization [20]. On the analytical side, the high dimensionality of more realistic settings will be treated by resorting to stochastic and approximate dynamic programming [19].

In addition, we plan to consider more complex models by considering different room pools, multiple customer profiles with different choice models taking substitution effects into account, and extending exact models to more general assumptions.

References

1. Battiti, R., Brunato, M.: Reactive Search Optimization: Learning while Optimizing. Handbook of Metaheuristics **146**, 543–571 (2010)
2. Battiti, R., Brunato, M.: The LION way. Machine Learning *plus* Intelligent Optimization. LIONlab, University of Trento, Italy (2018). URL `http://intelligent-optimization.org/LIONbook/`
3. Battiti, R., Tecchiolli, G.: Learning with first, second, and no derivatives: a case study in high energy physics. Neurocomputing **6**, 181–206 (1994)
4. Bayoumi, A.E.M., Saleh, M., Atiya, A.F., Aziz, H.A.: Dynamic pricing for hotel revenue management using price multipliers. Journal of Revenue and Pricing Management **12**(3), 271–285 (2013)
5. Bellman, R.E.: Adaptive control processes: a guided tour, vol. 2045. Princeton university press (2015)
6. Belobaba, P.P.: Or practice—application of a probabilistic decision model to airline seat inventory control. Operations Research **37**(2), 183–197 (1989)
7. Bertsimas, D., De Boer, S.: Simulation-based booking limits for airline revenue management. Operations Research **53**(1), 90–106 (2005)
8. den Boer, A.V.: Dynamic pricing and learning: historical origins, current research, and new directions. Surveys in operations research and management science **20**(1), 1–18 (2015)
9. den Boer, A.V., Zwart, B.: Simultaneously learning and optimizing using controlled variance pricing. Management science **60**(3), 770–783 (2013)
10. Brunato, M., Battiti, R.: RASH: A self-adaptive random search method. In: C. Cotta, M. Sevaux, K. Sörensen (eds.) Adaptive and Multilevel Metaheuristics, *Studies in Computational Intelligence*, vol. 136. Springer (2008)
11. Brunelli, R., Tecchiolli, G.: On random minimization of functions. Biol. Cybern. **65**, 501–506 (1991)
12. Brunelli, R., Tecchiolli, G.: Stochastic minimization with adaptive memory. J. of Computational and applied mathematics **57**, 329–343 (1995)
13. Chaneton, J.M., Vulcano, G.: Computing bid prices for revenue management under customer choice behavior. Manufacturing & Service Operations Management **13**(4), 452–470 (2011)
14. Gallego, G., Van Ryzin, G.: Optimal dynamic pricing of inventories with stochastic demand over finite horizons. Management science **40**(8), 999–1020 (1994)
15. Glover, F., Glover, R., Lorenzo, J., McMillan, C.: The passenger-mix problem in the scheduled airlines. Interfaces **12**(3), 73–80 (1982)
16. Hansen, N., Ostermeier, A.: Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In: Evolutionary Computation, 1996., Proceedings of IEEE International Conference on, pp. 312–317. IEEE (1996)
17. Klein, R.: Network capacity control using self-adjusting bid-prices. OR Spectrum **29**(1), 39–60 (2007)
18. Papadimitriou, C.H.: Computational complexity. John Wiley and Sons Ltd. (2003)
19. Powell, W.B.: Approximate Dynamic Programming: Solving the curses of dimensionality, *Wiley Series in Probability and Statistics*, vol. 703. John Wiley & Sons (2007)
20. Powell, W.B., Ryzhov, I.O.: Optimal learning, *Wiley Series in Probability and Statistics*, vol. 841. John Wiley & Sons (2012)
21. Robinson, L.W.: Optimal and approximate control policies for airline booking with sequential nonmonotonic fare classes. Operations Research **43**(2), 252–263 (1995)
22. van Ryzin, G., Vulcano, G.: Computing virtual nesting controls for network revenue management under customer choice behavior. Manufacturing & Service Operations Management **10**(3), 448–467 (2008)
23. Talluri, K.T., Van Ryzin, G.J.: The theory and practice of revenue management, vol. 68. Springer Science & Business Media (2006)
24. Van Ryzin, G., Vulcano, G.: Simulation-based optimization of virtual nesting controls for network revenue management. Operations Research **56**(4), 865–880 (2008)
25. Williamson, E.L.: Airline network seat inventory control: Methodologies and revenue impacts. Ph.D. thesis, Massachusetts Institute of Technology (1992)

A Optimal prices for sigmoidal acceptance

This appendix describes the steps to obtain the optimal prices described in the paper.

A.1 Non-saturated equilibrium price

Here is the derivation of the optimal price value (16) described in Section 4.1.2 in the hypothesis that infinite rooms are available.

We want to find the stationary point of $u \cdot p_a(u)$, where u is the price and $p_a(u)$ is the corresponding acceptance probability (15):

$$\frac{d}{du}\big(u \cdot p_a(u)\big) = 1 - \sigma\left(\frac{u-\mu}{\sigma}\right) - \frac{1}{\eta} u \sigma'\left(\frac{u-\mu}{\eta}\right); \tag{24}$$

by applying the substitutions

$$s = \frac{u-\mu}{\eta}, \quad \beta = \frac{\mu}{\eta},$$

and reminding the identity $\sigma'(s) = \sigma(s)\big(1-\sigma(s)\big)$, we obtain

$$(\beta + s)\sigma(s)^2 - (\beta + s + 1)\sigma(s) + 1 = 0. \tag{25}$$

After replacing the sigmoid function definition, multiplying by $(1+e^{-s})^2$ and simplifying, we are left with

$$(s + \beta - 1)e^s = 1$$

and, multiplying by $e^{\beta-1}$,

$$(s + \beta - 1)e^{s+\beta-1} = e^{\beta-1}. \tag{26}$$

Let us observe that this equation is in the form $xe^x = a$ for $a > -\pi/2$, whose solution can be analytically expressed as $x = W_0(a)$, where $W_0(\cdot)$ is the main branch of Lambert's function. We get

$$s + \beta - 1 = W_0(e^{\beta-1}).$$

By replacing the original variables, we finally obtain (16).

A.2 Dynamic programming optimal price

The derivation of the optimal price policy (23) described in Section 4.1.3 for the dynamic programming technique follows the same steps outlined above.

After replacing (15) into (22), let us perform the following variable substitutions and quantity replacements:

$$s = \frac{u-\mu}{\eta}, \quad \beta = \frac{V_1 - V_2 + \mu}{\eta}, \tag{27}$$

after which we obtain (25), whose solution is, again, (26).

By replacing the original variables from (27), we obtain (23).