Online Stochastic Reservation Systems

Pascal Van Hentenryck, Russell Bent, Luc Mercier, and Yannis Vergados
Department of Computer Science, Brown University,
Providence, RI 02912, USA

September 13, 2006

Abstract

This paper considers online stochastic reservation problems, where requests come online and must be dynamically allocated to limited resources in order to maximize profit. Multi-knapsack problems with or without overbooking are examples of such online stochastic reservations. The paper studies how to adapt the online stochastic framework and the consensus and regret algorithms proposed earlier to online stochastic reservation systems. On the theoretical side, it presents a constant sub-optimality approximation of multi-knapsack problems, leading to a regret algorithm that evaluates each scenario with a single mathematical programming optimization followed by a small number of dynamic programs for one-dimensional knapsacks. It also proposes several integer programming models for handling cancellations and proves their equivalence. On the experimental side, the paper demonstrates the effectiveness of the regret algorithm on multi-knapsack problems (with and without overbooking) based on the benchmarks proposed earlier.

1 Introduction

In an increasingly interconnected and integrated world, online optimization problems are quickly becoming pervasive and raise new challenges for optimization software. Moreover, in most applications, historical data or statistical models are available, or can be learned, for sampling. This creates significant opportunities at the intersection of online algorithms, combinatorial and stochastic optimization, and machine learning and increasing attention has been devoted to these issues in a variety of communities (e.g., [10, 1, 6, 11, 9, 5, 8]).

This paper considers online stochastic reservation systems and, in particular, the online stochastic multi-knapsack problems introduced in [1]. Typical applications include, for instance, reservation systems for holiday centers and advertisement placements in web browsers. These problems differ from the stochastic routing and scheduling considered in, say, [10, 6, 9, 5] in that online decisions are not about selecting the best request to serve but rather about how best to serve a request.

The paper shows how to adapt our online stochastic framework, and the consensus and regret algorithms, to online stochastic reservation systems. Moreover, in order to instantiate the regret algorithm, the paper presents a constant-factor suboptimality approximation for multi-knapsack problems using one-dimensional knapsack problems. As a result, on multi-knapsack problems with or without overbooking, each online decision involves solving a mathematical program and a series of dynamic programs. The algorithms were evaluated on the multi-knapsack problems proposed in [1] with and without overbooking. The results indicate that the regret algorithm is particularly effective, providing significant benefits over heuristic, consensus, and expectation approaches. It also dominates an earlier algorithm proposed in [1] (which applies the best-fit heuristic within the expectation algorithm) as soon as the time constraints allows for 10 optimizations for each online decision or between each two online decisions. The results are particularly interesting in our opinion, because the consensus and regret algorithms have now been applied generically and

1

successfully to online problems in scheduling, routing, and reservation using, at their core, either constraint programming, mathematical programming, or dedicated polynomial algorithms.

The rest of the paper is organized as follows. Section 2 introduces online stochastic reservation problems in their simplest form and section 3 shows how to adapt our online stochastic algorithms for them. Section 4 discusses several ways of dealing with cancellations and section 5 presents the sub-optimality approximation. Section 6 describes the experimental results.

2 Online Stochastic Reservation Problems

2.1 The Offline Problem

The offline problem is defined in terms of n bins B and each bin $b \in B$ has a capacity C_b. It receives as input a set R of requests. Each request is typically characterized by its capacity and its reward, which may or may not depend on which bin the request are allocated to. The goal is to find an assignment of a subset $T \subseteq R$ of requests to the bins satisfying the problem-specific constraints and maximizing the objective function.

The Multi-Knapsack Problem   The multi-knapsack problem is an example of a reservation problem. Here each request r is characterized by a reward w_r and a capacity c_r. The goal is to allocate a subset T of the requests R to the bins B so that the capacities of the bins are not exceeded and the objective function $w(T) = \sum_{r \in T} w_r$ is maximized. A mathematical programming formulation of the problem associates witch each request r and bin b a binary variable $x[r, b]$ whose value is 1 when the request is allocated to bin b and 0 otherwise. The integer program can be expressed as:

$$
\max \quad \sum_{r \in R, b \in B} w_r\, x_r^b
$$

$$
\text{such that}
$$

$$
\sum_{b \in B} x_r^b \leq 1 \quad (r \in R)
$$
$$
\sum_{r \in R} c_r\, x_r^b \leq C_b \quad (b \in B)
$$
$$
x_r^b \in \{0, 1\} \quad (r \in R, b \in B)
$$

The Multi-Knapsack Problem with Overbooking   In practice, many reservation systems allow for over-booking. The multi-knapsack problem with overbooking allows the bin capacities to be exceeded but over-booking is penalized in the objective function. To adapt the mathematical-programming formulation above, it suffices to introduce a nonnegative variable y^b representing the excess for each bin b and to introduce a penalty term $\alpha \times y^b$ in the objective function. The integer programming model now becomes

$$
\max \quad \sum_{r \in R, b \in B} w_r\, x_r^b - \sum_{b \in B} \alpha\, y^b
$$
$$
\text{such that} \quad \sum_{b \in B} x_r^b \leq 1 \quad (r \in R)
$$
$$
\sum_{r \in R} c_r\, x_r^b \leq C_b + y^b \quad (b \in B)
$$
$$
x_r^b \in \{0, 1\} \quad (r \in R, b \in B)
$$
$$
y^b \geq 0 \quad (b \in B)
$$

This is the offline problem considered in [1].

Compact Formulations   When requests come from specific types (defined by their rewards and capacities, more compact formulations are desirable. Requests of the same type are equivalent and the same variables should be used for all of them. This avoids introducing symmetries in the model, which may significantly slow the solvers down. Assuming that there are $|K|$ types and there are R_k requests of type k ($k \in K$), the multi-knapsack problem then becomes

$$\max \quad \sum_{k \in K, b \in B} w_k x_k^b$$

such that

$$\sum_{b \in B} x_k^b \leq R_k \quad (k \in K)$$
$$\sum_{k \in K} c_k x_k^b \leq C_b \quad (b \in B)$$
$$x_k^b \geq 0 \quad (k \in K, b \in B),$$

where variable x_k^b represents the number of requests of type k assigned to bin b. A similar formulation may be used for the overbooking case as well.

Generic Formalization   To formalize the online algorithms precisely and generically, it is convenient to assume the existence of a dummy bin $\perp$ with infinite capacity to assign the non-selected requests and to use $B_\perp$ to denote $B \cup \{\perp\}$. A solution σ can then be seen as a function $R \to B_\perp$. The objective function can be specified by a function $\mathcal{W}$ over assignments and the problem-specific constraints can be specified as a relation over assignments giving us the problem $\max_{\sigma: \mathcal{C}(\sigma)} \mathcal{W}(\sigma)$. We use $\sigma[r \leftarrow b]$ to denote the assignment where r is assigned to bin b, i.e.,

$$\sigma[r \leftarrow b](r) \quad = b$$
$$\sigma[r \leftarrow b](r') \quad = \sigma(r') \quad \text{if } r' \neq r.$$

and $\sigma \downarrow R$ to denote the assignment where the requests in R are now unassigned, i.e.,

$$(\sigma \downarrow R)(r) \quad = \perp \quad \text{if } r \in R$$
$$(\sigma \downarrow R)(r) \quad = \sigma(r) \quad \text{if } r \notin R.$$

Finally, we use $\sigma_\perp$ to denote the assignment satisfying $\forall r \in R : \sigma(r) = \perp$.

2.2   The Online Problem

In the online problem, the requests are not known a priori but are revealed online during the execution of the algorithm. For simplicity, we consider a time horizon $H = [1, h]$ and we assume that a single request arrives at each time $t \in H$. (It is easy to relax these assumptions). The algorithm thus receives a sequence of requests $\boldsymbol{\xi} = \langle \xi_1, \ldots, \xi_h \rangle$ over the course of the execution. At time i, the sequence $\boldsymbol{\xi_i} = \langle \xi_1, \ldots, \xi_i \rangle$ has been revealed, the requests $\xi_1, \ldots, \xi_{i-1}$ have been allocated in the assignment σ_{i-1} and the algorithm must decide how to serve request ξ_i. More precisely, step i produces an assignment $\sigma_i = \sigma_{i-1}[\xi_i \leftarrow b]$ that assigns a bin b to ξ_i keeping all other assignments fixed. The requests are assumed to be drawn from a distribution $\mathcal{I}$ and the goal is to maximize the expected value

$$\mathbf{E}_{\boldsymbol{\xi}}[\mathcal{W}(\sigma_\perp[\xi_1 \leftarrow b_1, \ldots, \xi_h \leftarrow b_h])$$

where the sequence $\boldsymbol{\xi} = \langle \xi_1, \ldots, \xi_h \rangle$ is drawn from $\mathcal{I}$.

The online algorithms have at their disposal a procedure to solve , or approximate, the offline problem, and the distribution $\mathcal{I}$. The distribution is a black-box available for sampling.[1] Practical applications often include severe time constraints on the decision time and/or on the time between decisions. To model this requirement, the algorithms may only use the optimization procedure $\mathcal{O}$ times at each time step.

It is interesting to contrast this online problem with those studied in [7, 5, 3]. In these applications, the key issue was to select which request to serve at each step. Moreover, in the stochastic vehicle routing applications, accepted requests did not have to be assigned a vehicle: the only constraint on the algorithm

---

[1]Our algorithms only require sampling and do not exploit other properties of the distribution which makes them applicable to many applications. Additional information on the distribution could also be beneficial but is not considered here.

ONLINEOPTIMIZATION($\boldsymbol{\xi}$)
1   $\sigma_0 \leftarrow \sigma_\perp$;
2   **for** $t \in H$ **do**
3     $b \leftarrow$ CHOOSEALLOCATION(σ_{t-1}, ξ_t);
4     $\sigma_t \leftarrow \sigma_{t-1}[\xi_t \leftarrow b]$;
5   **return** σ_h;

Figure 1: The Generic Online Algorithm

was the promise to serve every accepted request. The online stochastic reservation problem is different. The key issue is not which request to serve but rather whether and how the incoming request must be served. Indeed, whenever a request is accepted, it must be assigned a specific bin and the algorithm is not allowed to reshuffle the assignments subsequently.

The Generic Online Algorithm   The algorithms in this paper share the same online optimization schema depicted in Figure 1. They differ only in the way they implement function CHOOSEALLOCATION. The online optimization schema receives a sequence of online requests $\boldsymbol{\xi}$ and starts with an empty allocation (line 1). At each decision time t, the online algorithm considers the current allocation σ_{t-1} and the current request ξ_t and chooses the bin b to allocate the request (line 3), which is then included in the new assignment σ_t (line 4). The algorithm returns the last assignment σ_h whose value is $\mathcal{W}(\sigma_h)$ (line 5). To implement function CHOOSEALLOCATION, the algorithms have at their disposal two black-boxes:

1. a function OPTSOL(σ, R) that, given an assignment σ and a R of requests, returns an optimal allocation of the requests in R given the past decisions in σ. In other words, OPTSOL(σ, R) solves an offline problem where the decision variables for the requests in σ have fixed values.

2. a function GETSAMPLE(t) that returns a set of requests over the interval $[t, h]$ by sampling the arrival distribution.

To illustrate the framework, we specify a best-fit online algorithm as proposed in [1].

Best Fit (G):   This algorithm assigns the request ξ to a bin that can accommodate ξ and has the smallest capacity given the assignment σ:

CHOOSEALLOCATION-G(σ, ξ)
1   **return** $argmin(b \in B_\perp : \mathcal{C}(\sigma[\xi \leftarrow b]))\ C_b(\sigma)$;

where $C_b(\sigma)$ denotes the remaining capacity of the bin $b \in B_\perp$ in σ, i.e.,

$$C_b(\sigma) = C_b - \sum_{r \in R : \sigma(r) = b} c_r.$$

3   Online Stochastic Algorithms

This section reviews the various online stochastic algorithms. It starts with the expectation algorithm and shows how it can be adapted to incorporate time constraints.

Expectation (E):   Informally speaking, algorithm E generates future requests by sampling and evaluates each possible allocation against the samples. A simple implementation can be specified as follows:

CHOOSEALLOCATION-E(σ_{t-1}, ξ_t)
1   **for** $b \in B_\perp$ **do**
2      $f(b) \leftarrow 0$;
3   **for** $i \leftarrow 1 \ldots \mathcal{O}/|B_\perp|$ **do**
4      $R_{t+1} \leftarrow$ GETSAMPLE$(t + 1)$;
5   **for** $b \in B_\perp : \mathcal{C}(\sigma_{t-1}[\xi_t \leftarrow b])$ **do**
6         $\sigma^* \leftarrow$ OPTSOL$(\sigma_{t-1}[\xi_t \leftarrow b], R_{t+1})$;
7         $f(b) \leftarrow f(b) + \mathcal{W}(\sigma^*)$;
8   **return** $argmax(b \in B_\perp)\ f(b)$;

Lines 1-2 initialize the evaluation $f(b)$ of each request b. The algorithm then generates $\mathcal{O}/|B_\perp|$ samples of future requests (lines 3–4). For each such sample, it successively considers each available bin b that can accommodate the request ξ given the assignment σ_{t-1} (line 5). For each such bin b, it schedules ξ_t in bin b and applies the optimization algorithm using the sampled requests R_{t+1} (line 6). The evaluation of bin b is incremented in line 7 with the weight of the optimal assignment σ^*. Once all the bin allocations are evaluated over all samples, the algorithm returns the bin b with the highest evaluation. Algorithm E performs $\mathcal{O}$ optimizations but uses only $\mathcal{O}/|B_\perp|$ samples. When $\mathcal{O}$ is small (due to the time constraints), each request is only evaluated with respect to a small number of samples and algorithm E does not yield much information. To cope with tight time constraints, two approximations of E, consensus and regret, were proposed.

Consensus (C):   The consensus algorithm C was introduced in [7] as an abstraction of the sampling method used in online vehicle routing [6]. Its key idea is to solve each sample once and thus to examine $\mathcal{O}$ samples instead of $\mathcal{O}/|B_\perp|$. More precisely, instead of evaluating each possible bin at time t with respect to each sample, algorithm C executes the optimization algorithm once per sample. The bin to which request ξ is allocated in optimal solution σ^* is credited $\mathcal{W}(\sigma^*)$ and all other bins receive no credit. Algorithm C can be specified as follows:

CHOOSEALLOCATION-C(σ_{t-1}, ξ_t)
1   **for** $b \in B_\perp$ **do**
2      $f(b) \leftarrow 0$;
3   **for** $i \leftarrow 1 \ldots \mathcal{O}$ **do**
4      $R_t \leftarrow \{\xi_t\} \cup$ GETSAMPLE$(t + 1)$;
5      $\sigma^* \leftarrow$ OPTSOL(σ_{t-1}, R_t);
6      $f(\sigma^*(\xi_t)) \leftarrow f(\sigma^*(\xi_t)) + \mathcal{W}(\sigma^*)$;
7   **return** $argmax(b \in B_\perp)\ f(b)$;

The core of the algorithm are once again lines 4–6. Line 4 defines the set R_t of requests that now includes ξ_t in addition to the sampled requests. Line 5 calls the optimization algorithm with σ_{t-1} and R_t. Line 6 increments only the bin $\sigma^*(\xi_t)$ The main appeal of Algorithm C is its ability to avoid partitioning the available samples between the requests, which is a significant advantage when $\mathcal{O}$ is small and/or when the number of bins is large. Its main limitation is its *elitism*. Only the best allocatation is given some credit for a given sample, while other bins are simply ignored.

Regret (R): The regret algorithm R is the recognition that, in many applications, it is possible to estimate the loss of sub-optimal allocations (called regrets) quickly. In other words, once the optimal solution σ^* of a scenario is computed, algorithm E can be approximated with one optimization [5, 2].

Definition 1 (Regret). Let σ be an assignment, R be a set of requests, r be a request in R, and b be a bin. The regret of a bin allocation $r \leftarrow b$ wrt σ and R, denoted by $\text{REGRET}(\sigma, R, r \leftarrow b)$, is defined as

$$ \mid \mathcal{W}(\text{OPTSOL}(\sigma, R)) - \mathcal{W}(\text{OPTSOL}(\sigma[r \leftarrow b], R \setminus \{r\}))) \mid. $$

Definition 2 (Sub-Optimality Approximation). Let σ be an assignment, R be a set of requests, r be a request in R, and b be a bin. Assume that algorithm $\text{OPTSOL}(\sigma, R)$ runs in time $O(f_o(R))$. A sub-optimatily approximation runs in time $O(f_o(R))$ and, given the solution $\sigma^* = optSol(\sigma, R)$, returns, for each bin $b \in B_\perp$, an approximation $\text{SUBOPT}(\sigma^*, \sigma, R, r \leftarrow b)$ to all regrets $\text{REGRET}(\sigma, R, r \leftarrow b)$ such that

$$ \mathcal{W}(\text{OPTSOL}(\sigma[r \leftarrow b], R \setminus \{r\}))) \leq c \left(\mathcal{W}(\text{OPTSOL}(\sigma[r \leftarrow b], R \setminus \{r\}))) - \text{SUBOPT}(\sigma^*, \sigma, R, r \leftarrow b) \right) $$

for some constant $c \geq 1$.

Intuitively, the $|B_\perp|$ regrets must not take more time than the optimization. We are ready to present the regret algorithm R:

CHOOSEALLOCATION-R (σ_{t-1}, ξ_t)
1   **for** $b \in B_\perp$ **do**
2     $f(b) \leftarrow 0$;
3   **for** $i \leftarrow 1 \ldots \mathcal{O}$ **do**
4     $R_t \leftarrow \{\xi_t\} \cup \text{GETSAMPLE}(t+1)$;
5     $\sigma^* \leftarrow \text{OPTSOL}(\sigma_{t-1}, R_t)$;
6     $f(\sigma^*(\xi_t)) \leftarrow f(\sigma^*(\xi_t)) + \mathcal{W}(\sigma^*)$;
7   **for** $b \in B_\perp \setminus \{\sigma(\xi_t) : \mathcal{C}(\sigma_{t-1}[\xi_t \leftarrow b])\}$ **do**
8       $f(b) \leftarrow f(b) + (\mathcal{W}(\sigma^*) - \text{SUBOPT}(\sigma^*, \sigma_{t-1}, R_t, \xi_t \leftarrow b))$;
9   **return** $argmax(b \in B_\perp) \, f(b)$;

Its basic organization follows algorithm C. However, instead of assigning some credit only to the bin selected by the optimal solution, algorithm R (lines 7-8) uses the sub-optimality approximation to compute, for each available allocation $\xi_t \leftarrow b$, an approximation of the best solution that allocates ξ_t to b. Hence every available bin is given an evaluation for every sample at time t for the cost of a single optimization (asymptotically). Observe that algorithm R performs $\mathcal{O}$ optimizations at time t.

Precomputation Many reservation systems require immediate responses to requests, giving only limited time to the online algorithm for decision making. However, as is the case in vehicle routing, there is time between decisions to generate scenarios and optimize them. This idea can be accommodated in the framework by separating the optimization phase from the decision-making phase in the online algorithm. This is especially attractive for consensus and regret where each scenario is solved exactly once. Details on this separation can be found in [4] in the context of the original framework.

4   Cancellations

Most reservation systems allow requests to be cancelled after they are accepted. The online stochastic framework can accommodate cancellations by simple enhancements to the generic online algorithm and the

ONLINEOPTIMIZATION$(\boldsymbol{\xi}, \boldsymbol{\zeta})$
1   $\sigma_0 \leftarrow \sigma_\perp$;
2   **for** $t \in H$ **do**
3      $\sigma_{t-1} \leftarrow \sigma_{t-1} \downarrow \zeta_t$;
4      $b \leftarrow$ CHOOSEALLOCATION(σ_{t-1}, ξ_t);
5      $\sigma_t \leftarrow \sigma_{t-1}[\xi_t \leftarrow b]$;
6   **return** σ_h;

Figure 2: The Generic Online Algorithm with Cancellations

CHOOSEALLOCATION-C(σ_{t-1}, ξ_t)
1   **for** $b \in B_\perp$ **do**
2      $f(b) \leftarrow 0$;
3   **for** $i \leftarrow 1 \ldots \mathcal{O}$ **do**
4      $\langle R_{t+1}, Z_{t+1} \rangle \leftarrow$ GETSAMPLE$(t+1)$;
5      $\sigma^* \leftarrow$ OPTSOL$(\sigma_{t-1} \downarrow Z_{t+1}, \{\xi_t\} \cup R_{t+1})$;
6      $f(\sigma^*(\xi_t)) \leftarrow f(\sigma^*(\xi_t)) + \mathcal{W}(\sigma^*)$;
7   **return** $argmax(b \in B_\perp) \ f(b)$;

Figure 3: The Consensus Algorithm with Cancellations

sampling procedure. It suffices to assume that an (often empty) set of cancellations ζ_t is revealed at step t in addition to the request ξ_t and that the function GETSAMPLE return pairs $\langle R, Z \rangle$ of future requests R and cancellations Z. Figure 2 presents a revised version of the generic online algorithm: its main modification is in line 3 which removes the cancellations ζ_t from the current assignment σ_{t-1} before allocating a bin to the new request.

Figure 3 shows the consensus algorithm with cancellations, illustrating the enhanced sampling procedure (line 4) and how cancellations are taken into account when calling the optimization. The resulting multi-knapsack is optimistic in that it releases the capacities of the cancellations at time t, although they may occur much later. A pessimistic multi-knapsack may be obtained by replacing line 5 in Figure 3 by

$$\sigma^* \leftarrow \text{OPTSOL}(\sigma_{t-1}, \{\xi_t\} \cup R_{t+1});$$

where the capacities freed by future cancellations are not restored. It is however possible to specify the real offline problem in presence of cancellations, which is called the multi-period/multi-knapsack problem in this paper. The rest of this section studies various integer-programming formulations of this problem.

4.1   The Multi-Period/Multi-Knapsack Problem

The multi-period/multi-knapsack problem is a generalization of the multi-knapsack problem in which requests arrive at various times and the capacities of the bins may increase at specific times. The capacity constraints must be respected at all times, i.e., a request can only be assigned to a bin if the bin can accommodate the request upon arrival. The complete input of the problem can be specifies as follows:

- A set B of bins.

- A set K of request types, a request of type k having a capacity c_k and a reward w_k.

- Time points: $0 = t_0 < t_1 < \cdots < t_M < t_{M+1} = h$. The time points correspond to the start time (t_0), the end time (t_{M+1}), or a capacity increase for a bin (t_k for $m = 1, \ldots, M$).

- Time points for bin b: $0 = t_0^b < \cdots < t_{M_b}^b < t_{M_b+1}^b = h$; for each $m \in \{1, \ldots, M\}$, there is exactly one b and one p such that $t_m = t_p^b$. In other words, the t_m's are obtained by merging the t_p^b's.

- Capacity for bin b: $C_0^b < \cdots < C_{M_b}^b$, where C_p^b is the capacity of bin b on the time interval $[t_p^b, t_{p+1}^b)$ ($0 \leq p \leq M_b$).

- For $m \in \{0, \ldots, M\}$, and $k \in K$, there are $R_{m,k}$ requests of type k arriving between t_m and t_{m+1}.

4.2 A Natural Model

The natural model is based upon the observation that the bin capacities do not change before the next capacity increase. Hence, it is sufficient to post the capacity constraints for a bin just before its capacity increases. The model thus features a decision variable $x_{m,k}^b$ for each bin b, time interval m, and request type k: the variable represents the number of requests of type k assigned to bin b during the time interval (t_m, t_{m+1}). There are thus $(M+1)|B||K|$ variables. There are $M + |B|$ capacity constraints: one for each time t_m ($m \in \{1, \ldots, M\}$) and $|B|$ for the deadline (constraints of type 2). There are also $|K|$ availability constraints for each time interval in order to bound the number of requests of each type that can be selected during the interval. The model (IP_1) can thus be stated as:

$$\text{Maximize} \sum_{b,m,k} w_k\, x_{m,k}^b \tag{1}$$

(IP_1) Subject to:

$$\forall b \in B, p \in \{0, \ldots, M_i\} : \quad \sum_{k \in K} \sum_{m \mid t_m \leq t_p^b} c_k\, x_{m,k}^b \leq C_p^b \tag{2}$$

$$\forall m \in \{0, \ldots, M\}, k \in K : \quad \sum_{b \in B} x_{m,k}^b \leq R_{m,k} \tag{3}$$

Model (IP_1) contains many variables and may exhibit many symmetries. In the context of online reservation systems, experimental results indicated that this multi-period/multi-knapsack model cannot be used to obtain a fair comparison with the offline one-period model as it takes a significant time to reach the same accuracy.

4.3 An Improved Model

The key idea underlying the improve model (IP_2) is to reduce the number of variables by considering only the time intervals relevant to each bin. More precisely, model (IP_2) uses a decision variable $y_{p,k}^b$ in (IP_2) to represent the number of requests of type k assigned to bin b on interval $[t_p^b, t_{p+1}^b)$. In other words, variable $y_{p,k}^b$ corresponds to the sum of the variables $x_{s,k}^b, x_{s+1,k}^b, \ldots, x_{e-1,k}^b$ where t_s and t_e are the unique time points satisfying $t_s = t_p^b$ and $t_e = t_{p+1}^b$, that is

$$y_{p,k}^b = x_{s,k}^b + x_{s+1,k}^b + \ldots + \ldots, x_{e-1,k}^b. \tag{4}$$

Figure 5(a) depicts the relationship between these variables visually. There are $|K| \left(\sum_{b \in B} (M_b + 1) \right)$ variables in (IP_2) or, equivalently, $|K||B| + |K|M$ variables since $M = \sum_b M_b$.

The capacity constraints (6) are mostly similar but only use the intervals pertinent to the request type. The availability constraints (7) are however harder to express and more numerous. The idea is to consider

FROMYTOX(C, R, y)

1   $x \leftarrow 0;$
2   **while** $\exists b, p \mid y_p^b \neq 0$ **do**
3      $(b, p) \leftarrow \text{argmin} \left\{ t_{p+1}^b \mid y_p^b \neq 0 \right\};$
4      $s \leftarrow$ the unique index such that $t_a = t_p^b;$
5      $e \leftarrow$ the unique index such that $t_b = t_{p+1}^b;$
6      $i \leftarrow s$
7   **while** $y_p^b \neq 0$ **do**
8   **if** $t_i \geq t_s$ **then**
9   **return** FAILURE;
10       $\delta \leftarrow \min(y_p^b, R_i);$
11       $y_p^b \leftarrow y_p^b - \delta;$
12       $R_c \leftarrow R_c - \delta;$
13       $x_i^b \leftarrow \delta;$
14       $i \leftarrow i + 1;$
15   **return** $x;$

Figure 4: The Transformation from Model (IP_2) to Model (IP_1).

all pairs of time points (t_{m_1}, t_{m_2}) such that $m_1 < m_2$ and to make sure that the variables $y_{p,k}^b$ that can only consume requests of type k in the intervals $[t_{m_1}, t_{m_2})$ do not request more requests than available. There are thus $O(M^2|K|)$ availability constraints in (IP_2) instead of $O(M|K|)$ in (IP_1).

The model can thus be stated as follows:

$$(IP_2) \quad \begin{cases} \text{Maximize} \sum_{b,p,k} w_k y_{p,k}^b. & (5) \\[2mm] \text{Subject to:} \\ \forall b \in B, p \in \{0, \ldots, M_b\} : \quad \sum_{k \in K} \sum_{m \mid t_m^b \leq t_p^b} c_k \, y_{m,k}^b \leq C_p^b. & (6) \\[2mm] \forall 0 \leq m_1 < m_2 \leq M+1, k \in K : \quad \sum_{\substack{b \in B, p \\ t_{m_1} \leq t_p^b \\ t_{p+1}^b \leq t_{m_2}}} y_{p,k}^b \leq \sum_{m=m_1}^{m_2-1} R_{m,k} & (7) \end{cases}$$

4.4 Equivalence of the Models

Any solution to (IP_1) can be transformed into a solution to (IP_2): it suffices to use equation (4) to compute the values of the y variables. This section shows how to transform a solution to (IP_2) into a solution to (IP_1). First, observe that the transformation can consider each request type independently and derive the values of of variables $x_{s,k}^b, x_{s+1,k}^b, \ldots, \ldots, x_{e-1,k}^b$ from the value of the variable $y_{p,k}^b$. As a result, for simplicity, the rest of section omits the subscript k corresponding to the request type.

It remains to show how to derive the values of $x_s^b, x_{s+1}^b, \ldots, \ldots, x_{e-1}^b$ from the value of y_p^b. This transformation is depicted in algorithm FROMYTOX. The algorithm considers the variables $y_p^b \neq 0$ by increasing order of t_{p+1}^b, that is the endpoints of their time intervals. It greedily assigns the available requests to the variables $x_s^b, x_{s+1}^b, \ldots, x_{e-1}^b$ that correspond to y_p^b. Each iteration of lines 8–14 considers variables

9

Variable	(b) Input	(c) First pass	(d) Second pass	(e) Third pass	(f) Last pass, output
x_0^1	0	2	2	2	2
x_1^1	0	0	0	0	0
x_2^1	0	0	1	1	1
x_3^1	0	0	0	0	0
x_4^1	0	0	0	0	1
x_0^2	0	0	0	0	0
x_1^2	0	0	0	0	0
x_2^2	0	0	0	1	1
x_3^2	0	0	0	1	1
x_4^2	0	0	0	0	0
y_0^1	2	0	0	0	0
y_1^1	1	1	0	0	0
y_2^1	1	1	1	1	0
y_0^2	0	0	0	0	0
y_1^2	2	2	2	0	0
y_2^2	0	0	0	0	0
R_0	3	1	1	1	1
R_1	0	0	0	0	0
R_2	2	2	1	0	0
R_3	1	1	1	0	0
R_4	2	2	2	2	1

Figure 5: A Run of Algorithm FROMYTOX with a Feasible Input.

	(a) Input	(b) First pass	(c) Second pass	(d) Return "No Solution"
y^1	2 1 1	0 1 1	0 0 1	0 0 1
y^2	0 2 0	0 2 0	0 2 0	0 1 ⚠ 0
R	3 0 1 1 2	1 0 1 1 2	1 0 0 1 2	1 0 0 0 2

Figure 6: A Run of Algorithm FROMYTOX on an Infeasible Input.

x_i^b, selects as many requests as possible from R_i (but not more than y_p^b), decreases R_i and y_p^b, and assigns x_i^b. The algorithm fails if, at time t_e, the value y_p^b has not been driven down to zero, meaning that there are too few requests to distribute y_p^b among $x_s^b, x_{s+1}^b, \ldots, \ldots, x_{e-1}^b$.

Observe that, if (IP_2) satisfies (6) and the transformation succeeds, then the assignments to the x variables satisfies the capacity constraints (2) because of line 10. It remains to show that a failure cannot occur when the constraints (7) are satisfied, meaning that lines 8–9 are redundant and that the algorithm always succeeds in transforming a solution to (IP_2) into a solution to (IP_1) when the availability constraints (7) are satisfied.

Figure 5 depicts a successful run of this algorithm. Part (a) depicts the variables and part (b) specifies the inputs, that is the assignment of the y variables. The remaining parts (c)–(f) depict the successive iterations of the algorithm. The variables are selected in the order y_0^1, y_1^1, y_1^2, and y_2^1. The available requests $R_0, \ldots, R_4$ are shown in below. Observe how the algorithm assigns the value of y_1^1 to x_2^1, since $R_1 = 0$.

Figure 6 depicts a failing run of the algorithm. During the third iteration, the program returns, because there are too few available requests to decrease y_1^2 to zero. That means that the instance with the updated values of R_2 violates the constraints (7) with $m_1 = 2, m_2 = 4$. In turn, this implies that the y assignment violates the constraints (7) on the original input with $m_1 = 1, m_2 = 4$. The figure also depicts how the proof will construct the violated constraint. The intervals represented by short-dashed arrows correspond to the y_p^b considered during each iteration of the outermost loop. The long-dashed arrows represent an interval violating the availability constraint after the iteration is completed. These two intervals are combined to obtain an interval (shown by the plain arrows) violating the availability constraints at the beginning of the iteration. To obtain this last interval, the proof combines the two "dashed" intervals as follows. Whenever the vector R has been modified during the iteration at a position included in the long-dashed interval, the plain interval is the union of the two dashed one (this is the case on figure 6(c)). Otherwise, the plain interval is the long-dashed one (this is the case on figure 6(b)).

Lemma 1. If algorithm FROMYTOX fails, there exist $0 \leq m_1 < m_2 \leq M$ violating constraint (7).

Proof. By induction on $\left| \left\{ (b,p) \, \middle| \, y_p^b \neq 0 \right\} \right|$. The base case is immediate. Assume that the lemma holds for i non-zero variables. We show that it holds for $i + 1$ non-zero variables. Let $y_{p_0}^{b_0}$ be the variable considered during the first iteration of the outer loop and choose $m_1' = s$ and $m_2' = e$, with s and e defined as in lines 4 and 5 of the algorithm.

10

Suppose the algorithm fails during the first iteration. Then there are fewer than y_p^b available requests in the interval $[t_{m_1}, t_{m_2})$ with $m_1 = m_1'$ and $m_2 = m_2'$ and the result holds.

Suppose now that the program fails in a subsequent iteration and let $\overline{R}, \overline{y}$ the values of the vectors R and y after the first iteration of the outer loop (line 3–14). That means that the algorithm would have failed with $\overline{y}$ and $\overline{R}$ as input. By induction, since $\left|\left\{(b,p) \,\middle|\, \overline{y}_p^b \neq 0\right\}\right| = i$, there exist m_1'' and m_2'' such that $\overline{y}$ and $\overline{R}$ violate constraint (7). There are two cases to consider.

case 1. If $\overline{R}_m = R_m$ for all $m_1'' \leq m < m_2''$, then the same interval $[t_{m_1''}, t_{m_2''})$ for which (7) was violated with $\overline{y}$ and $\overline{R}$ also violates the constraint with y and R. As a consequence, the result holds with $m_1 = m_1''$ and $m_2 = m_2''$.

case 2. Suppose there exists $m^\star$ such that $m_1'' \leq m^\star < m_2''$ and $\overline{R}_{m^\star} < R_{m^\star}$. First, because the inner loop modifies R only in the range $[m_1', m_2')$, the intervals $[m_1', m_2' - 1]$ and $[m_1'', m_2'' - 1]$ intersect and hence their union is also an interval. Denote this union by $[m_1, m_2 - 1]$ and observe that $m_2 = m_2''$ by line 3 of algorithm FROMYTOX. In addition, because the inner loop decreases R_m from left to right (i.e., by increasing values of m), we have $\overline{R}_m = 0$ for all m such that $m_1' \leq m < m_1''$ (otherwise the inner loop would have stopped before m and the first case would apply). This proves that $\sum_{m=m_1}^{m_2-1} \overline{R}_m = \sum_{m=m_1''}^{m_2''-1} \overline{R}_{m,}$. As a consequence,

$$\sum_{\substack{b,p \\ t_{m_1} \leq t_p^b \\ t_{p+1}^b \leq t_{m_2}}} y_p^b = y_{p0}^{b0} + \sum_{\substack{b,p \\ t_{m_1} \leq t_p^b \\ t_{p+1}^b \leq t_{m_2}}} \overline{y}_p^b \geq y_{p0}^{b0} + \sum_{\substack{b,p \\ t_{m_1''} \leq t_p^b \\ t_{p+1}^b \leq t_{m_2''}}} \overline{y}_p^b > y_{p0}^{b0} + \sum_{m=m_1''}^{m_2''-1} \overline{R}_m = y_{p0}^{b0} + \sum_{m=m_1}^{m_2-1} \overline{R}_m = \sum_{m=m_1}^{m_2-1} R_m.$$

and thus the constraint (7) is violated for the specified m_1 and m_2.

$\square$

The following proposition summarizes the results of this section.

Proposition 1. The models (IP_1) and (IP_2) have the same optimal objective value.

In practice, this last model is very satisfying. On the benchmarks used in the experimental section, model (IPb) is solved about 2.5 times slower than the corresponding (single-period) multi-knapsack (for the same accuracy).

5   The Suboptimality Approximation

This section describes a sub-optimality algorithm approximating multi-knapsack problems within a constant factor. Given a set of requests R, a request $r \in R$, and an optimal solution σ^* to the multi-knapsack problem, the sub-optimality algorithm must return approximations to the regrets of allocating r to bin $b \in B_\perp$. The sub-optimality algorithm must run within the time taken by a constant number of optimizations.

The key idea behind the suboptimality algorithm is to solve a small number of one-dimensional knapsack problems (which takes pseudo-polynomial time). There are two main cases to study: either request r is allocated to a bin in B in solution σ^* or it is not allocated (that is, it is allocated to $\perp$). In the first case, the algorithm must approximate the optimal solutions in which r is allocated to other bins (procedure REGRET-SWAP) or not allocated (procedure REGRET-SWAP-OUT). In the second case, the request must be swapped in all the bins (procedure REGRET-SWAP-IN). The rest of this section presents algorithms for the non-overbooking case; they generalize to the overbooking case.

REGRET-SWAP$(i, 1, 2)$

```
 1   A ← bin(1, σ*) ∪ bin(2, σ*) ∪ U(σ*) \ {i};
 2   if C₁ − cᵢ ≥ C₂ then
 3       bin(1, σᵃ) ← knapsack(A, C₁ − cᵢ) ∪ {i};
 4       bin(2, σᵃ) ← knapsack(A \ bin(1, σᵃ), C₂);
 5   else
 6       bin(2, σᵃ) ← knapsack(A, C₂);
 7       bin(1, σᵃ) ← knapsack(A \ bin(2, σᵃ), C₁ − cᵢ) ∪ {i};
 8   e ← argmax(r ∈ bin(1, σ*) \ bin(1..2, σᵃ) : cᵣ > max(C₁ − cᵢ, C₂)) cᵣ;
 9   if e exists & wₑ > max(w(bin(1, σᵃ)), w(bin(2, σᵃ))) then
10       j ← argmax(j ∈ 3..n) Cⱼ;
11       bin(j, σᵃ) ← knapsack(bin(j, σᵃ) ∪ {e}, Cⱼ);
```

Figure 7: The Suboptimality Algorithm for the Knapsack Problem: Swapping i from Bin 2 to Bin 1.

Since the names of the bins have no importance, we assume that they are numbered $1..n$. Moreover, without loss of generality, we formalize the algorithms to move request i from bin 2 to bin 1, to swap request i out of bin 1, and to swap request i into bin 1. We use σ^* to represent the optimal solution to the multi-knapsack problem, σ^s to denote the optimal solution in which request i is assigned to bin 1 (REGRET-SWAP and REGRET-SWAP-OUT) or is not allocated (REGRET-SWAP-IN), and σ^a to denote the sub-optimality approximation. We also use $bin(b, \sigma)$ to denote the requests allocated to bin b and generalize the notation to sets of bins. The solution to the one-dimensional knapsack problem on R for a bin with capacity C is denoted by $knapsack(R, C)$. We also use $c(R)$ to denote the sum of the capacities of the requests in R, $w(R)$ to denote the sum of the rewards of the requests in R, and $U(\sigma^*)$ the requests that are not allocated in the optimal solution σ^*.

Swapping a Request Between Two Bins Figure 7 depicts the algorithm to swap request i from bin 1 to bin 2. The key idea is to consider all requests allocated to bins 1 and 2 in σ^* and to solve two one-dimensional problems for bin 1 (without the capacity taken by request i) and bin 2. The algorithm always starts with the bin whose remaining capacity is largest. After solving these two one-dimensional knapsacks, if there exists a request $e \in bin(1, \sigma^*)$ not allocated in $bin(1..2, \sigma^a)$ and whose value is higher than the values of these two bins, the algorithm solves a third knapsack problem to place this request in another bin if appropriate. This is important if request e is of high value but cannot be allocated in bin 1 due to the capacity taken by request i.

Theorem 3. *Algorithm* REGRET-SWAP *is a constant-factor approximation, that is, if σ^s be the sub-optimal solution and σ^a be the regret solution, there exists a constant $c \geq 1$ such that $w(\sigma^s) \leq c\, w(\sigma^a)$.*

Proof. Let σ^s be the sub-optimal solution, σ^a be the regret solution, and σ^* be the optimal solution. Consider the following sets

$$
\begin{aligned}
I_1 &= \sigma^s \cap \sigma^a & I_7 &= (bin(2, \sigma^s) \setminus \sigma^a) \cap bin(1, \sigma^*) \\
I_2 &= (bin(1, \sigma^s) \setminus \sigma^a) \cap U(\sigma^*) & I_8 &= (bin(2, \sigma^s) \setminus \sigma^a) \cap bin(2, \sigma^*) \\
I_3 &= (bin(2, \sigma^s) \setminus \sigma^a) \cap U(\sigma^*) & I_9 &= (bin(3..n, \sigma^s) \setminus \sigma^a) \cap bin(1, \sigma^*) \\
I_4 &= (bin(3..n, \sigma^s) \setminus \sigma^a) \cap U(\sigma^*) & I_{10} &= (bin(3..n, \sigma^s) \setminus \sigma^a) \cap bin(2, \sigma^*) \\
I_5 &= (bin(1, \sigma^s) \setminus \sigma^a) \cap bin(1, \sigma^*) & I_{11} &= (bin(1..n, \sigma^s) \setminus \sigma^a) \cap bin(3..n, \sigma^*) \\
I_6 &= (bin(1, \sigma^s) \setminus \sigma^a) \cap bin(2, \sigma^*).
\end{aligned}
$$

12

The suboptimal solution σ^s can be partitioned into $\sigma^s = \bigcup_{k=1}^{11} I_k$ and the proof shows that $w(I_k) \leq c_k\, w(\sigma^a)$ $(1 \leq k \leq 11)$ which implies that $w(\sigma^s) \leq c\, w(\sigma^a)$ for some constant $c = c_1 + \ldots c_{11}$. The proof of each inequality typically separates two cases:

A: $C_1 - c_i \geq C_2$;

B: $C_1 - c_i < C_2$.

Observe also that the proof that $w(I_1) \leq w(\sigma^a)$ is immediate. We now give the proofs for the remaining sets. In the proofs, C_1' denotes $C_1 - c_i$ and $K(E, C)$ is defined as follows:

$$K(E, C) = w(knapsack(E, C)).$$

$I_2.A$: By definition of I_2 and by definition of $bin(1, \sigma^a)$ in line 3,

$$K(I_2, C_1') \leq K(U(\sigma^*), C_1') \leq K(bin(1, \sigma^a), C_1') \leq w(\sigma^a).$$

$I_2.B$: By definition of I_2, $C_1' < C_2$, and by definition of $bin(2, \sigma^a)$ in line 6

$$K(I_2, C_1') \leq K(U(\sigma^*), C_1') \leq K(U(\sigma^*), C_2) \leq K(bin(2, \sigma^a), C_2) \leq w(\sigma^a).$$

$I_3.A$: By definition of I_3, $C_1' \geq C_2$, and by definition of $bin(1, \sigma^a)$ in line 3

$$K(I_3, C_2) \leq K(U(\sigma^*), C_2) \leq K(U(\sigma^*), C_1') \leq K(bin(1, \sigma^a), C_1') \leq w(\sigma^a).$$

$I_3.B$: By definition of I_3 and by definition of $bin(2, \sigma^a)$ in line 6

$$K(I_3, C_2) \leq K(U(\sigma^*), C_2) \leq K(bin(2, \sigma^a), C_2) \leq w(\sigma^a).$$

I_4 : Assume that $w(I_4) > w(\sigma^a)$. This implies

$$\begin{aligned} w(I_4) &> w(bin(1, \sigma^a)) + w(bin(2, \sigma^a)) + w(bin(3..n, \sigma^a)) \\ &> w(bin(3..n, \sigma^a)) > w(bin(3..n, \sigma^*)) \end{aligned}$$

which contradicts the optimality of σ^* since $I_4 \subseteq U(\sigma^*)$.

$I_5.A$: By definition of I_5 and line 3 of the algorithm

$$K(I_5, C_1') \leq K(bin(1, \sigma^*), C_1') \leq K(A, C_1') \leq w(bin(1, \sigma^a)) \leq w(\sigma^a).$$

$I_5.B$: By definition of I_5, $C_1' \geq C_2$, and line 6 of the algorithm

$$\begin{aligned} K(I_5, C_1') &\leq K(bin(1, \sigma^*), C_1') \leq K(bin(1, \sigma^*), C_2) \leq K(A, C_2) \\ &\leq K(bin(2, \sigma^a), C_2) \leq w(\sigma^a) \end{aligned}$$

$I_6.A$: By definition of I_6 and line 3 of the algorithm

$$K(I_6, C_1') \leq K(bin(2, \sigma^*) \setminus \{i\}, C_1') \leq K(bin(1, \sigma^a), C_1') \leq w(\sigma^a)$$

$I_6.B$: By definition of I_6 and line 6 of the algorithm.

$$K(I_6, C_1') \leq K(bin(2, \sigma^*) \setminus \{i\}, C_2) \leq K(bin(2, \sigma^a), C_2) \leq w(\sigma^a)$$

$I_7.A$: by definition of I_7, $C_2 \leq C_1'$, and line 3 of the algorithm,

$$K(I_7, C_2) \leq K(I_7, C_1') \leq K(bin(1, \sigma^*), C_1') \leq K(bin(1, \sigma^a), C_1') \leq w(\sigma^a).$$

$I_7.B$: By definition of I_7, $C_2 > C_1'$, and line 6 of the algorithm

$$K(I_7, C_2) \leq K(bin(1, \sigma^*), C_2) \leq K(bin(2, \sigma^a), C_2) \leq w(\sigma^a).$$

$I_8.A$: By definition of I_8, $C_2 \leq C_1'$, and line 3 of the algorithm

$$K(I_8, C_2) \leq K(I_8, C_1') \leq K(bin(2, \sigma^*), C_1') \leq K(bin(1, \sigma^a), C_1') \leq w(\sigma^a)$$

$I_8.B$: by definition of I_8, $C_2 > C_1'$, and line 6 of the algorithm,

$$K(I_8, C_2) \leq K(bin(2, \sigma^*), C_2) \leq K(bin(2, \sigma^a), C_2) \leq w(\sigma^a).$$

$I_9.A$: Consider

$$
\begin{aligned}
T &= knapsack(bin(1, \sigma^*), C_1'); \\
L &= bin(1, \sigma^*) \setminus T
\end{aligned}
$$

and let $e = \operatorname{argmax}_{e \in L} w_e$. By optimality of T, we know that $c(T) + c(e) > C_1'$ and, since $bin(1, \sigma^*) = T \cup L$, we have that $c(L \setminus \{e\}) < c_i$.

If $w_e \leq \max(w(bin(1, \sigma^a)), w(bin(2, \sigma^a)))$, then

$$
\begin{aligned}
w(I_9) &\leq w(T) + w(L \setminus \{e\}) + w_e \\
&\leq w(bin(1, \sigma^a)) + w(bin(2, \sigma^a)) + w_e \\
&\leq 2(w(bin(1, \sigma^a)) + w(bin(2, \sigma^a))) \leq 2w(\sigma^a).
\end{aligned}
$$

Otherwise, by optimality of $bin(1, \sigma^a)$ and $bin(2, \sigma^a)$, we have that

$$c(e) > C_1' \ \& \ c(e) > C_2$$

and the algorithm executes lines 10–11. If $c(e) \leq C_j$, then

$$
\begin{aligned}
w(I_9) &\leq w(T) + w(L \setminus \{e\}) + w_e \\
&\leq w(bin(1, \sigma^a)) + w(bin(2, \sigma^a)) + w(bin(j, \sigma^a)) \leq w(\sigma^a).
\end{aligned}
$$

Otherwise, if $c(e) > C_j$, $e \notin \sigma^s$ and

$$w(I_9) \leq w(T) + w(L \setminus \{e\}) \leq w(bin(1, \sigma^a)) + w(bin(2, \sigma^a)) \leq w(\sigma^a).$$

$I_9.B$: Consider

$$
\begin{aligned}
T &= knapsack(bin(1, \sigma^*), C_2); \\
L &= bin(1, \sigma^*) \setminus T
\end{aligned}
$$

and let $e = \operatorname{argmax}_{e \in L} w_e$. If $w(T) \geq w(L)$, we have that

$$w(bin(1, \sigma^*)) \leq 2w(T) \leq 2w(bin(2, \sigma^a)) \leq 2w(\sigma^a).$$

REGRET-SWAP-OUT$(i, 1)$
1   $A \leftarrow bin(1, \sigma^*) \cup U(\sigma^*) \setminus \{i\};$
2   $bin(1, \sigma^a) \leftarrow knapsack(A, C_1);$

Figure 8: The Suboptimality Algorithm for the Knapsack Problem: Swapping i out of Bin 1.

Otherwise, $c(L) > C_2$ by optimality of T and thus $c(L) > c_i$ since $C_2 \geq c_i$. By optimality of T, $c(T \cup \{e\}) > C_2 > C_1'$ and, since $bin(1, \sigma^*) = T \cup L$, it follows that $c(L \setminus \{e\}) \leq c_i$ Hence $w(L \setminus \{e\}) \leq w(T)$ by optimality of T and

$$w(I_9) \quad \leq \quad w(T) + w(L \setminus \{e\}) + w_e \leq 2w(T) + w_e \leq 2w(bin(2, \sigma^a)) + w_e.$$

If $w_e \leq w(bin(2, \sigma^a))$, $w(I_9) \leq 3w(bin(2, \sigma^a)) \leq 3w(\sigma^a)$ and the result follows. Otherwise, by optimality of $bin(2, \sigma^a)$, $c(e) > C_2 \geq C_1'$ and the algorithm executes lines 10–11. If $c(e) \leq C_j$, then

$$w(I_9) \quad \leq \quad 2w(bin(1, \sigma^a)) + w(bin(j, \sigma^a)) \leq w(\sigma^a).$$

Otherwise, if $c(e) > C_j$, $e \notin \sigma^s$ and

$$w(I_9) \quad \leq \quad w(T) + w(L \setminus \{e\}) \leq 2w(bin(2, \sigma^a)) \leq 2w(\sigma^a).$$

$I_{10}.A$: By definition of I_{10}, $C_1' \geq C_2$, and line 3 of the algorithm

$$w(I_{10}) \leq w(bin(2, \sigma^*)) - w(i) \leq w(bin(1, \sigma^a)) \leq w(\sigma^a).$$

$I_{10}.B$: By definition of I_{10} and by line 6 of the algorithm

$$w(I_{10}) \leq w(bin(2, \sigma^*)) - w(i) \leq w(bin(2, \sigma^a)) \leq w(\sigma^a).$$

I_{11} : By definition of the algorithm, $K(bin(3..n, \sigma^*)) \leq K(3..n, \sigma^a)$.

$\square$

Swapping a Request Out of a Bin   The algorithm to swap a request i out of bin 1 is depicted in Figure 8. It consists of solving a one-dimensional knapsack with the requests already in that bin and the unallocated requests. The proof is similar, but simpler, to the proof of Theorem 3.

Theorem 4. *Algorithm* REGRET-SWAP-OUT *is a constant-factor approximation.*

Swapping a Request Into a Bin   Figure 9 depicts the algorithm for swapping a request i in bin 1, which is essentially similar REGRET-SWAP but only uses one bin. It assumes that request i can be placed in at least two bins since otherwise a single additional optimization suffices to compute all the regrets. Once again, it solves a one-dimensional knapsack for bin 1 (after having allocated request i) with all the requests in $bin(1, \sigma^*)$ and the unallocated requests. If the resulting knapsack is of low quality (i.e., the remaining requests from $bin(1, \sigma^*)$ have a higher value than $bin(1, \sigma^a)$), REGRET-SWAP-IN solves an additional knapsack problem for the largest available bin. The proof is once again similar to the proof of Theorem 3.

Theorem 5. *Assuming that item i can be placed in at least two bins, Algorithm* REGRET-SWAP-IN *is a constant-factor approximation.*

REGRET-SWAP-IN$(i, 1)$
1   $A \leftarrow bin(1, \sigma^*) \cup U(\sigma^*)$;
2   $bin(1, R) \leftarrow knapsack(A, C_1 - c_i) \cup \{i\}$;
3   $L \leftarrow bin(1, \sigma^*) \setminus bin(1, \sigma^a)$;
4   **if** $w(L) > w(bin(1, \sigma^a))$ **then**
5       $j \leftarrow argmax(j \in 2..n) \, C_j$;
6       $bin(j, \sigma^a) \leftarrow knapsack(bin(j, \sigma^a) \cup L, C_j)$;

Figure 9: The Suboptimality Algorithm for the Knapsack Problem: Swapping i into Bin 1.

6   Experimental Results

6.1   The Instances

The experimental results use the benchmarks proposed in [1]. Requests are classified in k types. Each type is characterized by a weight, a value, two exponential distributions indicating how frequently requests of that type arrive and are cancelled, and an overbooking penalty. We generated ten instances based on the master problem proposed in [1]. The goal was to try to produce a diverse set of problems revealing strengths and weaknesses of the various algorithms. The ten problems are named (A-J) here. Problem A scales the master problem by doubling the weight and value of the request types in the master problem, as well as halving the number of items that arrive. Problem B further scales problem A by increasing the weight and value of the types. Problem C considers 7 types of items whose cost ratio takes the form of a bell shape. Problem D looks at the master problem and doubles the number of bins while dividing their capacity by 2. Problem E considers a version of the master problem with bins of variable capacity. Problem F depicts a version of the master problem whose items arrive three times as often and cancel three times as often. Problem G considers a much larger problem with 35 requests types who cost ratio is also shaped in a bell. Problem H is like problem G, the main difference is that the cost ratio shape is reversed. Problem I is a version of G with an extra bin. Problem J is a version of H with fewer bins.

The mathematical programs are solved with CPLEX 9.0 with a time limit of 10 seconds. The optimal solutions can be found within the time limit for all instances but I and J. Every instance is executed under various time constraints, i.e., $\mathcal{O} = 1, 5, 10, 25, 50$, or 100, and the results are the average of 10 executions. The default algorithm for cancellations uses the pessimistic multi-knapsack, which is slighly superior to the optimistic multi-knapsack.

It is important to highlight that, on the master problem and its variations, the best-fit heuristic performs quite well. On the offline problems, it is 5% off the optimum in the average and is never worse than 10% off. This will be discussed again when the regret algorithm is compared to earlier results.

6.2   Comparison of the Algorithms

Figure 10 describes the average profit (a) and loss (b) of the various online algorithms as a percentage of the optimal offline solution. The loss sums the weights of the rejected requests and the overbooking penalty (if any); it is often used in comparing online algorithms as it gives a sense of the "price" of uncertainty. The results clearly show the value of stochastic information as algorithms R, C, E recovers most of the gap between the online best-fit heuristic (G) and the offline optimum (which cannot typically be achieved in an online setting). Moreover, they show that algorithms R and C achieve excellent results even with small number of available optimizations (tight time constraints). In particular, algorithm R achieves about 89% of the offline optimum with only 10 samples and 91% with 50 optimizations. It also achieves a loss of 28% over the offline optimum for 25 optimizations and 34% for 10 optimizations. The regret algorithm
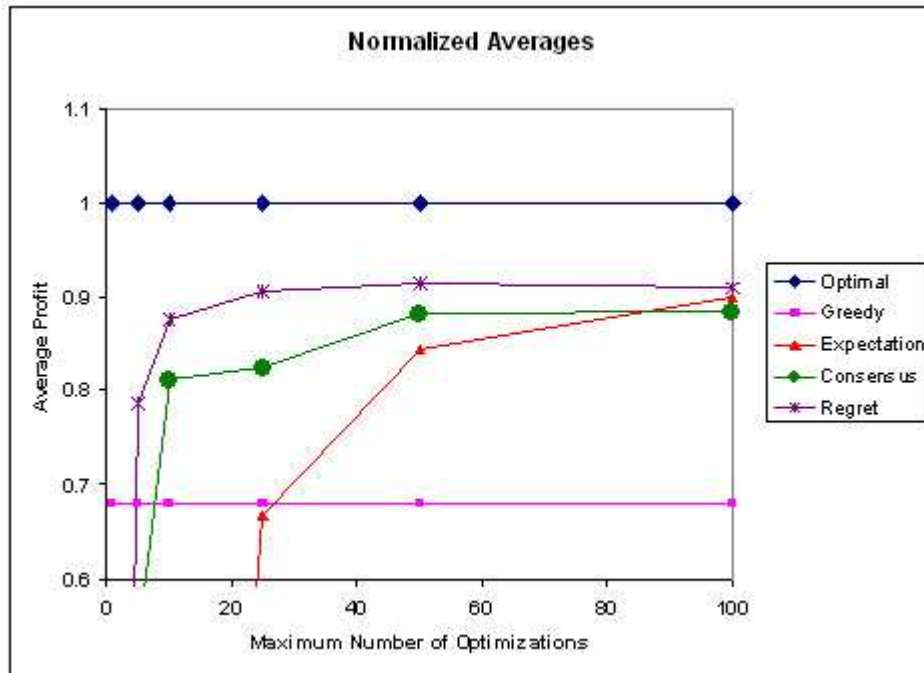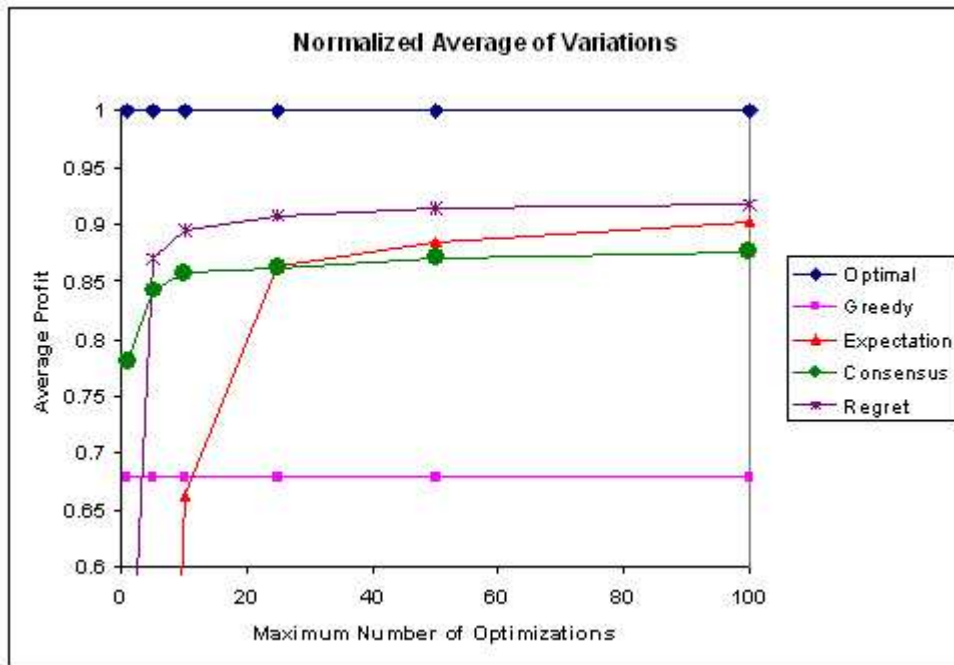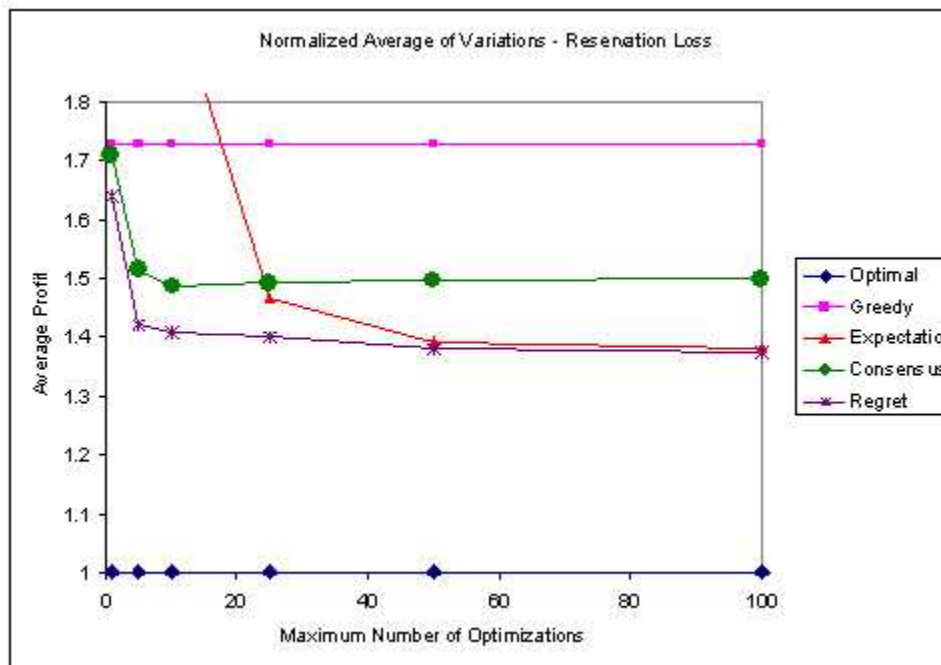
(a) Average Profit
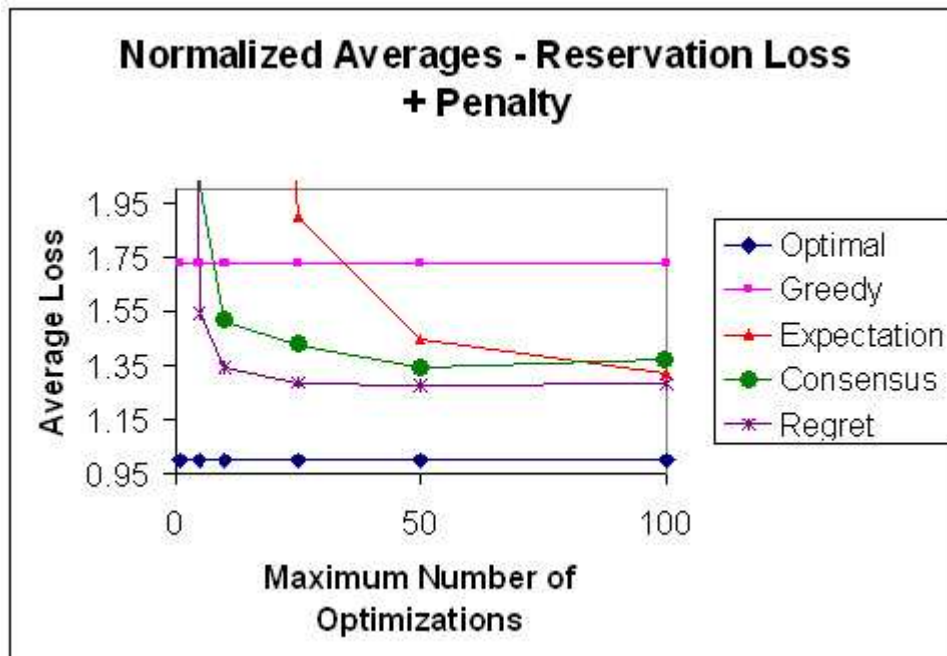
(b) Average Loss

Figure 10: Experimental Results over All Instances with Overbooking Allowed.

(a) Average Profit

(b) Average Loss

Figure 11: Experimental Results over All Instances with Overbooking Disallowed.

clearly dominates the expectation algorithm E which performs poorly for tight time constraints. It becomes reasonable for 50 optimizations and reaches the quality of the regret algorithm for 100 optimizations.

Figure 11 shows the same results when no overbooking is allowed. These instances are easier in the sense that fewer optimizations are necessary for the algorithms to converge. But they exhibit the same pattern as when overbooking is allowed. These results are quite interesting and shows that the benefits of the regret algorithm increase with the problem complexity but are significant even on easier instances.
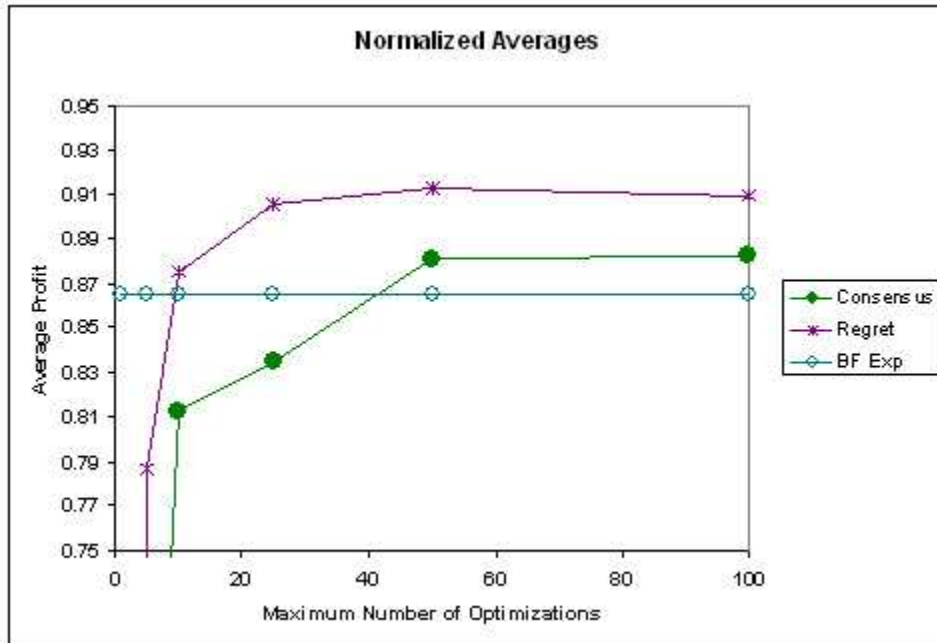
6.3 Comparison with Earlier Results

As mentioned earlier, the best-fit algorithm is only 5% below the optimal offline solution in these problems. It is thus tempting to replace the IP solver in algorithm E by the best-fit heuristic to evaluate more samples. The algorithm, denoted by BF Exp, was proposed in [1] and was shown to be superior to several approaches including yield management and an hybridization with Markov Models [12]. Because the best-fit algorithm is so fast, BF Exp can easily be run with 10,000 samples and remedies the limitations of algorithm E under tight time constraints.

Figure 12 compares algorithms BF Exp, R, and C when overbooking is allowed. The results show that BF Exp indeed produces excellent results but is quickly dominated by R as time increases. In particular, the loss of BF Exp is above 40%, although it goes down to 34% for 10 optimizations and 28% for 25 optimizations in algorithm R. Similarly, the profit increases by 4% in the average starting at 25 optimizations. BF Exp is also dominated by algorithm C but only for 50 optimizations or more.

What is quite remarkable here is that the 5% difference in quality between the best-fit heuristic and the offline algorithm translates into a similar difference in quality in the online setting. Moreover, when looking at specific instances, one can see that BF Exp is often comparable to R but its loss (resp. profit) may be significantly higher (resp. lower) on instances that seem particularly difficult. This is the case for instances E and G, where the gap between the offline solutions and the solutions by algorithm R is larger. This seems to indicate that the harder the problems the more beneficial algorithm R becomes. This in fact confirms our earlier results on stochastic vehicle routing where the algorithms use a large neighborhood heuristic [3, 13]. Indeed, using a simpler, lower-quality, heuristic on more samples did not produce high-quality results in an online setting. The results presented here also show that the additional information produced by a more sophisticated solver quickly amortizes its computational cost, making algorithm R particularly effective and robust for many problems.

6.4 The Impact of the IP Model

Figure 13 reports some experimental results on the impact of the IP model. It depicts the distributions of the distibution of ratios online/offline, depicting the maximum, the median, as well as the .75-tile and .25-tile. The minimum ratio does not appear, as it is always lower than .86. Notches represent a 95% confidence interval on the median. The data is obtained on 50 instances based on the master problem (no overbooking) and 20 runs per instances, accounting for 1,000 runs. Figure 13[a] compares the pessimistic multi-knapsack approach where the capacities of the cancelled requests is not restored (noCan) with the multi-period/multi-knapsack approach using model (IP_2) to take into account cancellations exactly. These two approaches are compared on 10, 25, and 50 scenarios per decision using the regret algorithm. The results indicate that the multi-period/multi-knapsack model definitely improves over the pessimistic multi-knapsack approach as the confidence interval around the median do not intersect. The ratio online/offline moves from 92% to 93%, which is not negligible given the fact that the algorithms are already producing very high-quality decisions. Figure 13[b] gives similar results for both the expectation and regret algorithm using 25 scenarios.

(a) Average Profit
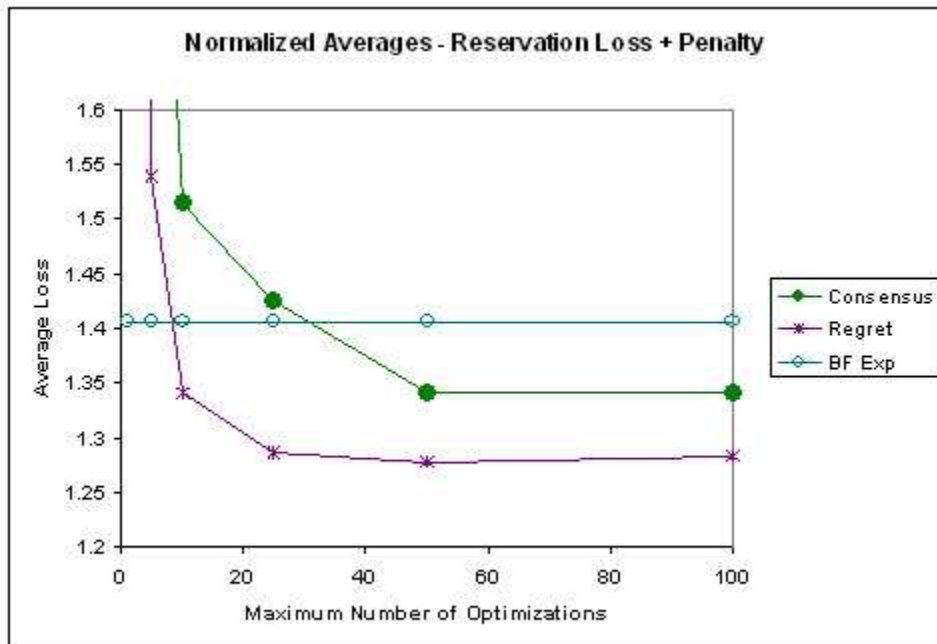
(b) Average Loss

Figure 12: Comparison with Earlier Results: Average Results for Instances with Overbooking

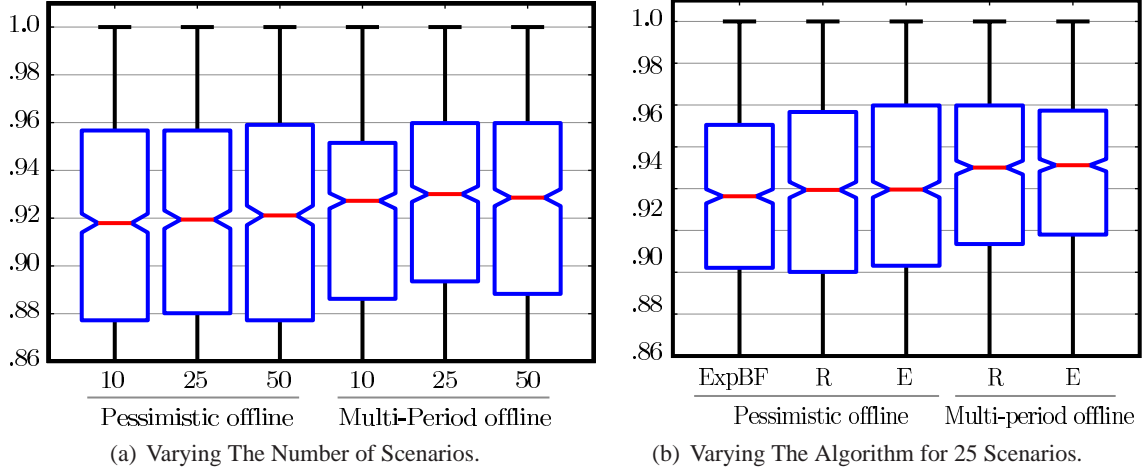(a) Varying The Number of Scenarios.　　　　(b) Varying The Algorithm for 25 Scenarios.

Figure 13: The Impact of the Integer Programming Model.

6.5　The Quality of the Regret Algorithm

Figure 14 reports experimental results on the quality of the regret algorithm. It depicts the frequencies of the differences between the optimal solution and the regret evaluation on all possible bin allocations for all scenarios, both the pessimistic knapsack and the multi-period/multi-knapsack approaches. What the results indicates is that the difference in evaluation is almost always very small, demonstrating experimentally the quality of the regret algorithm. For the pessimistic offline, the regret algorithm produces the optimal value 80% of the time and is at most 5 off the optimal value about 90% of the time. The results are slightly inferior for the multi-period offline, since the regret algorithm has less flexibility. Note that negative differences come from the tolerance used by CPLEX, which is not guaranteed to find the exact optimum. Also the gaps in the histogram are due to reward values: not all the differences between reward values are possible.

Figure 15 compares the quality of the decisions taken by the regret algorithms as a function of the consensus rate, that is the percentage of scenarios whose optimal bin allocation at a time t is the same as the decision taken by the expectation algorithm at time t. The experimental results are for 10 scenarios: They indicate that there is perfect agreement between the scenarios 60% of the time (the rightmost column) and that, 20% of the time, there is a 90% agreement (the next column on the right). The quality of the decisions is measured by the disagreements between algorithms E and R, that is the difference in quality between the decisions taken by algorithms E and R over all scenarios. The experimental results, depicted by the blue curve, show that the disagreements are always very small (less than 0.44 for a consensus rate of 50%) and decrease significantly when the consensus rate increases. This highlights a fundamental property of the regret algorithm: it is optimal for the optimal decision. Hence, when the consensus rate is large, it is optimal for a large number of scenarios and the disagreement decreases.

7　Conclusion

This paper adapted our online stochastic framework and algorithms to the online stochastic reservation problems initially proposed in [1]. These problems, whose core can be modelled as multi-knapsacks, are significant in practice and are also different from the scheduling and routing applications we studied earlier. Indeed the main decision is not which request to select next but rather how best to serve a request given limited resources. The paper shows that the framework and its associated algorithms naturally apply to online reservation systems and it presented a constant-factor sub-optimality approximation of multi-knapsack
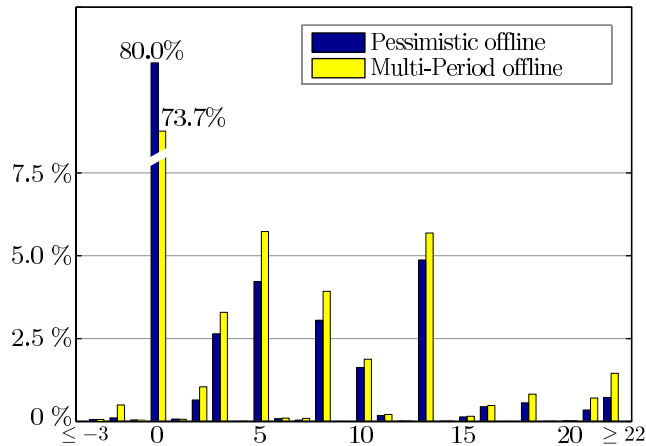
Figure 14: The Quality of the Regret Algorithm.

problems that only solves one-dimensional knapsack problems, leading to a regret algorithm that uses both mathematical programming and dynamic programming algorithms. It also proposed several approaches to deal with cancellations and studied IP models to solve the multi-period/multi-knapsack problem. The algorithms were evaluated on the multi-knapsack problems proposed in [1] with and without overbooking. The results indicate that the regret algorithm is particularly effective, providing significant benefits over heuristic, consensus, and expectation approaches. It also dominates an earlier algorithm proposed in [1] (which applies the best-fit heuristic with algorithm E) as soon as the time constraints allows for 10 optimizations at decision time or between decisions. The experimental results show that the regret algorithm closely approximates the expectation algorithm at a fraction of the cost. Even more interesting perhaps, the regret algorithm has now been applied to online stochastic problems where the offline problem is solved by either constraint programming, integer programming, or (special-purpose) polynomial algorithms, indicating its versatility and benefits for a wide variety of applications.

References

[1] T. Benoist, E. Bourreau, Y. Caseau, and B. Rottembourg. Towards stochastic constraint programming: A study of online multi-choice knapsack with deadlines. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming (CP'01)*, pages 61–76, London, UK, 2001. Springer-Verlag.

[2] R. Bent, I. Katriel, and P. Van Hentenryck. Sub-Optimality Approximation. In *Eleventh International Conference on Principles and Practice of Constraint Programming*, Stiges, Spain, 2005.

[3] R. Bent and P. Van Hentenryck. A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. *Transportation Science*, 8(4):515–530, 2004.

[4] R. Bent and P. Van Hentenryck. Online Stochastic and Robust Optimization. In *Proceeding of the 9th Asian Computing Science Conference (ASIAN'04)*, Chiang Mai University, Thailand, December 2004.
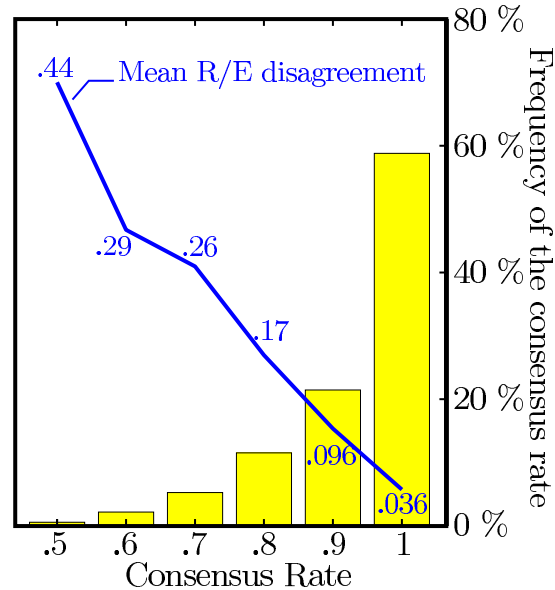
22

Figure 15: The Quality of the Decisions Taken by the Regret Algorithm.

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

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

[7] R. Bent and P. Van Hentenryck. The Value of Consensus in Online Stochastic Scheduling. In *Proceedings of the 14th International Conference on Automated Planning & Scheduling (ICAPS 2004)*, Whistler, British Columbia, Canada, 2004.

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

[9] A. Campbell and M. Savelsbergh. Decision Support for Consumer Direct Grocery Initiatives. *Report TLI-02-09, Georgia Institute of Technology*, 2002.

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

[11] B. Dean, M.X. Goemans, and J. Vondrak. Approximating the Stochastic Knapsack Problem: The Benefit of Adaptivity. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 208–217, Rome, Italy, 2004.

[12] M. Puterman. *Markov Decision Processes*. John Wiley & Sons, New York, 1994.

[13] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Proceedings of Fourth International Conference on the Principles and Practice of Constraint Programming (CP'98)*, pages 417–431, Pisa, October 1998.