

Online Stochastic Optimization Without Distributions

Russell Bent and Pascal Van Hentenryck

Brown University
Providence, RI 02912
{rbent,pvh}@cs.brown.edu

Abstract

This paper considers online stochastic scheduling problems where time constraints severely limit the number of optimizations which can be performed at decision time and/or in between decisions. Prior research has demonstrated that, whenever a distribution of the inputs is available for sampling, online stochastic algorithms may produce significant improvements in solution quality over oblivious approaches. However, the availability of an input distribution, although reasonable in many contexts, is too strong a requirement in a variety of applications. This paper broadens the applicability of online stochastic algorithms by relaxing this requirement and using machine learning techniques or historical data instead. In particular, it shows that machine learning techniques can be engineered to learn the distribution online, when its underlying structure is not available. Moreover, the paper presents the idea of historical sampling which provides a simple and effective way to leverage historical data in continuous and periodic online optimization. Experimental results on packet scheduling and vehicle routing indicate the potential of machine learning and historical sampling for online scheduling.

Introduction

Online scheduling and routing problems arise naturally in many application areas and have received increasing attention in recent years. Contrary to offline optimization, the data is not available a priori in online optimization. Rather it is incrementally revealed during algorithm execution. In many online optimization problems, the data is a set of requests (e.g., packets in network scheduling or customers in vehicle routing) which are revealed over time and the algorithm must decide which request to process next. In addition, time constraints typically restrict the number of optimizations that can be performed at decision time and/or in between decisions. Online problems of this kind arise in many applications, including vehicle routing, taxi dispatching, packet scheduling, and online deliveries. It is also useful to distinguish two classes of such problems: *continuous* and *periodic* optimization. In continuous optimization, the online

algorithm is applied continuously or, at least, for very long periods of time, as is typical in networking applications. In periodic optimization, the online algorithm is applied repeatedly (e.g., every day) on a new instance, as is typically the case in vehicle routing applications arising from courier services.

Recent research (e.g., (Chang, Givan, & Chong 2000; Bent & Van Hentenryck 2004b; 2004c; 2004d)) has demonstrated the value of stochastic information for online (continuous and periodic) optimization. By assuming the availability of a distribution of future requests, or an approximation thereof, online stochastic optimization algorithms can be designed to improve the quality of the solutions significantly, while making decisions within the time constraints. Moreover, these algorithms only rely on the assumption that the input distribution is a black box that can be sampled, not on an intimate knowledge of the distribution.

The availability of a distribution that can be sampled is a reasonable assumption in many contexts where predictive models are available. However, for many other applications, it may be unrealistic. This paper studies whether this assumption can be relaxed and it focuses primarily on continuous online optimization, although issues pertinent to periodic online optimization are addressed whenever relevant. To relax the assumption, the paper explores two orthogonal directions: machine learning (ML) and historical data (HD). The ML approach consists of learning the distribution on the fly, during the online algorithm, given some partial knowledge of its structure. The main contribution here is to adapt traditional techniques from machine learning, such as belief states and the Baum-Welch algorithm, to learn the distribution fast enough under strict time constraints. The HD approach amounts to exploiting historical data gathered from past instances (periodic optimization) or earlier inputs (continuous optimization). The main contribution in the HD approach is to present a novel technique, *historical sampling*, which relies only on past or earlier inputs and assumes no knowledge of the distribution. The only assumption underlying historical sampling is that the past is a reasonable predictor of the future.

The two approaches have been evaluated on packet scheduling and vehicle routing applications, two applications where the value of stochastic information has been clearly demonstrated. In packet scheduling, an application

initially proposed in (Chang, Givan, & Chong 2000), the distributions are specified in terms of Markov Models (MM) while, in vehicle routing, only historical data from previous days is available. The experimental results are particularly interesting. On packet scheduling problems, the ML approach can be engineered to be sufficiently fast and precise so that online algorithms become essentially comparable when using the exact or the learned distributions. Moreover, historical sampling, which is amazingly simple to implement, is also shown to be comparable with the exact distribution. The value of historical sampling is also demonstrated on vehicle routing, confirming its effectiveness of both continuous and periodic applications. Interestingly, historical sampling remains effective even if the historical data is relatively small.

As a consequence, the paper demonstrates that online stochastic optimization algorithms can be effective even without distributions, broadening their applicability significantly thanks to the use of machine learning or the exploitation of historical data.

The rest of this paper is organized as follows. The first three sections present the online stochastic framework, the online stochastic algorithms, the sampling algorithm when the distribution is known. The next three sections constitute the core of the paper. They describe the ML and HD approaches and their experimental results. The last section concludes the paper.

The Online Stochastic Framework

This section presents the online stochastic framework in its simplest form to crystallize the ideas. Its generalizations are described in detail in (Bent & Van Hentenryck 2004a).

The Offline Problem The framework assumes a discrete model of time and the offline problem considers a time horizon $H = [\underline{H}, \overline{H}]$ and a number of requests R . Each request r is associated with a weight $w(r)$ which represents the gain if the request is served. A solution to the offline problem serves a request r at each time $t \in H$ and can be viewed as a function $H \rightarrow R$. Solutions must satisfy the problem-specific constraints which are left unspecified in the framework. The goal is to find a feasible solution σ maximizing

$$\sum_{t \in H} w(\sigma(t)).$$

In the online version, the requests are not available initially and become progressively available at each time step. Note that the framework assumes that a decision is taken at each time step but it is not difficult to relax this assumption (Bent & Van Hentenryck 2004a).

The Online Problem The online algorithms have at their disposal a procedure to solve, or approximate, the offline problem, as well as the distribution of future requests. The distribution is considered as a black-box which is only available for sampling. Since, on continuous applications, it is not practical to sample the distribution for the entire time

```

ONLINEOPTIMIZATION( $H$ )
1  $R \leftarrow \emptyset$ ;
2  $w \leftarrow 0$ ;
3 for  $t \in H$ 
4 do  $\Theta(t) \leftarrow \text{NEWREQUESTS}(t)$ ;
5    $R \leftarrow \text{AVAILABLEREQUESTS}(R, t) \cup \Theta(t)$ ;
6    $r \leftarrow \text{CHOOSEREQUEST}(R, t)$ ;
7    $\text{SERVEREQUEST}(r, t)$ ;
8    $w \leftarrow w + w(r)$ ;
9    $R \leftarrow R \setminus \{r\}$ ;

```

Figure 1: The Generic Online Algorithm

horizon, the desired size of the samples is a sampling argument.

Time Constraints 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 offline procedure \mathcal{O} times at each time step.

Properties of the Framework The framework is general enough to model a variety of practical applications, yet it has some fundamental computational advantages compared to other models. *The key observation is that, in many practical applications, the uncertainty does not depend on the decisions.* There is no need to explore sequences of decisions and/or trees of scenarios: the distribution can be sampled to provide scenarios of the future without considering the decisions. As a consequence, the framework provides significant computational advantages over more general models such as multi-stage stochastic programming (Birge & Louveaux 1997), which is an offline framework, and partially observable Markov decision processes (POMDPs) (Kaelbling, Littman, & Cassandra 1998), where the uncertainty can depend on the decisions. As mentioned earlier, the framework can be naturally extended to incorporate service commitments, multiple decisions, and immediate decision making (Bent & Van Hentenryck 2004a).

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 CHOOSEREQUEST. The online optimization schema simply considers the set of available and new requests at each time step and chooses a request r which is then served and removed from the set of available requests. Function AVAILABLEREQUEST(R, t) returns the set of requests available for service at time t and function SERVEREQUEST(r, t) simply serves r at time t ($\sigma(t) \leftarrow r$). To implement function CHOOSEREQUEST, the algorithms have at their disposal two black-boxes:

1. a function OPTSOL(R, t) that, given a set R of requests and a time t , returns an optimal solution for R over $[t, \infty]$;
2. a function GETSAMPLE($[t, \Delta]$) that returns a set of requests over the interval $[t, t + \Delta]$ by sampling the arrival distribution. The implementation of this black box is the focus of the work presented here.

To illustrate the framework, we specify two oblivious algorithms as instantiations of the generic algorithm. These algorithms will serve as a basis for comparison.

Greedy (G): This algorithm serves the available request with highest weight. It can be specified formally as

```
CHOOSEREQUEST-G( $R, t$ )
1  return  $\operatorname{argmax}(r \in R) w(r)$ ;
```

Local Optimal (LO): This algorithm chooses the next request to serve at time t by finding the optimal solution for the available requests at t . It can be specified as

```
CHOOSEREQUEST-LO( $R, t$ )
1   $\sigma \leftarrow \operatorname{OPTSOL}(R, t)$ ;
2  return  $\sigma(t)$ ;
```

Online Stochastic Algorithms

This section reviews the various online stochastic algorithms considered in this paper. It starts with the expectation algorithm, proposed for instance in (Chang, Givan, & Chong 2000), and shows how it can be adapted to incorporate time constraints.

Expectation (E): Algorithm E chooses the action maximizing expectation at each time step. Informally speaking, the method generates future requests by sampling and evaluates each available request against that sample. A simple implementation can be specified as follows:

```
CHOOSEREQUEST-E( $R, t$ )
1  for  $r \in R$ 
2    do  $f(r) \leftarrow 0$ ;
3  for  $i \leftarrow 1 \dots \mathcal{O}/|R|$ 
4    do  $S \leftarrow R \cup \operatorname{GETSAMPLE}([t + 1, \Delta])$ ;
5    for  $r \in R$ 
6      do  $\sigma \leftarrow \operatorname{OPTSOL}(S \setminus \{r\}, t + 1)$ ;
7       $f(r) \leftarrow f(r) + w(r) + w(\sigma)$ ;
8  return  $\operatorname{argmax}(r \in R) f(r)$ ;
```

Lines 1-2 initialize the evaluation function $f(r)$ for each request r . The algorithm then generates a number of samples for future requests (line 3). For each such sample, it computes the set R of all available and sampled requests at time t (line 4). The algorithm then considers each available request r successively (line 5), it implicitly schedules r at time t , and applies the optimal offline algorithm using $S \setminus \{r\}$ and the time horizon (line 6). The evaluation of request r is updated in line 7 by incrementing it with its weight and the score of the corresponding optimal offline solution. All scenarios are evaluated for all available requests and the algorithm then returns the request $r \in R$ with the highest evaluation. Observe Line 3 of Algorithm E which distributes the available optimizations \mathcal{O} across all available requests R .

When \mathcal{O} is small (due to the time constraints), each request is only evaluated with respect to a small number of samples and the algorithm E does not yield much information. This is precisely why online vehicle routing algorithms (Bent & Van Hentenryck 2004c) cannot use E, since the

number of requests is very large (about 50 to 100), the time between decisions is relatively short, and optimizations are computationally demanding. To cope with time constraints, two approximations of algorithm E, consensus and regret, have been proposed.

Consensus (C): The consensus algorithm C was introduced in (Bent & Van Hentenryck 2004d) as an abstraction of the sampling method used in online vehicle routing (Bent & Van Hentenryck 2004c). Its key idea is to solve each scenario once and thus to examine \mathcal{O} scenarios instead of $\mathcal{O}/|R|$. More precisely, instead of evaluating each possible request at time t with respect to each sample, algorithm C executes the offline algorithm on the available and sampled requests once per sample. The request scheduled at time t in optimal solution σ is credited $w(\sigma)$ and all other requests receive no credit. Algorithm C can be specified as follows:

```
CHOOSEREQUEST-C( $R, t$ )
1  for  $r \in R$ 
2    do  $f(r) \leftarrow 0$ ;
3  for  $i \leftarrow 1 \dots \mathcal{O}$ 
4    do  $S \leftarrow R \cup \operatorname{GETSAMPLE}([t + 1, \Delta])$ ;
5     $\sigma \leftarrow \operatorname{OPTSOL}(S, t)$ ;
6     $f(\sigma(t)) \leftarrow f(\sigma(t)) + w(\sigma)$ ;
7  return  $\operatorname{argmax}(r \in R) f(r)$ ;
```

Observe line 5 which calls the offline algorithm with all available and sampled requests and a time horizon starting at t and line 6 which increments the number of times request $\sigma(t)$ is scheduled first. Line 7 simply returns the request with the largest score. The main appeal of Algorithm C is its ability to avoid partitioning the available samples between the requests, which is a significant advantage when the number of samples is small and/or when the number of requests is large. Its main limitation is its *elitism*. Only the best request is given some credit for a given sample, while other requests are simply ignored. It ignores the fact that several requests may be essentially similar with respect to a given sample. Moreover, it does not recognize that a request may never be the best for any sample, but may still be extremely robust overall.¹

Regret (R): The regret algorithm shows how to gather that kind of information from the sample solutions without solving additional optimization problems. Its key idea is to approximate the deviation of a request, i.e., the cost of scheduling a suboptimal request at time t .

Definition 1 (Deviation) Let R be the set of requests at time t and $r \in R$. The deviation of r wrt R and t , denoted by $\operatorname{DEVIATION}(r, R, t)$, is defined as

$$|w(\operatorname{OPTSOL}(R, t)) - (w(r) + w(\operatorname{OPTSOL}(R \setminus \{r\}, t + 1)))|.$$

¹The consensus algorithms behaves very well on many vehicle routing applications because, on these applications, the objective function is first to serve as many customers as possible. As a consequence, at a time step t , the difference between the optimal solution and a non-optimal solution is rarely greater than 1. It is over time that significant differences between the algorithms accumulate.

Algorithm R is the recognition that, in many applications, it is possible to estimate the deviation of a request r at time t quickly. In other words, once the optimal solution σ of a scenario is computed, it is easy to compute the deviation of all the requests, thus approximating E with one optimization. This intuition can be formalized using the concept of *regret*.

Definition 2 (Regret) Given a request r , a set R ($r \in R$), a time t , and an optimal solution $\sigma = \text{OPTSOL}(R, t)$, a regret is an upper bound to the deviation of r wrt R and t , i.e.,

$$\text{REGRET}(r, R, t, \sigma) \geq \text{DEVIATION}(r, R, t).$$

Moreover, there exist two functions f_o and f_r such that

- $\text{OPTSOL}(R, t)$ runs in time $O(f_o(R, \Delta))$;
- $\text{REGRET}(r, R, t, \sigma)$ runs in time $O(f_r(R, \Delta))$;
- $|R|f_r(R, \Delta)$ is $O(f_o(R, \Delta))$.

Intuitively, the complexity requirement states that the computation of the $|R|$ regrets does not take more time than the optimization. Regrets typically exist in practical applications. In an online facility location problem, the regret of opening a facility f can be estimated by evaluating the cost of closing the selected facility $\sigma(t)$ and opening f . In vehicle routing, the regret of serving a customer c next can be evaluated by swapping c with the first customer on the vehicle serving c . In packet scheduling, the regret of serving a packet p can be estimated by swapping and/or serving a constant number of packets. In all cases, the cost of computing the regret is small compared to the cost of the optimization and satisfy the above requirements. Note that there is an interesting connection to local search, since computing the regret may be viewed as evaluating the cost of a local move for the application at hand. We are now ready to present the regret algorithm R:

```

CHOOSEREQUEST-R( $R, t$ )
1  for  $r \in R$ 
2    do  $f(r) \leftarrow 0$ ;
3  for  $i \leftarrow 1 \dots \mathcal{O}$ 
4    do  $S \leftarrow R \cup \text{GETSAMPLE}([t + 1, \Delta])$ ;
5        $\sigma \leftarrow \text{OPTSOL}(S, t)$ ;
6        $f(\sigma(t)) \leftarrow f(\sigma(t)) + w(\sigma)$ ;
7       for  $r \in \text{READY}(R, t) \setminus \{\sigma(t)\}$ 
8         do  $f(r) \leftarrow f(r) + (w(\sigma) - \text{REGRET}(\sigma, r, R, t))$ ;
9  return  $\text{argmax}(r \in R) f(r)$ ;

```

Its basic organization follows algorithm C. However, instead of assigning some credit only to the request selected at time t for a given sample s , algorithm R (lines 7-8) uses the regrets to compute, for each available request r , an approximation of the best solution of s serving r at time t , i.e., $w(\sigma) - \text{REGRET}(\sigma, r, R, t)$. Hence every available request 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 .

Sampling

To describe the stochastic algorithms entirely, it is necessary to specify function GETSAMPLE , i.e., how to gener-

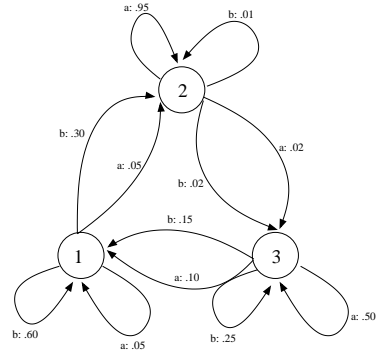


Figure 2: An Example of Markov Model

ate samples from the distribution. To make the presentation concrete, this paper assumes that the underlying distribution of the requests is a Markov model (Feller 1957), although the principles can easily be adapted to a variety of distributions. In addition, the paper assumes for simplicity that the structure of the Markov model directly reflects the uncertainty present in the online stochastic optimization framework. More precisely, each state $s \in S$ in the Markov model represents different arrival probabilities for all the request types and the transitions $T(s_1, s_2)$ between states s_1 and s_2 represent changes in the arrival rates of the request types. Finally, the transition function T is augmented by a third parameter specifying the set of requests generated by a transition. Thus $T(s_1, s_2, W)$ represents the probability of going from s_1 to s_2 and producing output W . Observe that a transition (possibly a self-transition) is taken at every time step t in order to generate the requests. Figure 2 provides an example of such a Markov model, with two possible outputs, the set of requests a and the set of requests b .

Given these assumptions, it is easy to specify function GETSAMPLE for fully observable distributions:

```

GETSAMPLE-FO( $t, \Delta$ )
1  return  $\text{RANDOMWALK}(S_t, \Delta)$ ;

```

The algorithm simply performs a random walk of Δ steps in the Markov model beginning at the state observed at time t . The random walk is specified as

```

RANDOMWALK( $s, \Delta$ )
1   $R \leftarrow \emptyset$ ;
2  for  $i \leftarrow 1 \dots \Delta$ 
3    do  $\langle s, s', W \rangle \leftarrow \text{SELECTRANDOMTRANSITION}(s, T)$ ;
4        $R \leftarrow R \cup W$ ;
5        $s \leftarrow s'$ ;
6  return  $R$ ;

```

where lines 2-5 walk through the Markov model for Δ steps. Line 3 chooses a transition at random based on the current state and the transition function. Line 4 stores the requests generated on that transition. The requests generated during the random walk are returned in line 6.

Learning the Distribution Online

It is unrealistic in practice to assume a fully observable distribution as input to the online optimization algorithm. As a consequence, this section investigates how to relax this assumption by using machine learning techniques. It starts by relaxing full observability before considering the case where the distribution parameters are also unknown.

Partially Observable Distributions Even if a precise distribution is available for the requests, it is unreasonable to assume that function GETSAMPLE knows the current state of the Markov model. Partial observability is a much more realistic assumption adopted in many uncertainty frameworks (e.g., (Kaelbling, Littman, & Cassandra 1998)). Under partial observability, the state S_t is unknown and can only be inferred through observations, that is through the set of requests generated at each time step.

To infer the current state, the sampling algorithm makes use of belief states, a fundamental idea from POMDP research. More precisely, the algorithm maintains, for each state $s \in S$, a probability $B_t(s)$ of being in state s at time t and updates these probabilities after every observation. An implementation of sampling under partial observability is as follows:

```

GETSAMPLE-PO( $t, \Delta$ )
1  for  $s \in S$ 
2    do  $B_t(s) \leftarrow \text{PR}(s|\Theta(t), B_{t-1})$ ;
3   $s \leftarrow \text{RANDOM}(S, B_t)$ ;
4  return  $\text{RANDOMWALK}(s, \Delta)$ ;

```

where $\text{RANDOM}(S, B)$ selects a request s in S with probability $B(s)$. Line 2 updates the belief probabilities by conditioning the probability of being in state s at time t on the belief states at time $t - 1$ and the requests observed at time t . Line 3 chooses an initial state randomly using the belief probabilities and line 4 returns a set of requests based on a random walk beginning at state s .

Structural Model of the Distribution In many circumstances, even partial observability is too strong a requirement. In particular, the structure of the distribution may be known but not its parameters. To infer these parameters, the implementation of GETSAMPLE may resort to the Baum-Welch algorithm (Baum 1972; Charniak 1993) for learning the parameters of a Markov model from a sequence of requests. This learning algorithm starts by initializing the transition probabilities arbitrarily. It then uses the existing probabilities and the training sequence to refine the probabilities. The process is repeated with the new probabilities until the algorithm converges. The algorithm

```

GETSAMPLE-PO-TRAIN( $t, \Delta$ )
1  while Converging
2    do for  $s \in S$ 
3      do  $\alpha_s(0) = B_{t-\lambda}(s)$ ;
4        for  $i \leftarrow 1 \dots \lambda$ 
5          do  $\alpha_s(i) \leftarrow \sum_{s' \in S} \alpha_{s'}(i-1)T(s', s, \Theta(t-\lambda+i))$ ;
6           $\beta_s(\lambda) = B_t(s)$ ;
7        for  $i \leftarrow (\lambda-1) \dots 0$ 

```

```

8          do  $\beta_s(i) \leftarrow \sum_{s' \in S} \beta_{s'}(i+1)T(s, s', \Theta(t-i))$ ;
9  for  $s \in S, s' \in S, w \in \Omega$ 
10    do  $\hat{C}(s, s', w) \leftarrow \sum_{t=1}^{\lambda} \alpha_s(t)T(s, s', w)\beta_{s'}(t+1)$ 
11  for  $s \in S, s' \in S, w \in \Omega$ 
12    do  $T(s, s', w) \leftarrow \frac{\hat{C}(s, s', w)}{\sum_{s'' \in S, w' \in \Omega} \hat{C}(s, s'', w')}$ 
13  return  $\text{GETSAMPLE-PO}(t, \Delta)$ ;

```

uses the requests from the last λ time steps in order to train the probabilities. Lines 3-5 compute the probabilities of being in a state s at a time t given the current probabilities and the training sequence up to time t . Similarly, lines 6-8 compute the reverse, i.e. the probability of beginning at state s at a time t given the training sequence from time t to the end. With these probabilities, lines 9-10 compute the probabilistic counts for the number of times each transition is taken. The transition probabilities are then updated in lines 11-12 using these counts. Once the algorithm has converged, line 13 returns a set of requests based on the learned parameters.

A critical drawback of the algorithm is the training time which can be considerable. As a consequence, in the online algorithm, some of the time allocated to optimizations must now be allocated to training. This time tradeoff is discussed at length in the experimental results where an effective approach to balance optimization and learning is proposed for packet scheduling. Note however that the size of the training sequence is critical to obtain reasonable accuracy and efficiency.

Sampling From Historical Data

The previous section assumed that the sampling algorithm has some knowledge of the distribution. This section relaxes this hypothesis even further and only assumes that historical data is available to the online algorithm. In continuous online optimization, this assumption can be formalized by assuming that the (unknown) input distribution is specified by an ergodic process.² In periodic online optimization, this can be viewed as assuming the existence of past instances, which is typically the case in courier services applications. In both cases, the intuition is simply that the past is a good predictor of the future, which is the only reasonable assumption available.

Historical Averaging *Historical averaging* is a simple implementation of GETSAMPLE which looks at the past λ time steps and derives the probability that a request arrives at any subsequent time step. Conceptually, this approximates the overall output probability of the underlying (unknown) distribution. The implementation

```

GETSAMPLE-A( $t, \Delta$ )
1   $R \leftarrow \emptyset$ ;
2  for  $i \leftarrow t - \lambda \dots t, r \in \Theta(i)$ 
3    do  $A(r) \leftarrow A(r) + 1$ ;
4  for  $r \in R$ 
5    do  $P(r) \leftarrow \frac{A(r)}{\lambda}$ ;
6  for  $i \leftarrow 1 \dots \Delta$ 

```

²An ergodic process is a random process in which the time series produced are the same in statistical properties (Chatfield 2004).

```

7   do  $R \leftarrow R \cup \text{RANDOM}(P, R)$ 
8   return  $R$ ;

```

counts the number of times each possible output is observed in the last λ time steps (line 3) and uses these numbers to compute a probability for the next output (line 5). Line 7 generates random requests based on the probabilities of the possible outputs.

Historical Sampling One of the drawbacks of averaging is that it loses the sequence structure (e.g., changes in request arrival rates). *Historical sampling* aims at overcoming this drawback by using past subsequences as samples. Its implementation

```

GETSAMPLE-S( $t, \Delta$ )
1   $t' \leftarrow \text{RANDOM}([0, t - \Delta])$ ;
2   $R \leftarrow \emptyset$ ;
3  for  $i \leftarrow t' \dots t' + \Delta$ 
4    do  $R \leftarrow R \cup \Theta(i)$ 
5  return  $R$ ;

```

is amazingly simple: It randomly selects a position in the past sequence of requests and generates a sample of size Δ from that starting position.

In addition to its conceptual and implementation simplicity, historical sampling is appealing for a variety of reasons. First, and contrary to historical averaging, it captures structural information on the sequence. Second, whenever the underlying distribution is a Markov model, the resulting sequence corresponds to a random walk in the model from a random state. As a consequence, historical sampling implicitly implements a version of the partial observability algorithm where the belief states are approximated by the state frequencies. Third, in the case of periodic optimization, historical sampling simply amounts to using past instances as samples of the (unknown distribution) with the additional benefit that the starting state is known in this case. As a consequence, historical sampling is a simple and effective technique to exploit historical data in continuous and periodic online optimization.

Packet Scheduling

This section reports experimental results on the online packet scheduling problem first studied in (Chang, Givan, & Chong 2000). This networking application is of interest experimentally since (1) the number of requests to consider at each time t is small and (2) the offline algorithm can be solved in polynomial time. As a result, it is possible to evaluate all the algorithms experimentally, contrary to vehicle routing applications where this is not practical. The packet scheduling is also interesting as it features a complex arrival distribution for the packets based on Markov models.

The Offline Problem We are given a set $Jobs$ of jobs partitioned into a set of classes C . Each job j is characterized by its weight $w(j)$, its arrival date $a(j)$, and its class $c(j)$. Jobs in the same class have the same weight (but different arrival times). We are also given a schedule horizon $H = [\underline{H}, \overline{H}]$

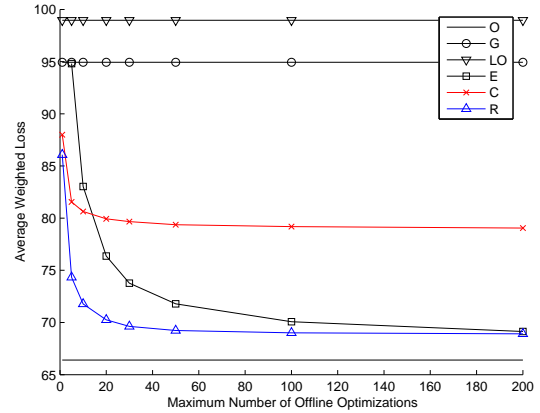


Figure 3: Fully Observable Sampling

during which jobs must be scheduled. Each job j requires a single time unit to process and must be scheduled in its time window $[a(j), a(j) + d]$, where d is the same constant for all jobs (i.e., d represents the time a job remains available to schedule). In addition, no two jobs can be scheduled at the same time and jobs that cannot be served in their time windows are dropped. The goal is to find a schedule of maximal weight, i.e., a schedule which maximizes the sum of the weights of all scheduled jobs. This is equivalent to minimizing weighted packet loss. More formally, assume, for simplicity and without loss of generality, that there is a job scheduled at each time step of the schedule horizon. Under this assumption, a schedule is a function $\sigma : H \rightarrow Jobs$ which assigns a job to each time in the schedule horizon. A schedule σ is feasible if

$$\begin{aligned}
& \forall t_1, t_2 \in H : t_1 \neq t_2 \rightarrow \sigma(t_1) \neq \sigma(t_2) \\
& \forall t \in H : a(\sigma(t)) \leq t \leq a(\sigma(t)) + d.
\end{aligned}$$

The weight of a schedule σ , denoted by $w(\sigma)$, is given by $w(\sigma) = \sum_{t \in H} w(\sigma(t))$. The goal is to find a feasible schedule σ maximizing $w(\sigma)$. This offline problem can be solved in quadratic time $O(|J||H|)$.

The Online Problem The experimental evaluation is based on the problems of (Chang, Givan, & Chong 2000; Bent & Van Hentenryck 2004d), where all the details can be found. In these problems, the arrival distributions are specified by independent MMs, one for each job class similar in form as seen in Figure 2. Thus, each class can be thought of as having its own black box for generating request samples. The results are given for the reference 7-class problems and for an online schedule consisting of 200,000 time steps. Because it is impractical to sample the future for so many steps, the algorithms use a sampling horizon of 50, which seems to be an excellent compromise between time and quality. The regret function is given in (Bent & Van Hentenryck 2004b) and consists of swapping a constant number of packets in the optimal schedule.

Sampling with the Exact Distribution Figure 3 depicts the average packet loss as a function of the number of avail-

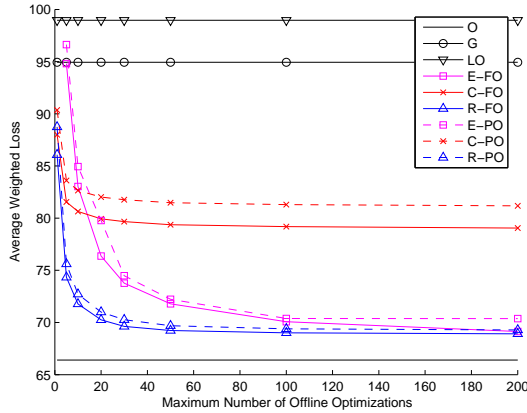


Figure 4: Partially Observable Sampling

able optimizations \mathcal{O} for the various algorithms on a variety of 7-class problems when the Markov Model is fully observable. It also gives the optimal, a posteriori, packet loss O . The results indicate the value of stochastic information as algorithm E significantly outperforms the oblivious algorithms G and LO and bridge much of the gap between these algorithms and the optimal solution. Note that LO is worse than G, illustrating the (frequent) pathological behavior of over-optimization.

Consensus is dominated by E when the number of available optimizations increases, although consensus still produces significant improvements over the oblivious algorithms. This is of course pertinent, since E is not practical for many problems with time constraints. Finally, the benefits of the regret algorithm are clearly apparent. Algorithm R indeed dominates all the other algorithms, including consensus when there are very few optimizations (strong time constraints) and expectation even when there are a reasonably large number of them (weak time constraints). Reference (Bent & Van Hentenryck 2004b) also shows this to be the case for complex online vehicle routing with time windows.

Partial Observability Figure 4 considers the case when the Markov model is partially observable and compares it to the fully observable case. It is interesting to note that there is no significant degradation in solution quality, indicating that maintaining the belief probabilities is sufficient, on this problem, to obtain good results.

Structural Model Figure 5 depicts the experimental results when the MM parameters are learned. In order to get reasonable results, λ is set to 560. The results are comparable to the results if the MM is fully observable. However, these results are misleading, as they do not take into account time spent in training. Figure 6 shows the rough amount of time it takes to train on sequences of a certain size in terms of the equivalent number of optimizations. As can be seen from this plot, training on a sequence of length 560 is roughly equivalent to performing 230 optimizations every time step! If that is taken into account, the results look like

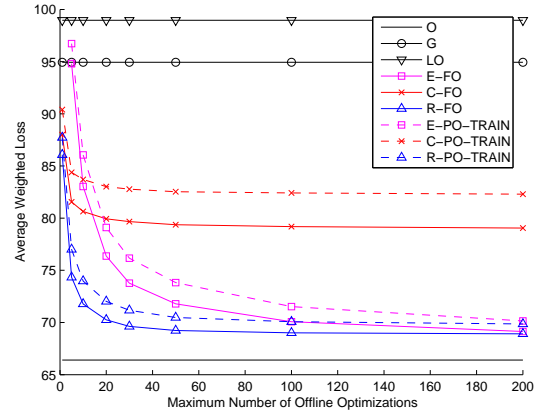


Figure 5: Experimental Results for Packet Scheduling after Learning the MM Parameters

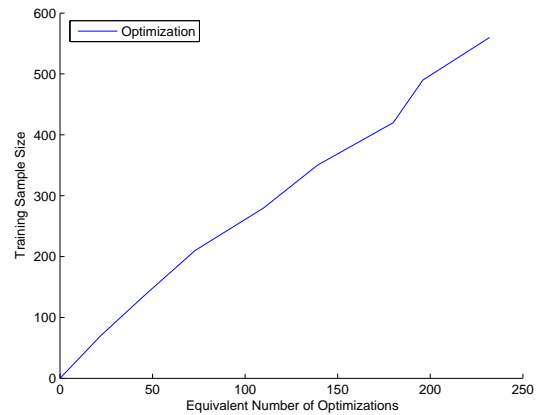


Figure 6: Training vs. Optimization Time Tradeoff

Figure 7. To a certain extent, this limitation can be overcome by reducing the size of the training sequence, i.e., 140 time steps, as depicted in Figure 8, albeit at a loss of overall solution quality. A more promising direction is to apply the learning algorithm periodically (e.g., every 230 steps) instead of at each time steps in order to amortize the cost of training. This is shown in Figure 9, which indicates the feasibility of sophisticated learning techniques in an online setting. Indeed, the loss in solution quality compared to the exact distribution is small and the improvement with respect to oblivious algorithms is significant.

Historical Averaging Figure 10 depicts the experimental results for historical averaging. These are the weakest results presented here, yet this simple method still produces significant improvements over the oblivious algorithms. The results are given for $\lambda = 500$, which achieves the best results experimentally.

Historical Sampling Finally, Figure 11 depicts the experimental results for historical sampling which are very inter-

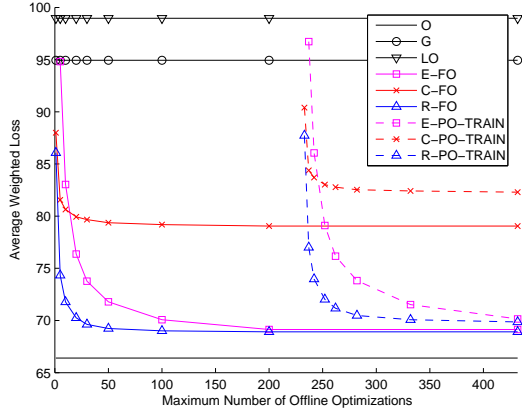


Figure 7: Experimental Results for Packet Scheduling: Including the Learning Time

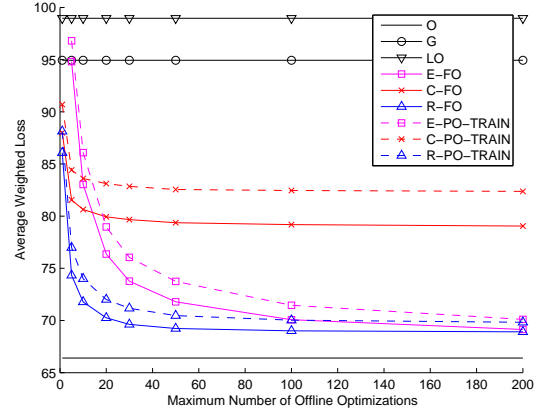


Figure 9: Experimental Results for Packet Scheduling when MM Parameters are Unknown: Balancing Optimization and Training

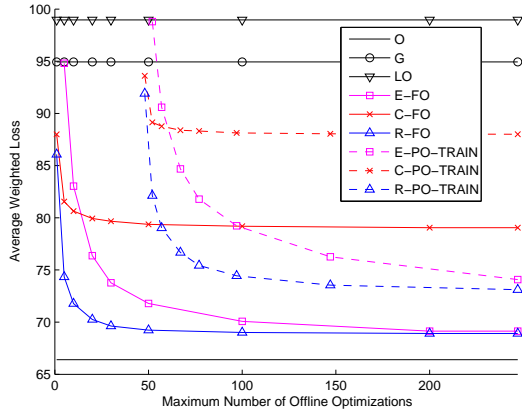


Figure 8: Experimental Results for Packet Scheduling: Including the Learning Time on Sorter Sequences

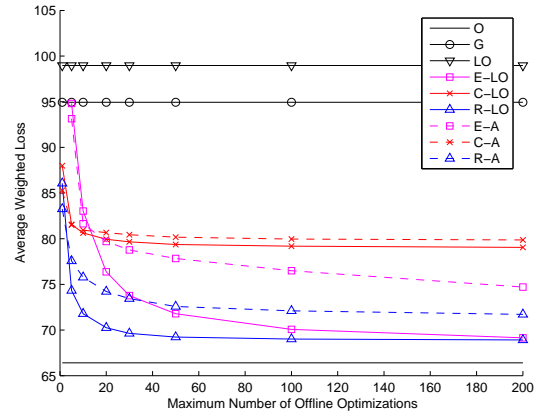


Figure 10: Experimental Results on Packet Scheduling for Historical Averaging

esting. Indeed, historical sampling produces the best results for the most advanced algorithm: the regret algorithm R. (It is also superior for algorithm E). In other words, without any knowledge of the distribution and without inducing any time overhead, historical sampling is as effective as the partial observability model that maintains belief states based on observations.

Vehicle Routing

We now evaluate historical sampling on a significantly more complicated problem: online multiple vehicle routing with time windows. This problem was studied initially in (Bent & Van Hentenryck 2004c) to show the value of stochastic information in vehicle routing. It is a periodic online optimization problems where historical data is typically available.

Problem Formulation The vehicle routing problems are specified formally in (Bent & Van Hentenryck 2004c) where all the details can be found. Each problem contains a depot, a number of customer regions and a number of customer ser-

vice requests from the regions. Each request has a demand, a service time, and a time window specified by an interval $[e, l]$, which represents the earliest and latest possible arrival times respectively. There are a number of identical vehicles available for use, each with capacity Q . A vehicle route starts at the depot, serves some customers at most once, and returns to the depot. The demand of a route is the summation of the demand of its customers. A routing plan is a set of routes servicing each customer exactly once. A solution to the offline VRPTW is a routing plan that satisfies the capacity constraints on the vehicle and the time window constraints of the requests. The objective is to find a solution maximizing the number of served customers. In the dynamic VRPTW, customer requests are not known in advance and become available during the course of the day. In general, a number of requests are available initially.

The online VRPTW is much more complicated than the online packet scheduling and does not fit directly in the stochastic framework presented earlier. First, decisions are triggered by two types of events: new customer requests and

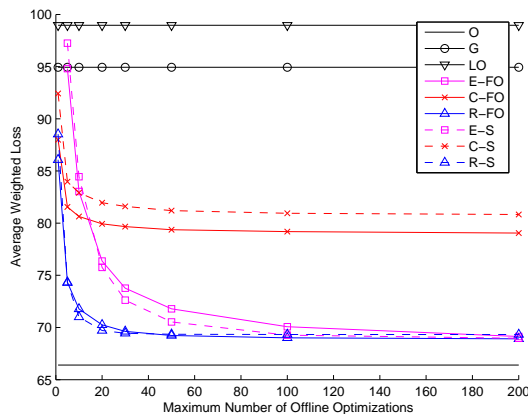


Figure 11: Experimental Results on Packet Scheduling for Historical Sampling

the arrival of a vehicle at a customer site. Second, a decision consists of choosing which customers to serve next on a given vehicle. Third, the online algorithm must decide whether to accept or reject customer requests immediately and must service the accepted requests. Finally, the VRPTW is a hard NP-complete problem whose instances are extremely difficult to solve optimally. Only 2 to 10 optimizations can be solved in between two events and the number of events is large (e.g., 50 different requests). Hence, the expectation method is not practical at all.

Experimental Setting The experimental results are based on the five hardest class-4 problems from (Bent & Van Hentenryck 2004c), where all details can be found. They are derived from the Solomon benchmarks which are very challenging and involve 100 customers. The instances exhibit various degrees of dynamism (i.e., the ratio between known and dynamic customers), different distributions of early and late requests, as well as time windows of very different sizes. Hence they cover a wide spectrum of possibilities and structures. The number of vehicles available for the dynamic algorithms was determined by solving the offline problems and adding two vehicles.

Historical sampling has either 10 or 1000 past samples at its disposal depending on the experiments. For a given time t , the implementation of method GETSAMPLE in this periodic application consists of choosing one of these past samples and returns all the requests occurring after time t . Compared to earlier algorithms which assumed perfect knowledge of the distribution, historical sampling loses two kinds of information. On the one hand, it works from a limited pool of samples and, on the other hand, it loses some dependency information. Indeed, with a perfect distribution, the arrival of actual requests affects the arrival probability of future requests, i.e. the arrival of a request in a region and time period eliminates future requests from that region in the same time period (Bent & Van Hentenryck 2004c).

Experimental Results Table 1 report some preliminary results to evaluate the potential of historical sampling. It depicts the number of missed customers for the consensus (C) and regret (R) algorithms using the actual distribution, as well as the same results when historical sampling is used instead of the actual distribution (HS(C) and HS(R)). Each entry in the table is the average of 20 independent runs with 5 optimizations being allowed in between events. The algorithm all implement the least commitment approach described in (Bent & Van Hentenryck 2004a). Historical sampling was evaluated with a historical data containing 10 and 1000 data points. These preliminary results indicate that historical sampling performs very well on these problems. When the historical data (1000 data points) is large, no significant difference can be observed between historical sampling and the actual distribution. When the historical data is limited (10 data points), the quality of the solutions seems to decrease slightly for consensus, while there is no real difference for regret. The fact that HS^{10} may sometimes outperform HS^{1000} was very puzzling at first sight. A careful analysis indicated that the samples taken were luckily similar to the actual future requests.

Conclusion

This paper focused online stochastic optimization problems where time constraints severely limit the number of optimizations which can be performed at decision time and/or in between decisions. The last couple of years witnessed significant progress in this area, including the design of novel algorithms exploiting stochastic information to make better scheduling and routing decisions. Experimental results have shown that, whenever a distribution of future requests is available for sampling, these algorithms may produce significant improvements in solution quality, while running within the time constraints.

This paper reconsiders the fundamental assumption underlying these algorithms: that a distribution of future requests, or an approximation thereof, is available for sampling. It relaxed that assumption and studied whether the underlying distribution can be learned (when part of its structure is known) or whether historical data can be exploited for sampling purposes. Both continuous and periodic online optimization were considered.

As far as the machine learning approach is concerned, the paper showed that techniques such as belief states and the Baum-Welch algorithm for learning MMs can be engineered to work in an online setting. The experimental results on packet scheduling are very promising: they indicate that the approach preserves most of the benefits of the online algorithms working on the actual distribution.

As far as historical data is concerned, the paper proposed the idea of historical sampling, which consists of exploiting historical data for sampling, i.e., earlier sequences of requests in the case of continuous online optimization and past instances in the case of periodic online optimization. The experimental results, both on packet scheduling (continuous optimization) and vehicle routing (periodic optimization), shows that historical sampling is extremely effective

Problem	C	HS(C) ¹⁰	HS(C) ¹⁰⁰⁰	R	HS(R) ¹⁰	HIS(R) ¹⁰⁰⁰
20-20-60-rc104-1	13.50	16.65	14.30	10.65	10.45	9.45
20-20-60-rc104-2	14.55	15.00	13.60	13.05	11.95	11.70
20-20-60-rc104-3	11.45	12.75	11.60	8.30	9.00	9.25
20-20-60-rc104-4	14.00	14.10	15.40	9.40	9.35	10.25
20-20-60-rc104-5	12.80	14.95	16.00	9.90	9.70	8.40

Table 1: Distribution and Historical Sampling

and produces the same solution quality as the actual distribution.

These results clearly boost the potential of online stochastic optimization, since they significantly weaken their main assumption. In addition to cases where predictive models are available, they also show that online stochastic algorithms can be applied under the weak assumption that the future will resemble the past. In other words, the online algorithms only require the existence of an underlying, unknown, ergodic process (continuous optimization) or the availability of historical data which are representative of what the future will hold.

This research also opens up many avenues for future research. In particular, it would be interesting to generalize the machine learning approach to other classes of distributions. It also seems important to measure the sensitivity of historical sampling on the quality of the historical data. Finally, the quality of the results could also be improved by filtering the historical data to reflect the online observations. Indeed, it is likely that, in many periodic problems, the historical data can be classified into different categories each of which modeling different types of periods. Recognizing these categories and restricting the sampling to the relevant ones may give the algorithm more precise predictions.

Acknowledgements

This research is partially supported by NSF ITR Award DMI-0121495. We would also like to thank Tom Dean for his helpful pointers on machine learning and the reviewers for their interesting comments and suggestions.

References

- Baum, E. 1972. An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of a Markov Process. *Inequalities* 3:1–8.
- Bent, R., and Van Hentenryck, P. 2004a. Online Stochastic and Robust Optimization. In *Proceedings of the Ninth Asian Computing Science Conference (ASIAN)*, 286–300.
- Bent, R., and Van Hentenryck, P. 2004b. Regrets Only! Online Stochastic Optimization Under Time Constraints. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*, 501–506.
- Bent, R., and Van Hentenryck, P. 2004c. Scenario-Based Planning for Partially Dynamic Vehicle Routing with Stochastic Customers. *Operations Research* 52(6), 977–987.

Bent, R., and Van Hentenryck, P. 2004d. The Value of Consensus in Online Stochastic Scheduling. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS)*, 219–226.

Birge, J., and Louveaux, F. 1997. *Introduction to Stochastic Programming*. Springer Verlag.

Chang, H.; Givan, R.; and Chong, E. 2000. Online Scheduling Via Sampling. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 62–71.

Charniak, E. 1993. *Statistical Language Learning*. MIT Press.

Chatfield, C. 2004. *The Analysis of Time Series: An Introduction*. Chapman and Hall.

Feller, W. 1957. *An Introduction to Probability Theory and its Applications*. John Wiley and Sons.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* 101 (1-2):99–124.